



Scalable Vector Graphics (SVG) Tiny 1.2 Specification

W3C Working Draft 08 December 2005

This version:

<http://www.w3.org/TR/2005/WD-SVGMobile12-20051207/>

Previous version:

<http://www.w3.org/TR/2005/WD-SVGMobile12-20050413/>

Latest version:

<http://www.w3.org/TR/SVGMobile12/>

Editors:

Ola Andersson (Ikivo) <ola.andersson@ikivo.com>
Robin Berjon (Expway) <robin.berjon@expway.fr>
Jon Ferraiolo (Adobe Systems) <jon.ferraiolo@adobe.com>
Vincent Hardy (Sun Microsystems, Inc.) <vincent.hardy@sun.com>
Scott Hayman (Research In Motion Limited)
[Dean Jackson](#) (W3C) <dean@w3.org>
[Chris Lilley](#) (W3C) <chris@w3.org>
Craig Northway (Canon, Inc.) <craign@cisra.canon.com.au>
Antoine Quint (Invited Expert) <aq@fuchsia-design.com>

Authors:

See [author list](#)

Copyright ©2005 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use](#) rules apply.

Abstract

This specification defines the features and syntax for Scalable Vector Graphics (SVG) Tiny, Version 1.2, a modularized language for describing two-dimensional vector and mixed vector/raster graphics in XML. SVG Tiny 1.2 is the baseline profile of SVG, implementable on a range of devices from cellphones and PDAs to desktop and laptop computers, and is the core of SVG 1.2. Other SVG 1.2 specifications will extend this functionality to form supersets (for example, SVG 1.2 Full).

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at <http://www.w3.org/TR/>.

This is a W3C Last Call Working Draft. If the feedback is positive, the [SVG Working Group](#) plans to submit this specification for consideration as a W3C Candidate Recommendation. This is the third Last Call Working Draft for this specification. The second Last Call resulted in a number of comments which all have been addressed by the [SVG](#)

[Working Group](#), a list of all comments can be found in the [Last Call Comments List](#). The purpose of this third Last Call is to allow reviewers to verify that their comments have been included as agreed by the [SVG Working Group](#), it is not intended as a new, general review period. Comments for this specification should have a subject starting with the prefix '[SVGMobile12]' followed by the number of the comment from the [Last Call Comments List](#). E.g. "[SVGMobile12], SVGT12-4, subject here". Please send them to www-svg@w3.org, the public email list for issues related to SVG. This list is [archived](#) and acceptance of this archiving policy is requested automatically upon first post. To subscribe to this list send an email to www-svg-request@w3.org with the word subscribe in the subject line. Comments are accepted until 28 December 2005.

This document has been produced by the [SVG Working Group](#) as part of the W3C [Graphics Activity](#), following the procedures set out for the W3C [Process](#). The authors of this document are listed at the end in the [Author List](#) section.

The patent policy for this document is the [5 February 2004 W3C Patent Policy](#). Patent disclosures relevant to this specification may be found on the [SVG Working Group's patent disclosure page](#). An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

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

Available languages

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

Table of Contents

- [1 Introduction](#)
- [2 Concepts](#)
- [3 Rendering Model](#)
- [4 Basic Data Types and Color Keywords](#)
- [5 Document Structure](#)
- [6 Styling](#)
- [7 Coordinate Systems, Transformations and Units](#)
- [8 Paths](#)
- [9 Basic Shapes](#)
- [10 Text](#)
- [11 Painting: Filling, Stroking, Colors and Paint Servers](#)
- [12 Multimedia](#)
- [13 Interactivity](#)
- [14 Linking](#)
- [15 Scripting](#)
- [16 Animation](#)
- [17 Fonts](#)
- [18 Metadata](#)
- [19 Extensibility](#)
- Appendix A [The SVG Micro DOM \(uDOM\)](#)
- Appendix B [IDL Definitions](#)
- Appendix C [Implementation Requirements](#)
- Appendix D [Conformance Criteria](#)
- Appendix E [Conformance to QA Framework Specification Guidelines](#)
- Appendix F [Accessibility Support](#)
- Appendix G [Internationalization Support](#)
- Appendix H [JPEG Support](#)
- Appendix I [Minimizing SVG File Sizes](#)
- Appendix J [Feature strings](#)
- Appendix K [Element Table](#)

- Appendix L [Attribute and Property Tables](#)
- Appendix M [Media Type registration for image/svg+xml](#)
- Appendix N [RelaxNG Schema for SVG Tiny 1.2](#)
- Appendix O [References](#)
- Appendix P [Change History](#)

[Full Table of Contents](#)

The authors of the SVG Tiny 1.2 specification are the people who participated in the SVG Working Group as members or alternates.

Authors:

- Ola Andersson, Ikivo
- Phil Armstrong, Corel Corporation
- Henric Axelsson, Ericsson AB
- Selim Balcısoy, Nokia
- Robin Berjon, Expway
- Benoît Bézaire, Itedo (formerly Corel Corporation)
- John Bowler, Microsoft Corporation
- Gordon Bowman, Corel Corporation
- Craig Brown, Canon Information Systems Research Australia
- Mike Bultrowicz, Savage Software
- Tolga Çapın, Nokia
- Milt Capsimalis, Autodesk Inc.
- Mathias Larsson Carlander, Ericsson AB
- Jakob Cederquist, Ikivo
- Suresh Chitturi, Nokia
- Charilaos Christopoulos, Ericsson AB
- Richard Cohn, Adobe Systems Inc.
- Lee Cole, Quark
- Cyril Concolato, Groupe des Ecoles des Télécommunications (GET)
- Don Cone, America Online Inc.
- Alex Danilo, Canon Information Systems Research Australia
- Thomas DeWeese, Eastman Kodak
- David Dodds, Lexica
- Andrew Donoho, IBM
- David Duce, Oxford Brookes University
- Jean-Claude Dufourd, Streamezzo (formerly GET)
- Jerry Evans, Sun Microsystems
- Jon Ferraiolo, Adobe Systems Inc.
- Darryl Fuller, Schema Software
- 藤沢 淳 (FUJISAWA Jun), Canon
- Scott Furman, Netscape Communications Corporation
- Brent Getlin, Macromedia
- Christophe Gillette, Motorola (formerly BitFlash)
- Peter Graffagnino, Apple
- Rick Graham, BitFlash
- Vincent Hardy, Sun Microsystems Inc.
- 端山 貴也 (HAYAMA Takanari), KDDI Research Labs
- Scott Hayman, Research In Motion Limited
- Stephane Heintz, BitFlash
- Lofton Henderson, OASIS
- Ivan Herman, W3C
- Jan Christian Herlitz, ExcOSOFT
- Alan Hester, Xerox Corporation
- Bob Hopgood, RAL (CCLRC)
- Bin Hu, Motorola
- Michael Ingrassia, Nokia
- 石川 雅康 (ISHIKAWA Masayasu), W3C
- Dean Jackson, W3C (*W3C Team Contact*)
- Christophe Jolif, ILOG S.A.
- Lee Klosterman, Hewlett-Packard
- 小林 亜令 (KOBAYASHI Arei), KDDI Research Labs
- Thierry Kormann, ILOG S.A.

- o Yuri Khramov, Schema Software
- o Kelvin Lawrence, IBM
- o Håkon Lie, Opera
- o Chris Lilley, W3C (*Working Group Chair*)
- o Vincent Mahe, France Telecom
- o Philip Mansfield, Schema Software
- o Kevin McCluskey, Netscape Communications Corporation
- o 水口 充 (MINAKUCHI Mitsuru), Sharp Corporation
- o Luc Minnebo, Agfa-Gevaert N.V.
- o Jean-Claude Moissinac, Groupe des Ecoles des Télécommunications (GET)
- o Craig Northway, Canon Information Systems Research Australia
- o Tuan Nguyen, Microsoft Corporation
- o 小野 修一郎 (ONO Shuichiro), Sharp Corporation
- o Lars Piepel, Vodafone
- o Antoine Quint, Fuchsia Design (formerly of ILOG)
- o ●●●●● ●●●● (Nandini Ramani), Sun Microsystems
- o Bruno David Simões Rodrigues, Vodafone
- o 相良 毅 (SAGARA Takeshi), KDDI Research Labs
- o Troy Sandal, Visio Corporation
- o Peter Santangeli, Macromedia
- o Sebastian Schnitzenbaumer, SAP AG
- o Haroon Sheikh, Corel Corporation
- o Brad Sipes, Ikivo
- o Andrew Sledd, Ikivo
- o (Peter Sorotokin), Adobe Systems Inc.
- o Gavriel State, Corel Corporation
- o Robert Stevahn, Hewlett-Packard
- o Timothy Thompson, Eastman Kodak
- o 上田 宏高 (UEDA Hirotaka), Sharp Corporation
- o Rick Yardumian, Canon Development Americas
- o Charles Ying, Openwave Systems Inc.
- o Shenxue Zhou, Quark

Acknowledgments

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

1 Introduction

Contents

- 1.1 [About SVG](#)
- 1.2 [SVG Tiny 1.2](#)
 - 1.2.1 [Modularization](#)
 - 1.2.2 [Element and Attribute collections](#)
 - 1.2.3 [Profiling the SVG specification](#)
- 1.3 [Defining an SVG Tiny 1.2 document](#)
- 1.4 [SVG MIME type, file name extension and Macintosh file type](#)
- 1.5 [Compatibility with Other Standards Efforts](#)
- 1.6 [Definitions](#)

1.1 About SVG

SVG is a language for describing two-dimensional graphics in XML [[XML11](#)]. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), multimedia (such as raster images and video) and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects.

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

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

SVG is a language for rich graphical content. For accessibility reasons, if there is an original source document containing higher-level structure and semantics, it is recommended that the higher-level information be made available somehow, either by making the original source document available, or making an alternative version available in a format which conveys the higher-level information, or by using SVG's facilities to include the higher-level information within the SVG content. For suggested techniques in achieving greater accessibility, see [Accessibility](#).

It is believed that this specification is in accordance with the Web Architecture principles as described in [Web Architecture](#) [AWWW].

1.2 SVG Tiny 1.2

Industry demand, overwhelming support in the SVG working group and requests from the SVG developer community established the need for some form of SVG suited to displaying vector graphics on small devices. Moreover, the mission statement of SVG 1.0 specifically addressed small devices as a target area for vector graphics display. In order to meet these demands the SVG Working Group created a profile specification that was suitable for use on mobile devices as well as on desktops. The [SVG Mobile 1.1 specification](#) addressed that requirement and defined two profiles to deal with the variety of mobile devices having different characteristics in terms of CPU speed, memory size, and color support. The SVG Mobile 1.1 specification defined SVG Tiny (SVGT) 1.1, suitable for highly restricted mobile devices; it also defined a second profile, SVG Basic (SVGB) 1.1, targeted for higher level mobile devices. The major difference between SVG Tiny 1.1 and SVG Basic 1.1 was the absence of scripting and styling in SVG 1.1 Tiny, and thus any requirement to maintain a Document Object Model (DOM). This saved a substantial amount of memory in most implementations.

Experience with SVG Tiny 1.1, which was widely adopted in the industry and shipped as standard on a variety of cellphones, indicated that the profile was a little too restrictive in some areas. Features from SVG 1.1 such as gradients and opacity, were seen to have substantial value for creating attractive content, and were shown to be implementable on cellphones. There was also considerable interest in adding audio and video capabilities, building on the SMIL support in SVG Tiny 1.1.

Advances such as DOM Level 3, which introduces namespace support and value normalization, prompted a second look at the use of programming languages and scripting with SVG Tiny. In conjunction with the Java JSR 226 group [[JSR226](#)], a lightweight interface called the microDOM, or uDOM, was developed. This could be, but need not be, implemented on top of DOM Level 3. With this advance, lightweight programmatic control of SVG (for example, for games or user interfaces) and use with scripting languages, became feasible on the whole range of platforms from cellphones through to desktops. In consequence, there is only a single Mobile profile for SVG 1.2 - SVG Tiny 1.2

This specification defines the features and syntax for [Scalable Vector Graphics \(SVG\) Tiny 1.2](#), the core specification and baseline profile of SVG 1.2. Other SVG specifications will extend this baseline functionality to create supersets (for example, SVG 1.2 Full). The SVG Tiny 1.2 specification adds to SVG Tiny 1.1 features requested by SVG authors, implementors and users; SVG Tiny 1.2 is a superset of SVG Tiny 1.1.

1.2.1 Modularization

This specification describes a collection of abstract modules that provide specific units of functionality. These modules may be combined with each other and with modules defined in other specifications (such as XHTML) to create SVG subset and extension document types that qualify as members of the SVG family of document types. See [Conformance](#) for a description of SVG family documents, and [XHTMLplusMathMLplusSVG](#) for a profile that combines XHTML, MathML and SVG.

Each major section of the SVG specification produces a module named after that section, e.g. "Text Module" or "Structure Module".

1.2.2 Element and Attribute collections

Modules define a named collection of either elements or attributes. These collections are used as a shorthand when describing the set of attributes allowed on a particular element (eg. the "Style" attribute collection) or the set of elements allowed as children of a particular element (eg. the "Shape" element collection). All collections have names that begin with an uppercase character.

When defining a profile, it is assumed that all the element and attribute collections are defined to be empty. That way, a module can redefine the collection as it is included in the profile, adding elements or attributes to make them available within the profile. Therefore, it is not a mistake to refer to an element or attribute collection from a module that is not included in the profile, it simply means that collection is empty.

1.2.3 Profiling the SVG specification

This specification defines the baseline language definition for SVG 1.2. The specification is defined in terms of a series of modules, each of which consists of a collection of language features. The modular definition of the language enables alternate profiles of the SVG language.

The Tiny profile of SVG 1.2 consists of all of the features defined within this specification. As a modularized baseline specification, it is possible for: *superset profiles* (e.g., SVG Full 1.2) which include all of the Tiny profile but add other features to the baseline; *subset profiles*; and *special-purpose profiles* which incorporate some modules from this specification in combination with other features as needed to meet particular industry requirements.

When applied to conformance, the term "SVG Tiny 1.2" refers to the Tiny profile of SVG 1.2 defined by this specification. If an implementation does not implement the Tiny profile completely, the UA's conformance claims must state either the profile to which it conforms and/or the specific set of features it implements.

1.3 Defining an SVG Tiny 1.2 document

SVG Tiny 1.2 is a backwards compatible upgrade to [SVG Tiny 1.1](#). A few key differences from SVG Tiny 1.1 should be

noted:

- The value of the version attribute on the [rootmost svg element](#) should be "1.2". See discussion of [Version Control](#).
- There is no DTD for SVG 1.2, and therefore no need to specify the DOCTYPE for an SVG 1.2 document (unless it is desired to use the [internal DTD subset](#), for purposes of entity definitions for example). Instead, identification is by the SVG namespace, plus the version and baseProfile attributes. In SVG Tiny 1.2, validation is provided by the [SVG Tiny 1.2 RelaxNG schema](#).

The namespace for SVG Tiny 1.2 is the same as that of SVG 1.0 and 1.1, <http://www.w3.org/2000/svg>

Here is an example of an SVG 1.2 file:

Example: [01_01.svg](#)

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg"
    version="1.2" baseProfile="tiny"
    viewBox="0 0 30 30">
  <desc>Example SVG file</desc>
  <rect x="10" y="10" width="10" height="10" fill="red"/>
</svg>
```

Here is an example of defining an entity in the internal DTD subset. Note that in XML, there is no requirement to fetch the external DTD subset and so relying on an external subset reduces interoperability. Also note that the SVG Working Group does not provide a normative DTD for SVG Tiny 1.2 but instead provides a normative RelaxNG schema.

Example: [entity.svg](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [
  <!ENTITY Smile "
    <rect x='.5' y='.5' width='29' height='39' fill='black' stroke='red'/>
    <g transform='translate(0, 5)'>
      <circle cx='15' cy='15' r='10' fill='yellow'/>
      <circle cx='12' cy='12' r='1.5' fill='black'/>
      <circle cx='17' cy='12' r='1.5' fill='black'/>
      <path d='M 10 19 L 15 23 20 19' stroke='black' stroke-width='2'/>
    </g>
  ">
]>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <title>Smiley face</title>
  <desc>This example shows the use of an entity defined in the
    internal DTD subset. Note that there is no external DTD subset
    for SVG 1.2, and thus no formal public identifier.</desc>
  &Smile;
</svg>
```

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

The MIME type for SVG is "image/svg+xml" (see [Media Type registration for image/svg+xml](#)).

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

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

1.5 Compatibility with Other Standards Efforts

SVG Tiny 1.2 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 Tiny 1.2 is an application of XML and is compatible with the "Extensible Markup Language (XML) 1.1" Recommendation [[XML11](#)]
- SVG Tiny 1.2 is compatible with the "Namespaces in XML 1.1" Recommendation [[XML-NS](#)]
- SVG Tiny 1.2 utilizes "XML Linking Language (XLink)" [[XLINK](#)] for IRI referencing and requires support for base IRI specifications defined in "XML Base" [[XML-BASE](#)].
- SVG Tiny 1.2 uses the xml:id attribute [[XMLID](#)].
- SVG Tiny 1.2 content can be generated by XSLT (see "XSL Transformations (XSLT) Version 1.0" [[XSLT](#)]). (See [Styling with XSL](#))
- SVG Tiny 1.2 supports formatting properties drawn from CSS and XSL. See [SVG's styling properties](#).
- SVG Tiny 1.2 includes a compatible subset of the Document Object Model (DOM) and supports many of the facilities described in "Document Object Model (DOM) level 3" [[DOM3](#)], including namespace support and event handling.
- SVG Tiny 1.2 incorporates some features from "Synchronized Multimedia Integration Language (SMIL) 2.0 Specification" [[SMIL2.0](#)], including the '[switch](#)' element, the [systemLanguage](#) attribute, animation features (see [Animation](#)) and the ability to reference audio and video media (see [Multimedia](#)). SVG's animation features incorporate and extend the general-purpose XML animation capabilities described in SMIL 2.0. In addition, SVG Tiny 1.2 has been designed to allow SMIL 2.0 to use animated or static SVG content as media components.
- SVG is compatible with W3C work on internationalization. References (W3C and otherwise) include: Unicode [[UNICODE](#)], Resource Identifiers [[CHARMOD-RI](#)] and the Character Model [[CHARMOD](#)]. Also, see [Internationalization Support](#).
- SVG is compatible with W3C work on Web Accessibility [[WAI](#)]. Also, see [Accessibility Support](#).

In environments which support the [Document Object Model \(DOM\)](#) [[DOM3](#)] for other XML grammars (e.g., [XHTML](#) [[XHTML](#)]) and which also support SVG and the SVG DOM, a single scripting approach can be used simultaneously for both XML documents and SVG graphics, in which case interactive and dynamic effects will be possible on multiple XML namespaces using the same set of scripts.

1.6 Definitions

When used in this specification, terms have the meanings assigned in this section.

after-edge

Defined in the [XSL Area Model](#).

basic shape

Standard shapes which are predefined in SVG as a convenience for common graphical operations. Specifically any instance of the following elements: '[rect](#)', '[circle](#)', '[ellipse](#)', '[line](#)', '[polyline](#)', '[polygon](#)'.

before-edge

Defined in the [XSL Area Model](#).

canvas

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

container element

An element which can have graphics elements and other container elements as child elements. Specifically: '[svg](#)', '[g](#)', '[defs](#)', '[a](#)' and '[switch](#)'.

current SVG document fragment

The current SVG document fragment of an element is the XML document sub-tree such that:

1. The sub-tree is a valid SVG document fragment.
2. The sub-tree contains the element in question.
3. All ancestors of the element in question in the sub-tree are elements in the SVG language and namespace.

A given element may have no current SVG document fragment.

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 [Coordinate system transformations](#).

Document Begin

Defined in [SMIL](#).

Document End

Defined in [SMIL](#).

fill

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

font

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

glyph

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

graphics element

One of the element types that can cause graphics to be drawn onto the target canvas. Specifically: ['path'](#), ['text'](#), ['textArea'](#), ['rect'](#), ['circle'](#), ['ellipse'](#), ['line'](#), ['polyline'](#), ['polygon'](#), ['image'](#), ['use'](#), ['animation'](#) and ['video'](#).

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'](#), ['image'](#), ['animation'](#) and ['video'](#).

in error

A value is 'in error' if it is specifically stated as being 'in error' or 'an error' in the prose of this specification. See [Error Processing](#) for more detail on handling errors.

IRI Reference

An International Resource Identifier (IRI). Defined in [RFC 3987](#). Serves as a reference to a resource or (with a fragment identifier) to a secondary resource. See [References](#).

Local IRI Reference

An International Resource Identifier (IRI) that references a fragment within the same resource. See [References](#).

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 two types of built-in paint: [color](#) and [gradients](#).

presentation attribute

An XML attribute on an SVG element which specifies a value for a given property for that element. See [Styling](#).

property

A parameter that helps specify how a document should be rendered. A complete list of SVG's properties can be found in [Property Index](#). Properties are assigned to elements in the SVG language by [presentation attributes](#). See [Styling](#).

Rendering Tree

The set of elements being rendered when the painters model is applied to an SVG document fragment. The following elements in the fragment:

- o a ['defs'](#) element
- o elements whose ['display'](#) property is set to 'none
- o elements with one or more [test attributes](#) that evaluate to false
- o direct children of a ['switch'](#) element, other than the child that evaluates to true

and their children, are part of the SVG document fragment, but *not* part of the Rendering tree (and thus do not get rendered). Elements with zero opacity, or no fill and no stroke, or with the visibility property set to hidden, are still in the rendering tree. The copies of elements referenced by a ['use'](#) element, on the other hand, are *not* in the SVG document fragment but are in the rendering tree.

rootmost svg element

The furthest ['svg'](#) ancestor element that does not exit an [SVG context](#).

Note that this definition has been carefully chosen to be applicable not only to SVG Tiny 1.2 (where the rootmost svg element is the only svg element, except when there is an svg element inside a foreignObject) but also for SVG Full 1.2 and SVG that uses sXBL. See also [SVG document fragment](#).

shape

A graphics element that comprises a defined combination of straight lines and curves. Specifically any instance of the element types: ['path'](#), ['rect'](#), ['circle'](#), ['ellipse'](#), ['line'](#), ['polyline'](#), ['polygon'](#).

stroke

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

SVG canvas

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

SVG context

A document fragment where all elements within the fragment must be subject to processing by an SVG user agent according to the rules in this specification.

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

SVG document fragment

The XML document sub-tree whose rootmost element is an ['svg'](#) element. (The [rootmost svg element](#).) An SVG document fragment can consist of a stand-alone SVG document, or a fragment of a parent XML

document enclosed by an **'svg'** element. In SVG Tiny 1.2 each SVG document fragment must not contain nested **'svg'** elements.

SVG element

An element within the SVG namespace defined by the SVG language specification.

SVG viewport

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

Syncbase

Defined in [SMIL](#).

text content element

One of SVG's elements that can define a text string that is to be rendered onto the canvas. SVG Tiny 1.2's text content elements are the following: **'text'**, **'tspan'** and **'textArea'**

Timed Elements

Elements that support the [SVG Timing Attributes](#). They are: [animate](#), [animateColor](#), [animateMotion](#), [animateTransform](#), [set](#), [video](#), [audio](#) and [animation](#).

transformation

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

transformation matrix

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

unsupported value

An unsupported value is a value that does not conform to this specification, but is not specifically listed as 'in error'. See the [Implementation Notes](#) for more detail on processing unsupported values.

user agent

The general definition of a user agent is an application that retrieves and renders Web content, including text, graphics, sounds, video, images, and other content types. A user agent may require additional user agents that handle some types of content. For instance, a browser may run a separate program or plug-in to render sound or video. User agents include graphical desktop browsers, multimedia players, text browsers, voice browsers; used alone or in conjunction with assistive technologies such as screen readers, screen magnifiers, speech synthesizers, onscreen keyboards, and voice input software [\[UAAG10\]](#).

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

user coordinate system

In general, a coordinate system defines locations and distances on the current [canvas](#). The current **user coordinate system** is the coordinate system that is currently active and which is used to define how coordinates and lengths are located and computed, respectively, on the current [canvas](#). See [initial user coordinate system](#) and [Coordinate system transformations](#).

user space

A synonym for [user coordinate system](#).

user units

A coordinate value or length expressed in user units represents a coordinate value or length in the current

[user coordinate system](#). Thus, 10 user units represents a length of 10 units in the current user coordinate system.

viewport

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

viewport coordinate system

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

viewport space

A synonym for [viewport coordinate system](#).

viewport units

A coordinate value or length expressed in viewport units represents a coordinate value or length in the [viewport coordinate system](#). Thus, 10 viewport units represents a length of 10 units in the viewport coordinate system.

Note: When this specification uses the term *'svg' element*, *'path' element*, or similar reference to an SVG element defined within this specification, it is referring to the element whose namespace URI is `http://www.w3.org/2000/svg` and whose localname is the string in quotes (e.g., "svg" or "path").

2 Concepts

Contents

- 2.1 [Explaining the name: SVG](#)
 - 2.1.1 [Scalable](#)
 - 2.1.2 [Vector](#)
 - 2.1.3 [Graphics](#)
 - 2.1.4 [XML](#)
 - 2.1.5 [Namespace](#)
 - 2.1.6 [Scriptable](#)
- 2.2 [Important SVG concepts](#)
 - 2.2.1 [Graphical Objects](#)
 - 2.2.2 [Reuse](#)
 - 2.2.3 [Fonts](#)
 - 2.2.4 [Animation](#)
- 2.3 [Options for using SVG in Web pages](#)

This chapter is informative

2.1 Explaining the name: SVG

SVG stands for [Scalable](#) [V](#)ector [G](#)raphics, an [XML](#) grammar for 2D vector graphics, usable as an [XML namespace](#).

2.1.1 Scalable

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

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

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

2.1.2 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 to produce a complete illustration; SVG is no exception.

Since all modern displays are raster-oriented, the difference between raster-only and vector graphics comes down to where they are rasterized; client side in the case of vector graphics, as opposed to already rasterized on the server. SVG provides hints to control the rasterization process, for example to allow anti-aliased artwork without the ugly aliasing typical of low quality vector implementations.

2.1.3 Graphics

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

2.1.4 XML

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

2.1.5 Namespace

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

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

2.1.6 Scriptable

The combination of scripting and the HTML DOM is often termed "Dynamic HTML" and is widely used for animation, interactivity and presentational effects. Similarly SVG allows the script-based manipulation of the document tree using a subset of the XML DOM and the SVG [uDOM](#).

2.2 Important SVG concepts

2.2.1 Graphical Objects

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

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

2.2.2 Reuse

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

Graphics may be re-used at different sizes and orientations.

2.2.3 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.

2.2.4 Animation

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

2.3 Options for using SVG in Web pages

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

- **A stand-alone SVG Web page**

In this case, an SVG document (i.e., a Web resource whose MIME type is "image/svg+xml") is loaded directly into a user agent such as a Web browser. The SVG document is the Web page that is presented to the user.

- **Embedding by reference**

In this case, a parent Web page references a separately stored SVG document and specifies that the given SVG document should be embedded as a component of the parent Web page. For HTML or XHTML, here are three options:

- The HTML/XHTML **'img'** element is the most common method for using graphics in HTML pages. For faster display, the width and height of the image can be given as attributes. One attribute that is required is **alt**, used to give an alternate textual string for people browsing with images off, or who cannot see the images. The string cannot contain any markup. A **longdesc** attribute lets you point to a longer description - often in HTML - which can have markup and richer formatting.
- The HTML/XHTML **'object'** element can contain other elements nested within it, unlike **'img'**, which is empty. This means that several different formats can be offered, using nested **'object'** elements, with a final textual alternative (including markup, links, etc). The rootmost element which can be displayed will be used.
- The HTML/XHTML **'applet'** element which can invoke a Java applet to view SVG content within the given Web page. These applets can do many things, but a common task is to use them to display images, particularly ones in unusual formats or which need to be presented under the control of a program for some other reason.

- **Embedding inline**

In this case, SVG content is embedded inline directly within the parent Web page. An example is an XHTML Web page with an SVG document fragment textually included within the XHTML.

- **External link, using the HTML **'a'** element**

This allows any stand-alone SVG viewer to be used, which can (but need not) be a different program to that used to display HTML. This option typically is used for unusual image formats.

- **Referenced from a CSS or XSL property**

When a user agent supports [Cascading Style Sheets, Level 2 \[CSS2\]](#) styled XML content, or [Extensible Stylesheet Language \[XSL\]](#) Formatting Objects, and the user agent is a [Conforming SVG Viewer](#), then that user agent must support the ability to reference SVG resources wherever CSS or XSL properties allow for the referencing of raster images, including the ability to tile SVG graphics wherever necessary and the ability to composite the SVG into the background if it has transparent portions. Examples include the **'background-image'** and **'list-style-image'** properties that are included in both CSS and XSL.

3 Rendering Model

Contents

- 3.1 [Introduction](#)
- 3.2 [The painters model](#)
- 3.3 [Rendering Order](#)
- 3.4 [Types of graphics elements](#)
 - 3.4.1 [Rendering shapes and text](#)
 - 3.4.2 [Rendering raster images](#)
 - 3.4.3 [Rendering video](#)
- 3.5 [Object Opacity](#)
- 3.6 [Parent Compositing](#)

3.1 Introduction

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

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

3.2 The painters model

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

3.3 Rendering Order

Elements in an SVG document fragment have an implicit drawing order. Element are rendered using a pre-order, depth-first walk of the SVG document fragment. Subsequent elements are painted on top of previously painted elements.

3.4 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
- Replaced content:
 - 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](#).)
 - Video, which represents a timed sequence of raster images.
 - Animation, which represents a timed vector animation
 - Foreign objects, which represent rendering of non-SVG content

3.4.1 Rendering 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.

The fill is painted first, then the stroke.

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

- [Solid color](#)
- [Gradients](#) (linear and radial)

3.4.2 Rendering raster images

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

3.4.3 Rendering video

As a video stream is a timed sequence of raster images, rendering video has some similarity to rendering raster images. However, given the processing required to decode a video stream, not all implementations may be able to transform the video output into SVG's userspace. Instead they may be limited to rendering in device space. More information can be found in the definition for [video](#).

3.5 Object Opacity

Each fill or stroke painting operation must behave as though the operation were first performed to an intermediate canvas which is initialized to transparent black onto which either the solid color or gradient paint is applied. Then, the alpha values on the intermediate canvas are multiplied by the fill-opacity or stroke-opacity values. The resulting canvas is composited into the background using simple alpha blending.

3.6 Parent Compositing

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

4 Basic Data Types and Color Keywords

4.1 Basic Data Types

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

- **<Boolean>**: A <Boolean> is specified as either 'true' or 'false', and is therefore a subset of [XML Schema Part 2](#) `xs:boolean` types in that it does not accept '0' or '1' as valid values.
- **<integer>**: An <integer> is specified as an optional sign character ('+' or '-') followed by one or more digits "0" to "9". If the sign character is not present, the number is non-negative.

Conforming SVG Tiny 1.2 content must use <integer> values with a range of -32,768 to 32,767.

- **<number>** (real number value): A real number value is specified in either [decimal notation](#) or in **scientific notation**. A **<decimal-number>** consists of either an [integer](#), or an optional sign character followed by zero or more digits followed by a dot (.) followed by one or more digits. A **<scientific-number>** consists of a [decimal-number](#) immediately followed by the letter "e" or "E" immediately followed by an [integer](#).

Conforming SVG Tiny 1.2 content must use <number> with the capacity for a fixed point number in the range '-32,767.9999 to +32,767.9999'. It is recommended that higher precision floating point storage and computation be performed on operations such as coordinate system transformations to provide the best possible precision and to prevent round-off errors.

- **<length>**: A length is a distance measurement. The format of a <length> is a [number](#) optionally followed immediately by a unit identifier. If the <length> is expressed as a value without a unit identifier (e.g., **48**), then the <length> represents a distance in the current user coordinate system.

SVG Tiny 1.2 only supports CSS units on the 'width' and 'height' attributes on the 'svg' element. These can specify values in any of the following CSS units: in, cm, mm, pt, pc, px and %. If one of the unit identifiers is provided (e.g., **12mm**), then the <length> is processed according to the description in [Units](#).

Percentage values (e.g., **10%**) on the width and height attributes of the svg element represent a percent of the viewport (refer to the section that discusses [Units](#) in general).

- **<coordinate>**: A <coordinate> represents a [length](#) in the user coordinate system that is the given distance from the origin of the user coordinate system along the relevant axis (the x-axis for X coordinates, the y-axis for Y coordinates).
- **<list of xxx>** (where xxx represents a value of some type): A list consists of a separated sequence of values. Unless explicitly described differently, lists can be either comma-separated, with optional white space before or after the comma, or white space-separated.

White space in lists is defined as one or more of the following consecutive characters: "space" (U+0020), "tab" (U+0009), "line feed" (U+000A) and "carriage return" (U+000D)

Here is a description of the grammar for a <list of xxx>:

```
ListOfXXX:
  XXX
  | XXX comma-wsp ListOfXXX
comma-wsp:
  (wsp+ comma? wsp*) | (comma wsp*)
comma:
  ", "
wsp:
  (#x20 | #x9 | #xD | #xA)
```

where XXX represents a particular type of value.

- **<color>**: The basic type <color> defines a color within the sRGB color space [[sRGB](#)]. <color> applies to SVG's use of the **'color'** property and is a component of the definitions of properties **'fill'**, **'stroke'**, **'viewport-fill'**, **'stop-color'** and **'solid-color'**.

SVG supports all of the syntax alternatives for <color> defined in [Syntax for color values](#). All RGB colors are specified in the sRGB color space (see [[sRGB](#)]). Using sRGB provides an unambiguous and objectively measurable definition of the color, which can be related to international standards [[COLORIMETRY](#)].
- **<paint>**: The values for properties **'fill'** and **'stroke'** are specifications of the type of paint to use when filling or stroking a given graphics element. The available options and syntax for **<paint>** are described in [Specifying paint](#).
- **<transform-list>**: The detailed description of the possible values for a <transform-list> are detailed in [Modifying the User Coordinate System: the transform attribute](#).
- **<iri>** The <iri> type holds a reference to an International Resource Identifier (see [IRI](#)): An [IRI](#) is the address of a resource on the Web. For the specification of [IRI](#) references in SVG, see [IRI references](#).
- **<Clock-value>**: The detailed description of the possible values for a <Clock-value> are detailed in [Animation: Clock values](#).
- **<xslt-qname>**: As xslt-qname is a QName occurring in attribute content as defined in the [XSLT 1.0 Recommendation](#) whereby if it has a prefix, then the prefix is expanded into an IRI reference using the namespace declarations in effect where the name occurs, and the default namespace is *not* used for unprefixed names.
- **<languageID>**: A languageID is the set of valid language identifiers as defined by the `xs:language` data type in [XML Schema Part 2](#), plus the empty string.
- **<Focus>**: The detailed description of the possible values for a <Focus> are detailed in [Specifying navigation](#).
- **<ContentType>**: a media type, as per [[RFC2045](#)].
- **<FontFamilyValue>**: a font family as detailed in [Font selection properties](#).
- **<FontSizeValue>**: a font size as detailed in [Font selection properties](#).
- **<FormatList>**: a list of format specifiers as detailed in [The requiredFormats attribute](#).
- **<ID>**: an XML ID.
- **<PathData>**: path data as detailed in [the Paths chapter](#).
- **<Points>**: a list of points as detailed in [the Basic Shapes chapter](#).

5 Document Structure

Contents

- 5.1 [Defining an SVG document fragment: the 'svg' element](#)
 - 5.1.1 [Overview](#)
 - 5.1.2 [The 'svg' element](#)
- 5.2 [Grouping: the 'g' element](#)
 - 5.2.1 [Overview](#)
 - 5.2.2 [The 'g' element](#)
- 5.3 [The 'defs' element](#)
- 5.4 [The 'discard' element](#)
- 5.5 [The 'desc' and 'title' elements](#)
- 5.6 [The 'use' element](#)
- 5.7 [The 'image' element](#)
- 5.8 [Conditional processing](#)
 - 5.8.1 [Conditional processing overview](#)
 - 5.8.2 [The 'switch' element](#)
 - 5.8.3 [The 'requiredFeatures' attribute](#)
 - 5.8.4 [The 'requiredExtensions' attribute](#)
 - 5.8.5 [The 'systemLanguage' attribute](#)
 - 5.8.6 [The 'requiredFormats' attribute](#)
 - 5.8.7 [The 'requiredFonts' attribute](#)
- 5.9 [External Resources](#)
 - 5.9.1 [The 'externalResourcesRequired' attribute](#)
 - 5.9.2 [Progressive Rendering](#)
 - 5.9.3 [The 'prefetch' element](#)
- 5.10 [Common attributes](#)
 - 5.10.1 [Attributes common to all elements: 'class', 'id', 'xml:id' and 'xml:base'](#)
 - 5.10.2 [The 'xml:lang' and 'xml:space' attributes](#)
- 5.11 [Core Attribute Module](#)
- 5.12 [Structure Module](#)
- 5.13 [Conditional Processing Module](#)
- 5.14 [Conditional Processing Attribute Module](#)
- 5.15 [Image Module](#)
- 5.16 [Prefetch Module](#)
- 5.17 [Discard Module](#)
- 5.18 [ExternalResourcesRequired Attribute Module](#)

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

5.1.1 Overview

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

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

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

The following example shows simple SVG content embedded inline as a fragment within a parent XML document. Note the use of XML namespaces to indicate that the **'svg'** and **'ellipse'** elements belong to the SVG namespace:

Example: 05_01.xml

```
<?xml version="1.0"?>
<parent xmlns="http://example.org"
  xmlns:svg="http://www.w3.org/2000/svg">
  <!-- parent contents here -->
  <svg:svg width="4cm" height="8cm" version="1.2" baseProfile="tiny" viewBox="0 0 100 100">
    <svg:ellipse cx="50" cy="50" rx="40" ry="20" />
  </svg:svg>
  <!-- ... -->
</parent>
```

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

Example: 05_02.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg width="5cm" height="4cm" xmlns="http://www.w3.org/2000/svg">
```

```

    version="1.2" baseProfile="tiny" viewBox="0 0 100 100">
<desc>Four separate rectangles
</desc>
<rect x="20" y="20" width="20" height="20"/>
<rect x="50" y="20" width="30" height="15"/>
<rect x="20" y="50" width="20" height="20"/>
<rect x="50" y="50" width="20" height="40"/>
<!-- Show outline of canvas using 'rect' element -->
<rect x="1" y="1" width="98" height="98"
      fill="none" stroke="blue" stroke-width="2" />
</svg>

```

An [SVG document fragment](#) can only contain one single '[svg](#)' element, this means that '[svg](#)' elements cannot appear in the middle of SVG content.

In all cases, for compliance with the "Namespaces in XML 1.1" Recommendation [[XML-NS](#)], an SVG namespace declaration must be in scope for the '[svg](#)' element, so that all SVG elements are identified as belonging to the SVG namespace.

For example, an '[xmlns](#)' attribute without a prefix could be specified on an '[svg](#)' element, which means that SVG is the default namespace for all elements within the scope of the element with the '[xmlns](#)' attribute:

Example: 05_03.svg

```

<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Demonstrates use of a default namespace prefix for elements.</desc>
  <rect width="7" height="3"/>
</svg>

```

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

Example: 05_04.svg

```

<?xml version="1.0"?>
<s:svg xmlns:s="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <s:desc>Demonstrates use of a namespace prefix for elements.
  Notice that attributes are not namespaced</s:desc>
  <s:rect width="7" height="3"/>
</s:svg>

```

Namespace declarations can also be specified on ancestor elements (illustrated in the [above example](#)). For more information, refer to the "Namespaces in XML 1.1" Recommendation [[XML-NS](#)].

5.1.2 The '[svg](#)' element

Schema: [svg](#)

```

<define name='svg'>
  <element name='svg'>
    <ref name='svg.AT' />
    <zeroOrMore><ref name='svg.G.group' /></zeroOrMore>
  </element>
</define>

<define name='svg.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.AnimateSyncDefault.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.WH.attr' />
  <ref name='svg.PAR.attr' />
  <optional>
    <attribute name='viewBox' svg:animatable='true' svg:inheritable='false'>
      <ref name='ViewBoxSpec.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='zoomAndPan' a:defaultValue='magnify' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>disable</value>
        <value>magnify</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='version' a:defaultValue='1.1' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value type='string'>1.0</value>
        <value type='string'>1.1</value>
        <value type='string'>1.2</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='baseProfile' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value type='string'>none</value>
        <value type='string'>tiny</value>
      </choice>
    </attribute>
  </optional>

```

```

    <value type='string'>basic</value>
    <value type='string'>full</value>
  </choice>
</attribute>
</optional>
<optional>
  <attribute name='contentType' a:defaultValue='application/ecmascript' svg:animatable='false' svg:inheritable='false'>
    <ref name='ContentType.datatype' />
  </attribute>
</optional>
<optional>
  <attribute name='snapshotTime' svg:animatable='false' svg:inheritable='false'>
    <choice>
      <value type='string'>none</value>
      <ref name='Clock-value.datatype' />
    </choice>
  </attribute>
</optional>
<optional>
  <attribute name='timelineBegin' a:defaultValue='onLoad' svg:animatable='false' svg:inheritable='false'>
    <choice>
      <value type='string'>onLoad</value>
      <value type='string'>onStart</value>
    </choice>
  </attribute>
</optional>
<optional>
  <attribute name='playbackOrder' a:defaultValue='all' svg:animatable='false' svg:inheritable='false'>
    <choice>
      <value type='string'>all</value>
      <value type='string'>forwardOnly</value>
    </choice>
  </attribute>
</optional>
</define>

```

Attribute definitions:

version = "1.0" | "1.1" | "1.2"

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

In SVG 1.0 and SVG 1.1 this attribute had the value '1.0' or '1.1' respectively, and SVG 1.2 adds the '1.2'. See [rules for version processing](#) for further instructions, notably on handling of unsupported values.

This attribute is read-only and cannot be modified via the [uDOM](#) api. Attempt to set the 'version' attribute with the [setAttributeNS](#) must result in an [NO_MODIFICATION_ALLOWED_ERR](#) being raised.

Animatable: no.

baseProfile = "profile-name"

Describes the minimum SVG language profile that the author believes is necessary to correctly render the content. See [rules for baseProfile processing](#) for further instructions.

This specification defines the values 'none' and 'tiny'. The value 'full' corresponds to all features in the SVG language; for SVG 1.1, this corresponds to the language features defined in the [SVG 1.1 specification](#). The value 'basic' was defined in the [SVG Mobile 1.1 specification](#). This specification corresponds to baseProfile="tiny" and version="1.2". A value of 'none' provides no information about the minimum language profile that is necessary to render the content.

If the attribute is not specified, the effect is as if a value of 'none' were specified.

This attribute is read-only and cannot be modified via the [uDOM](#) api. Attempt to set the baseProfile attribute with the [setAttributeNS](#) must result in an [NO_MODIFICATION_ALLOWED_ERR](#) being raised.

Animatable: no.

width = "<length>"

The intrinsic width of the SVG document fragment.

A negative value is an error (see [Error processing](#)). A value of zero disables rendering of the element.

If the attribute is not specified, the effect is as if a value of "100%" were specified.

Animatable: yes.

height = "<length>"

The intrinsic height of the SVG document fragment.

A negative value is an error (see [Error processing](#)). A value of zero disables rendering of the element.

If the attribute is not specified, the effect is as if a value of "100%" were specified.

Animatable: yes.

viewBox = "<list of number>" | "none"

See [attribute definition](#) for description.

Animatable: yes.

preserveAspectRatio = "[defer] <align> [<meet>]"

See [attribute definition](#) for description.

Animatable: yes.

snapshotTime = "<clock-value>" | "none"

Indicates a moment in time which is most relevant for a still-image of the animated SVG content. This time may be used as a hint to the [SVG User Agent](#) for rendering a still-image of an animated SVG Document, such as a preview. A value of 'none' means that no 'snapshotTime' is available. See an [example](#) of using the 'snapshotTime' attribute.

If the attribute is not specified, the effect is as if a value of 'none' was specified.

Animatable: no.

playbackOrder = "forwardOnly" | "all"

Indicates whether it is possible to seek backwards. In earlier versions of SVG there was no need to put restrictions on the direction of seeking but with the newly introduced facilities for long-running documents (e.g. the ['discard'](#) element) there is sometimes a need to restrict this.

If ['playbackOrder'](#) is set to ['forwardOnly'](#), the content will probably contain ['discard'](#) elements or scripts that destroy resources, thus seeking back in the document's timeline may result in missing content. If ['playbackOrder'](#) is ['forwardOnly'](#), the content should not provide a way, through hyperlinking or script, of seeking backwards in the timeline. Similarly the UA should disable any controls it may provide in the user interface for seeking backwards. Content with ['playbackOrder'](#) = ['forwardOnly'](#) that provides a mechanism for seeking backwards in time may result in undefined behavior or a document that is in error.

['forwardOnly'](#)
This file is intended to be played only in the forward direction, sequentially, therefore seeking backwards should not be allowed.

['all'](#)

Indicates that the document is authored appropriately for seeking in both directions.

The default value is ['all'](#).

[Animatable](#): no.

timelineBegin = ["onLoad"](#) | ["onStart"](#)

Controls the initialization of the timeline for the document.

The ['svg'](#) element controls the *Document Timeline*, which is the timeline of the ['svg'](#) element's time container. For progressively loaded animations, the author would typically set this attribute to ['onStart'](#), thus allowing the timeline to begin as the document loads, rather than waiting until the complete document is loaded.

['onLoad'](#)

The document's timeline starts the moment the [load](#) event for the [rootmost svg element](#) is triggered.

['onStart'](#)

The document's timeline starts at the moment the [rootmost svg element](#)'s open tag is fully parsed and processed.

The default value is ['onLoad'](#).

[Animatable](#): no.

contentScriptType = ["content-type"](#)

Identifies the default scripting language for the given document. This attribute sets the default scripting language for all the instances of script in the document fragment. This language must be used for all scripts that do not specify their own scripting language. The value [content-type](#) specifies a media type, per [Multipurpose Internet Mail Extensions \(MIME\) PartTwo \[RFC2046\]](#). The default value is ["application/ecmascript"](#).

[Animatable](#): no.

focusable = ["true"](#) | ["false"](#) | ["auto"](#)

See [attribute definition](#) for description.

[Animatable](#): Yes

Navigation Attributes

See [definition](#).

Content produced by illustration programs originally targeted at print often has a fixed width and height, which will prevent it scaling for different display resolutions. The first example below has a fixed width and height in pixels, and no [viewBox](#).

Example: [width-height.svg](#)

```
<?xml version="1.0"?>
<svg width="300px" height="600px" xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny">
  <desc>...</desc>
</svg>
```

Normally, SVG content is designed to be scalable. To be scalable, SVG content must include a [viewBox](#) attribute on the ['svg'](#) element. This describes the region of world coordinate space (the initial user coordinate system) used by the graphic. This attribute thus provides a convenient way to design SVG documents to scale-to-fit into an arbitrary viewport.

The second example is scalable, using a [viewBox](#) rather than a fixed width and height.

Example: [viewBox.svg](#)

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
  baseProfile="tiny" viewBox="0 0 300 600">
  <desc>...</desc>
</svg>
```

Below is an example of ['snapshotTime'](#). An [SVG User Agent](#) is displaying a number of SVG files in a directory by rendering a thumbnail image. It uses the ['snapshotTime'](#) as the time to render when generating the image, thus giving a more representative static view of the animation. The appearance of the thumbnail for an [SVG User Agent](#) that honors the ['snapshotTime'](#) and for an [SVG User Agent](#) that does not is shown below the example (UA with snapshot support at the left, without snapshot support at the right).

Example: [05_22.svg](#)

```
<svg version="1.2" snapshotTime="3" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
  viewBox="0 0 400 300">
  <title>Snapshot time example</title>
  <desc>This example shows the use of snapshotTime on an animation of
  color
  </desc>
  <rect x="60" y="85" width="256" height="65" fill="none" stroke="rgb(60,126,220)" stroke-width="4"/>
  <text x="65" y="140" fill="rgb(60,126,220)" font-size="60">Hello SVG
  <animateColor attributeName="fill" begin="0" dur="3" from="white" to="rgb(60,126,220)"/>
  </text>
</svg>
```

Hello SVG



5.2 Grouping: the 'g' element

5.2.1 Overview

The 'g' element is a [container element](#) for grouping together related [graphics elements](#).

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

A group of elements, as well as individual objects, can be given a name using the ['xml:id'](#) attribute. Named groups are needed for several purposes such as animation and re-usable objects.

An example:

Example: 05_05.svg

```
<?xml version="1.0"?>
<svg width="5cm" height="5cm" xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny" viewBox="0 0 5 5">
  <desc>Two groups, each of two rectangles
  </desc>
  <g xml:id="group1" fill="red" >
    <desc>First group of two red rectangles</desc>
    <rect x="1" y="1" width="1" height="1" />
    <rect x="3" y="1" width="1" height="1" />
  </g>
  <g xml:id="group2" fill="blue" >
    <desc>Second group of two blue rectangles</desc>
    <rect x="1" y="3" width="1" height="1" />
    <rect x="3" y="3" width="1" height="1" />
  </g>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01" y=".01" width="4.98" height="4.98"
    fill="none" stroke="blue" stroke-width=".02" />
</svg>
```

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

Example: 05_06.svg

```
<?xml version="1.0"?>
<svg width="5cm" height="5cm" xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny">
  <desc>Groups can nest
  </desc>
  <g>
    <g>
      <g>
        </g>
      </g>
    </g>
  </g>
</svg>
```

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

5.2.2 The 'g' element

Schema: g

```
<define name='g'>
  <element name='g'>
    <ref name='g.AT' />
    <zeroOrMore><ref name='svg.G.group' /></zeroOrMore>
  </element>
</define>

<define name='g.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.Focus.attr' />
</define>
```



```
<ref name='svg.Transform.attr' />
</define>
```

Attribute definitions:

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

5.3 The 'defs' element

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

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

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

Schema: defs

```
<define name='defs'>
  <element name='defs'>
    <ref name='defs.AT' />
    <zeroOrMore><ref name='svg.G.group' /></zeroOrMore>
  </element>
</define>

<define name='defs.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
</define>
```

Creators of SVG content are encouraged to place all elements which are targets of local [IRI references](#) (except of course for animation targets) within a 'defs' element which is a direct child of one of the ancestors of the referencing element. For example:

Example: 05_10.svg

```
<?xml version="1.0"?>
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny" viewBox="0 0 8 3">
  <desc>Local URI references within ancestor's 'defs' element.</desc>
  <defs>
    <linearGradient xml:id="Gradient01">
      <stop offset="0.2" stop-color="#39F" />
      <stop offset="0.9" stop-color="#F3F" />
    </linearGradient>
  </defs>
  <rect x="1" y="1" width="6" height="1"
    fill="url(#Gradient01)" />
  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01" y=".01" width="7.98" height="2.98"
    fill="none" stroke="blue" stroke-width=".02" />
</svg>
```

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

5.4 The 'discard' element

The 'discard' element allows authors to specify the time at which particular elements are to be discarded, therefore reducing the resources required by an SVG UA. This is particularly useful for the SVG viewers to handle long-running documents. This element will not be processed by static SVG viewers.

The 'discard' element may occur wherever the [animate](#) element may.

Schema: discard

```
<define name='discard'>
  <element name='discard'>
    <ref name='discard.AT' />
    <ref name='discard.CM' />
  </element>
</define>

<define name='discard.AT' combine='interleave'>
```

```

<ref name='svg.Core.attr' />
<ref name='svg.XLinkRequired.attr' />
<ref name='svg.AnimateBegin.attr' />
</define>

<define name='discard.CM'>
<zeroOrMore>
<ref name='svg.Desc.group' />
<ref name='svg.Handler.group' />
</zeroOrMore>
</define>

```

Attribute definitions:

xlink:href = "[<iri>](#)"

An [IRI reference](#) to the element to discard. See the definition of [target element](#). If the attribute is not specified, the effect is as if an empty value ("") was specified.

Animatable: no.

begin = "[<offset-value>](#)" | "[<syncbase-value>](#)" | "[<event-value>](#)" | "[<accessKey-value>](#)" | "indefinite"

The **'discard'** element has an implicit [simple duration](#) of 'indefinite'. As soon as the element's [active duration](#) starts, the [SVG User Agent](#) discards the element identified by the **xlink:href** attribute. The removal operation acts as if the method [removeChild](#) were called on the parent of the target element with the target element as parameter. The [SVG User Agent](#) must remove the target node as well as all of its attributes and descendants.

If the attribute is not specified, the effect is as if a value of "0s" were specified. This indicates that the [target element](#) should be discarded immediately.

Animatable: no.

If the [target element](#) does not exist when the **'discard'** element becomes active, then the **'discard'** element must be ignored.

[SVG User Agents](#) must discard an element when it is the target of a **'discard'** element and the time specified by the value of the **begin** attribute of this **'discard'** element is reached. When an element is discarded, all its descendants must be discarded as well.

After removal of the [target element](#), the **'discard'** element is no longer useful. It must also be discarded following the [target element](#) removal. If the target element did not exist the **'discard'** element must still be removed following activation.

[Seeking backwards](#) in the timeline must not re-insert the discarded elements. Discarded elements are intended to be completely removed from memory. So, authors are encouraged to set the **playbackOrder** attribute to **forwardOnly** when using the **'discard'** element.

The **'discard'** element itself can be discarded prior to its activation, in which case it will never trigger the removal of its own target element. UA's must allow the **'discard'** element to be the target of another **'discard'** element.

The following example shows simple usage of the **'discard'** element. The list below describes relevant happenings in the document timeline of this example:

At time = 0:

When the document timeline starts, the blue ellipse starts to move down the page.

At time = 1 second:

The red rectangle starts moving up the page.

At time = 2 seconds:

The animateTransform on the ellipse ends. The ellipse and its children are also discarded, as it is the [target element](#) of a discard with begin='2'. The green polygon starts to move across the page.

At time = 3 seconds:

The animation on the red rectangle ends. The rectangle and its children are discarded as it is the target of a discard element with begin='3'.

At time = 4 seconds:

The animation on the green triangle ends. The green polygon and its children are discarded as it is the target of a discard element with begin='4'.

Example: [discard01.svg](#)

```

<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
width="352" height="240" playbackOrder="forwardOnly">

<ellipse cx="98.5" cy="17.5" rx="20.5" ry="17.5" fill="blue" stroke="black"
transform="translate(9 252) translate(3 -296)">
<animateTransform attributeName="transform" begin="0s" dur="2s" fill="remove"
calcMode="linear" type="translate" additive="sum" from="0 0" to="-18 305"/>
<discard begin="2s"/>
</ellipse>

<rect x="182" y="-39" width="39" height="30" transform="translate(30 301)" fill="red" stroke="black">
<animateTransform attributeName="transform" begin="1s" dur="2s" fill="remove"
calcMode="linear" type="translate" additive="sum" from="0 0" to="-26 -304"/>
<discard begin="3s"/>
</rect>

<polygon points="-66,83.5814 -43,123.419 -89,123.419" fill="green" stroke="black"
transform="matrix(1 0 0 1.1798 0 -18.6096)">
<animateTransform attributeName="transform" begin="2s" dur="2s"
fill="remove" calcMode="linear" type="translate" additive="sum" from="0 0"
to="460 63.5699"/>
<discard begin="4s"/>
</polygon>
</svg>

```

5.5 The 'desc' and 'title' elements

Each **container element** or **graphics element** in an SVG document may supply a **'desc'** and may also supply a **'title'** description. The **'desc'** element contains a detailed description for the **container** or **graphics element** containing it. This description should be usable as replacement content for cases when the user cannot see the rendering of the **SVG element** for some reason. The **'title'** element contains a short title for the **container** or **graphics element** containing it. This short title provides information supplementary to the rendering of the element, but is not sufficient to replace it. These are typically text, but can be content in other markup languages (see [foreign namespaces](#)). When the **current SVG document fragment** is rendered as SVG on visual media, **'desc'** and **'title'** elements are not rendered as part of the graphics. **SVG User Agents** may, however, for example, display the **'title'** element as a tooltip, as the pointing device moves over particular elements. Alternate presentations are possible, both visual and aural, which display the **'desc'** and **'title'** elements but do not display **'path'** elements or other **graphics elements**. For deep hierarchies, and for following **use** element references, it is sometimes desirable to allow the user to control how deep they drill down into descriptive text.

Schema: desc

```
<define name='desc'>
  <element name='desc'>
    <ref name='DTM.AT' />
    <ref name='DTM.CM' />
  </element>
</define>

<define name='DTM.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
</define>

<define name='DTM.CM'>
  <text />
</define>
```

Schema: title

```
<define name='desc'>
  <element name='desc'>
    <ref name='DTM.AT' />
    <ref name='DTM.CM' />
  </element>
</define>
```

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

Example: 05_11.svg

```
<?xml version="1.0"?>
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny">
  <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>
```

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

Example: 05_12.svg

```
<?xml version="1.0"?>
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny">
  <desc xmlns:mydoc="http://example.org/mydoc">
    <mydoc:title>This is an example SVG file</mydoc:title>
    <mydoc:para>The global description uses markup from the
      <mydoc:emph>mydoc</mydoc:emph> namespace.</mydoc:para>
  </desc>
  <g>
    <!-- the picture goes here -->
  </g>
</svg>
```

Authors should always provide a **'title'** child element to the **'svg'** element within a stand-alone SVG document. The **'title'** child element to an **'svg'** element serves the purposes of identifying the content of the given **SVG document fragment**. Since users often consult documents out of context, authors should provide context-rich titles. Thus, instead of a title such as "Introduction", which doesn't provide much contextual background, authors should supply a title such as "Introduction to Medieval Bee-Keeping" instead. For reasons of accessibility, **SVG User Agents** should always make the content of the **'title'** child element to the **'svg'** element available to users (See the User Agent Accessibility Guidelines 1.0 [\[UAAG\]](#)). The mechanism for doing so depends on the **SVG User Agent** (e.g., as a caption, spoken).

It is strongly recommended that at most one **'desc'** and at most one **'title'** element appear as a child of any particular element, and that these elements appear before any other child elements (except possibly **'metadata'** elements) or character data content. If **SVG User Agents** need to choose among multiple **'desc'** or **'title'** elements for processing (e.g., to decide which string to use for a tooltip), the user agent shall choose the first one the test attributes of which resolve to true.

5.6 The 'use' element

Any **'g'** or **graphics element** is potentially a template object that can be re-used (i.e. "instantiated") in the SVG Document via a **'use'** element, thus creating an instance

tree. The **'use'** element references another element and indicates that the graphical contents of that element is to be included and drawn at that given point in the document.

Unlike **'animation'**, the **'use'** element cannot reference entire files.

Besides what is described about the **'use'** element in this section important restrictions for **'use'** can be found in the [Reference Section](#).

The **'use'** element has optional attributes **'x'** and **'y'** which are used to place the referenced element and its contents into the current coordinate system.

The effect of a **'use'** element is as if the [SVG Element](#) 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 the **'use'** element. The deeply-cloned tree, also referred to as the shadow tree, is then kept in synchronization with the contents of the referenced element, so that any animation or DOM manipulation occurring on the referenced element are also applied to the **'use'** element's deeply-cloned tree.

Relative [IRIs](#) on nodes in shadow trees are resolved relative to any **'xml:base'** on the node itself, then recursively on any **'xml:base'** on their [parentNode](#), and finally any **'xml:base'** on the [ownerDocument](#) if there is no [parentNode](#).

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

If an event listener is registered on a referenced element, then the actual target for the event will be the [SVGElementInstance](#) object within the "instance tree" corresponding to the given referenced element.

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

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

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

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

As listed in the [Reference Section](#) the **'use'** element is not allowed to reference an **'svg'** element

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

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

When a **'use'** references another element which is another **'use'** or whose content contains a **'use'** element, then the deep cloning approach described above is recursive. However, a set of references that directly or indirectly reference a element to create a circular dependency is an error, as described in the [References](#) section.

Schema: use

```
<define name='use'>
  <element name='use'>
    <ref name='use.AT' />
    <ref name='use.CM' />
  </element>
</define>

<define name='use.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.XLinkEmbed.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.XY.attr' />
</define>

<define name='use.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
      <ref name='svg.Handler.group' />
    </choice>
  </zeroOrMore>
</define>
```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of one corner of the rectangular region into which the referenced element is placed.

If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

y = "[<coordinate>](#)"

The y-axis coordinate of one corner of the rectangular region into which the referenced element is placed.

If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

xlink:href = "[<iri>](#)"

An [IRI Reference](#) to an element/fragment within an SVG document. An empty attribute value (**xlink:href=""**) disables rendering of the element. If the attribute is not specified, the effect is as if an empty value ("") was specified.

[Animatable](#): yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

[Animatable](#): Yes

Navigation Attributes

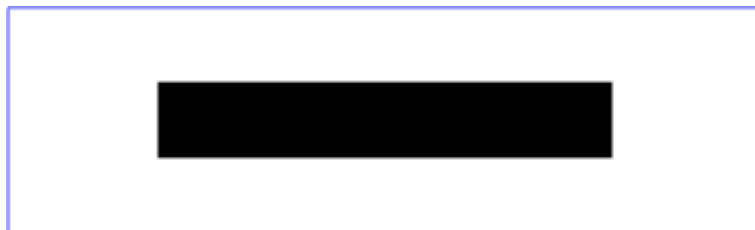
See [definition](#).

Below are two examples of the '[use](#)' element, for another example see [use and animation example](#).

[Example 05_13](#) below has a simple '[use](#)' on a '[rect](#)'.

Example: 05_13.svg

```
<?xml version="1.0"?>
<svg width="10cm" height="3cm" viewBox="0 0 100 30" version="1.2"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  baseProfile="tiny">
  <desc>Example Use01 - Simple case of 'use' on a 'rect'</desc>
  <defs>
    <rect xml:id="MyRect" width="60" height="10"/>
  </defs>
  <rect x=".1" y=".1" width="99.8" height="29.8"
    fill="none" stroke="blue" stroke-width=".2" />
  <use x="20" y="10" xlink:href="#MyRect" />
</svg>
```



The visual effect would be equivalent to the following document:

Example: 05_14.svg

```
<?xml version="1.0"?>
<svg width="100%" height="100%" viewBox="0 0 100 30" version="1.2"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  baseProfile="tiny">
  <desc>Example Use01-GeneratedContent - Simple case of 'use' on a 'rect'</desc>
  <!-- 'defs' section left out -->
  <rect x=".1" y=".1" width="99.8" height="29.8"
    fill="none" stroke="blue" stroke-width=".2" />
  <!-- Start of generated content. Replaces 'use' -->
  <g transform="translate(20,10)">
    <rect width="60" height="10"/>
  </g>
  <!-- End of generated content -->
</svg>
```

[Example 05_17](#) illustrates what happens when a '[use](#)' has a [transform](#) attribute.

Example: 05_17.svg

```
<?xml version="1.0"?>
<svg width="10cm" height="3cm" viewBox="0 0 100 30" version="1.2"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  baseProfile="tiny">
```

```

<desc>Example Use03 - 'use' with a 'transform' attribute</desc>
<defs>
  <rect xml:id="MyRect" x="0" y="0" width="60" height="10"/>
</defs>
<rect x=".1" y=".1" width="99.8" height="29.8"
      fill="none" stroke="blue" stroke-width=".2" />
<use xlink:href="#MyRect"
      transform="translate(20,2.5) rotate(10)" />
</svg>

```



The visual effect would be equivalent to the following document:

Example: 05_18.svg

```

<?xml version="1.0"?>
<svg width="100%" height="100%" viewBox="0 0 100 30" version="1.2"
      xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
      baseProfile="tiny">
  <desc>Example Use03-GeneratedContent - 'use' with a 'transform' attribute</desc>
  <!-- 'defs' section left out -->
  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />
  <!-- Start of generated content. Replaces 'use' -->
  <g transform="translate(20,2.5) rotate(10)">
    <rect x="0" y="0" width="60" height="10"/>
  </g>
  <!-- End of generated content -->
</svg>

```

5.7 The 'image' element

The **'image'** element indicates that the contents of a complete document are to be rendered into a given rectangle within the current user coordinate system. In SVG Tiny 1.2, the **'image'** must reference content that is a raster image format, such as PNG and JPG. SVG Tiny 1.2 does not allow an SVG document to be referenced by the **'image'** element; instead, authors should use the **'animation'** element for referencing SVG Documents. [Conforming SVG viewers](#) must support PNG and JPEG image file formats. Other image file formats may be supported.

For details of the required JPEG support see the [JPEG Support Appendix](#). PNG support is required as defined the PNG specification [\[PNG\]](#).

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

The **'image'** element supports the **'opacity'** property for controlling the image opacity. The **'fill-opacity'** property does not affect the rendering of an image.

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

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

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

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

The SVG specification does not specify when an image that is not being displayed should be loaded. An **SVG User Agent** is not required to load image data for an image that is not displayed (e.g. is outside the initial document **viewport**), except when that image is contained inside a subtree for which **'externalResourcesRequired'** is set to **true**. However, it should be noted that this may cause a delay when an image becomes visible for the first time. In the case where an author wants to suggest that the **SVG User Agent** loads image data before it is displayed, they should use the **'prefetch'** element.

Note that an **SVG User Agent** may choose to incrementally render an image as it is loading but is not required to do so.

Schema: image

```
<define name='image'>
  <element name='image'>
    <ref name='image.AT' />
    <ref name='image.CM' />
  </element>
</define>

<define name='image.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Media.attr' />
  <ref name='svg.XLinkEmbed.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.Opacity.attr' />
  <ref name='svg.XYWH.attr' />
  <ref name='svg.PAR.attr' />
  <ref name='svg.ContentType.attr' />
</define>

<define name='image.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
      <ref name='svg.Handler.group' />
      <ref name='svg.Discard.group' />
    </choice>
  </zeroOrMore>
</define>
```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of one corner of the rectangular region.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

y = "[<coordinate>](#)"

The y-axis coordinate of one corner of the rectangular region.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

width = "[<length>](#)"

The width of the rectangular region.

A negative value is an error (see [Error processing](#)). A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

height = "[<length>](#)"

The height of the rectangular region.

A negative value is an error (see [Error processing](#)). A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

xlink:href = "[<iri>](#)"

An [IRI Reference](#). An empty attribute value (`xlink:href=""`) disables rendering of the element. If the attribute is not specified, the effect is as if an empty value ("") was specified.

Animatable: yes.

type = "[<media/type>](#)"

A hint about the expected Internet Media Type of the raster image. Implementations may choose to not fetch images of formats that they do not support. Note that if an Internet Media type returned by the server, the server metadata is authoritative over the type attribute. See TAG finding [Authoritative Metadata](#), section 5 Metadata Hints in Specifications.

Animatable: no.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

An example:

Example: 05_21.svg

```
<?xml version="1.0"?>
<svg width="100%" height="100%" version="1.2"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  baseProfile="tiny">
  <desc>This graphic links to an external image
```

```

</desc>
<image x="200" y="200" width="100" height="100"
  xlink:href="externalImage.png">
  <title>External image</title>
</image>
</svg>

```

5.8 Conditional processing

5.8.1 Conditional processing overview

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

Schema: conditional

```

<define name='svg.Conditional.attr' combine='interleave'>
  <optional>
    <attribute name='requiredFeatures' svg:animatable='false' svg:inheritable='false'>
      <ref name='FeatureList.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='requiredExtensions' svg:animatable='false' svg:inheritable='false'>
      <ref name='ExtensionList.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='requiredFormats' svg:animatable='false' svg:inheritable='false'>
      <ref name='FormatList.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='requiredFonts' svg:animatable='false' svg:inheritable='false'>
      <ref name='FontList.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='systemLanguage' svg:animatable='false' svg:inheritable='false'>
      <ref name='LanguageCodes.datatype' />
    </attribute>
  </optional>
</define>

```

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

Example **systemLanguage** below displays one of three text strings (in Welsh, Greek, or Spanish) if those are the users preferred languages. Otherwise, in this example, it displays nothing.

Example: systemLanguage.svg

```

<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" id="svg-root" width="100%" height="100%" viewBox="0 0 170 200"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>systemLanguage example</title>
  <switch>
    <g systemLanguage="cy">
      <text x="20" y="220" xml:lang="cy" font-size="20">Pam dydyn nhw ddim yn siarad
        Cymraeg?</text>
    </g>
    <g systemLanguage="el">
      <text x="20" y="220" xml:lang="el-GR" font-size="22">Μα γιατί δεν μπορούν να μιλάσουν
        Ελληνικά; </text>
    </g>
    <g systemLanguage="es">
      <text x="20" y="220" xml:lang="es-ES" font-size="18">¿Por qué no pueden simplemente
        hablar en castellano ?</text>
    </g>
  </switch>
</svg>

```

Similar to the **'display'** property, conditional processing attributes only affect the direct rendering of elements and do not prevent elements from being successfully referenced by other elements (such as via a **'use'**). The conditional processing attributes in the shadow tree are processed normally. Conditional properties do not effect evaluation of **'script'** elements, whose logic is run independently of conditional properties of any ancestor **'switch'** elements.

Furthermore, the following rules must be applied:

- **'requiredFeatures'**, **'requiredExtensions'**, **'systemLanguage'**, **'requiredFormats'** and **'requiredFonts'** attributes do not apply to elements that are not part of the **rendering tree** such as **'defs'** or **'linearGradient'** elements but do apply to the **'audio'** element.
- **'requiredFeatures'**, **'requiredExtensions'**, **'systemLanguage'**, **'requiredFormats'** and **'requiredFonts'** attributes affect **'animate'**, **'animateColor'**, **'animateMotion'**, **'animateTransform'**, and **'set'** elements. If the conditional statement on these animation elements fails, the animation will never be triggered.
- **'requiredFeatures'**, **'requiredExtensions'**, **'systemLanguage'**, **'requiredFormats'** and **'requiredFonts'** attributes affect the **'discard'** element. If the conditional statement on **'discard'** fails, the element referred to will not be discarded.

5.8.2 The 'switch' element

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

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

Schema: switch

```
<element name='switch'>
  <ref name='switch.AT' />
  <!-- the content model for switch is defined as part
       of the element that can contain it -->
</element>

<define name='switch.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.Focus.attr' />
</define>
```

For more information and an example, see [Embedding foreign object types](#).

Attribute definitions:

requiredFeatures = *"list-of-features"*
See [attribute definition](#) for description.

Animatable: no.

requiredExtensions = *"list-of-extensions"*
See [attribute definition](#) for description.

Animatable: no.

systemLanguage = *"list-of-languages"*
See [attribute definition](#) for description.

Animatable: no.

requiredFormats = *"list-of-format-definitions"*
See [attribute definition](#) for description.

Animatable: no.

requiredFonts = *"list-of-font-names"*
See [attribute definition](#) for description.

Animatable: no.

focusable = *"true" | "false" | "auto"*
See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes
See [definition](#).

5.8.3 The 'requiredFeatures' attribute

Definition of **'requiredFeatures'**:

requiredFeatures = *"list-of-features"*

The value is a list of feature strings, with the individual values separated by white space. Determines whether all of the named *features* are supported by the **SVG User Agent**. Only feature strings defined in an existing version of the SVG specification at the time the document is authored (such as the [Feature String](#) appendix) should be used, while third party extension features that are not part of an SVG standard should be indicated using the **'requiredExtensions'** attribute instead. If all of the given features are supported, then the attribute evaluates to "true"; otherwise, the current element and its children are skipped and thus will not be rendered.

Animatable: no.

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

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

5.8.4 The 'requiredExtensions' attribute

The **'requiredExtensions'** attribute defines a list of required language extensions. Language extensions are capabilities within an **SVG User Agent** that go beyond the feature set defined in this specification. Each extension is identified by a **IRI Reference**.

Definition of `requiredExtensions`:

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

The value is a list of [IRI References](#) which identify the required extensions, with the individual values separated by white space. Determines whether all of the named *extensions* are supported by the user agent. If all of the given extensions are supported, then the attribute evaluates to "true"; otherwise, the current element and its children are skipped and thus will not be rendered.

[Animatable](#): no.

If a given [IRI Reference](#) contains white space within itself, that white space must be escaped.

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

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

5.8.5 The `'systemLanguage'` attribute

Definition of `'systemLanguage'`:

`systemLanguage` = "*list-of-languages*"

The value is a comma-separated list of language names as defined in [\[RFC3066\]](#)

[Animatable](#): no.

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

Evaluates to "false" otherwise.

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

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

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

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

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

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

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

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

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

5.8.6 The `'requiredFormats'` attribute

Many resources, especially media such as audio and video, have a wide range of formats. As it is often not possible to require support for a particular format, due to legal or platform restrictions, it is often necessary to provide alternatives so that [SVG User Agents](#) can choose the format they support.

The `'requiredFormats'` attribute is a generic conditional processing attribute that can be used to enable or disable particular branches in the SVG document. It defines a list of resource formats. Each format is defined by the format definition with the syntax varied according to the specific type of resource. The [SVG User Agent](#) must support all of the resource types for the attribute to evaluate to "true".

Definition of `'requiredFormats'`:

`requiredFormats` = "*list-of-format-definitions*"

Each format definition is separated by whitespace. A format definition can be a MIME-type beginning `"image/"`, `"video/"` or `"audio/"`, or must use one of the following formats:

image(<mime-type>):
Test the MIME-type as an image format.
video(<mime-type>):
Test the MIME-type as a video format.
audio(<mime-type>):
Test the MIME-type as an audio format.
font(<mime-type>):
Test the MIME-type as a font format.
script(<mime-type>):
Test the MIME-type as scripting language type.
foreignObject(namespaceURI):
Test if the namespace is understood in the SVG [foreignObject](#) element.

For a list of MIME types for audio/video codecs, see the [IANA registry](#) and [RFC2361](#).

Given that several important file formats are still not registered or not specific enough, the following format definitions are also understood:

- o **font(truetype)**
- o **font(type1)**
- o **font(opentype)**

The following ['requiredFormats'](#) must always evaluate to "true" in compliant SVG viewers:

- o font(image/svg+xml)
- o image/png
- o image/peg
- o image/svg+xml,

[Animatable](#): no.

If the attribute is not present, then its implicit return value is "true". If a null string or empty string value is given to attribute ['requiredFormats'](#), the attribute returns "false". Format definitions that are not understood by the [SVG User Agent](#) return "false".

['requiredFormats'](#) is often used in conjunction with the ['switch'](#) element. If the ['requiredFormats'](#) is used in other situations, then it represents a simple switch on the given element whether to render the element or not.

5.8.7 The ['requiredFonts'](#) attribute

If the author wishes to have complete control over the appearance and location of text in the document then they must ensure that the correct font is used when rendering the text. This can be achieved by using SVG Fonts and embedding the font in the document. However, this is not practical in all cases, especially when the number of glyphs used is very large or if the licensing of the font forbids such embedding.

Definition of ['requiredFonts'](#):

requiredFonts = "list-of-font-names"

The ['requiredFonts'](#) attribute is a generic conditional processing attribute that can be used to enable or disable particular branches in the SVG document. It defines a list of fonts, separated by commas. The [SVG User Agent](#) must have access to all of the fonts, either installed on the system or as an SVG font defined or embedded within the document, for the attribute to evaluate to "true". ['requiredFonts'](#) uses same syntax as the ['font-family'](#) property, for example when processing quoted strings, multiple, leading and trailing spaces, and case sensitivity.

[Animatable](#): no.

If the attribute is not present, then its implicit return value is "true". If a null string or empty string value is given to attribute ['requiredFonts'](#), the attribute returns "false".

['requiredFonts'](#) is often used in conjunction with the ['switch'](#) element. If the ['requiredFonts'](#) is used in other situations, then it represents a simple switch on the given element whether to render the element or not.

5.9 External Resources

5.9.1 The ['externalResourcesRequired'](#) attribute

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

Attribute ['externalResourcesRequired'](#) is available on all [container elements](#) except ['defs'](#) and on all elements which potentially can reference external resources. It specifies whether referenced resources that are not part of the current document are required for proper rendering of the given element.

Attribute definition:

externalResourcesRequired = "false" | "true"

false

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

true

Indicates that resources external to the current document are required. If an external resource is not available (the request for the required resource times out), progressive rendering is suspended, the document's [load](#) event is not fired and the document goes into an error state (see [Error processing](#)). The document remains in an error state until all required resources become available.

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

Because setting `externalResourcesRequired="true"` on a [container element](#) can have the effect of disabling progressive display of the contents of that container, tools that generate SVG content should normally not just set `externalResourcesRequired="true"` on the `'svg'` element on a universal basis. Instead, it is better to specify `externalResourcesRequired="true"` on those particular elements which specifically need the availability of external resources in order to render properly.

For `externalResourcesRequired`: [Animatable](#): *no*.

5.9.2 Progressive Rendering

When progressively downloading a document, an [SVG User Agent](#) conceptually builds a tree of nodes in various states. The possible states for these nodes are unresolved, resolved and error.

This description uses two conceptual parsing events to simplify the prose in explaining the intended behaviour of progressive rendering. The events referred to in the following prose are the 'start element' and 'end element' events. The 'start element' event is considered to be triggered when the Start-Tag or an Empty-Element Tag is read. The 'end element' event occurs either immediately following the 'start element' event in the case of an Empty-Element Tag or when the End-Tag is read.

When loading a document following the 'start element' event on a node, that node becomes part of the document tree in the unresolved state. If the node's dependencies are successfully resolved, then the node enters the resolved state or if the node's dependencies are found to be in error, then the node enters the error state.

Node dependencies include both children content (like the child elements on a `'g'`) and resources (e.g. images referenced by an `'image'`) referenced from that node or from its children. Children become resolved when the 'end element' event occurs on an element. Resources become resolved (or found in error) by an [SVG User Agent](#) specific mechanism.

[SVG User Agents](#) must implement progressive rendering although there is no minimum rendering update frequency required for conformance. Implementations should find their own balance between processing the changes in the document tree and rendering the tree to produce a smooth rendering avoiding significant pauses. The following rules apply to progressive rendering:

- The [SVG User Agent](#) has the opportunity to update the rendering following each 'start element' and/or 'end element' event, i.e. each time the [SVG User Agent](#) parses a Start-Tag, Empty-Element Tag or End-Tag.
- The [SVG User Agent](#) renders the conceptual document tree nodes in document order up to, and not including the first node in the 'unresolved' state which has `externalResourcesRequired` set to `true`. Nodes in the 'resolved' state are always rendered. Nodes in the unresolved state but with `externalResourcesRequired` set to `false` are rendered in their current state. If the node has no rendering (e.g. an `'image'` pending a resource), then nothing is rendered for that node.
- If a node enters the error state then the document enters the error state and progressive rendering stops.

Note that even if the [SVG User Agent](#) has the opportunity to update the rendering after each start/end element event there are situations where such an update shouldn't be done. E.g. `'font'` element children (`'font-face'`, `'h kern'`, `'missing-glyph'`, `'glyph'`) should not cause an update of the document rendering, only the 'end element' event on the `'font'` element should cause a document rendering as for other node types.

Example

Example: progRend01.svg

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2"
  baseProfile="tiny" xml:id="svg-root" width="100%" height="100%" viewBox="0 0 480 360">
  <desc>externalResourcesRequired example.</desc>
  <g externalResourcesRequired="true">
    <rect xml:id="rect_1" width="5" height="7"/>
    ...
    <rect xml:id="rect_1000" width="5" height="7"/>

    <image xlink:href="myImage.png" width="5" height="7"
      externalResourcesRequired="true"/>
    <rect xml:id="rect_1001" width="5" height="7"/>
  </g>
</svg>
```

In this example, the `'g'` element rendering will start when the `'g'` End-Tag has been parsed and processed and when all the resources needed by its children have been resolved. This means that the group's rendering will start when the group has been fully parsed and `myImage.png` has been successfully retrieved.

Forward reference of [use](#) element

Example: progRend02.svg

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2"
  baseProfile="tiny" xml:id="svg-root" width="100%" height="100%" viewBox="0 0 480 360">
  <desc>Forward reference of use element</desc>
  <use xlink:href="#myRect" x="200" fill="green"/>
  <circle cx="450" cy="50" r="50" fill="yellow" />

  <g fill="red">
    <rect xml:id="myRect" width="100" height="100" />
  </g>
</svg>
```

In this example, the various renderings will be (the rendering state follows the semi-colon):

1. `'use'`: 'start element' : empty
2. `'circle'`: 'start element': yellow circle

3. **'g'**:start element': no update
4. **'rect'**:start element' (use reference becomes resolved): green rect, yellow circle, red rect

Forward reference on **'use'** with **externalResourcesRequired="true"**

Example: progRend03.svg

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2"
  baseProfile="tiny" xml:id="svg-root" width="100%" height="100%" viewBox="0 0 480 360">
  <desc>Forward reference on use with eRR=true</desc>
  <use xlink:href="#myGroup" x="200" fill="green"
    externalResourcesRequired="true"/>
  <circle cx="450" cy="50" r="50" fill="yellow" />

  <g fill="red">
    <g xml:id="myGroup">
      <rect xml:id="myRect" width="100" height="100" />
      <use xlink:href="#myRect" x="50" fill="purple"/>
    </g>
  </g>
</svg>
```

1. **'use'**:start element' : empty
2. **'circle'**:start element': empty **'use'** is unresolved, **externalResourcesRequired="true"**, rendering is stopped at the **'use'**)
3. **'g'**:start element': no update
4. **g.myGroup**:start element': no update (use is resolved but **externalResourcesRequired="true"** so rendering will not proceed until that reference enters the resolved state)
5. **'rect'**:start element': no update
6. **'use'**:start element': no update
7. **g.myGroup**:end element' (#myGroup reference becomes resolved, rendering can proceed): green rect, purple rect, yellow circle, red rect, purple rect.

Forward reference with **'use'** to an element under a container with **externalResourcesRequired="true"**

Example: progRend04.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2" baseProfile="tiny" xml:id="svg-root"
width="100%" height="100%" viewBox="0 0 480 360">
  <desc>Forward Reference to a use under a container with eRR=true</desc>
  <use xlink:href="#myRect" x="200" fill="green"/>
  <circle cx="250" cy="50" r="50" fill="pink" />
  <g fill="red" externalResourcesRequired="true">
    <circle cx="450" cy="50" r="50" fill="yellow" />
    <rect xml:id="myRect" width="100" height="100" />
  </g>
</svg>
```

1. **'use'**:start element' : empty
2. **pink.circle**:start element': pink circle
3. **'g'**:start element': no update (rendering is suspended because of **externalResourcesRequired="true"** on the **'g'** element, i.e. because the children of **'g'** are not resolved at the time of parsing of the start tag of **'g'**).
4. **yellow.circle**:start element': no update (rendering suspended because of **'g'**)
5. **myRect.rect**:start element': no update
6. **'g'**:end element'. Resources referenced by **'use'** become resolved and can be rendered. Rendering can proceed: green rect, pink circle, yellow circle, red rect

Font Resolution Example

Example: progRend05.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2"
  baseProfile="tiny" xml:id="svg-root" width="100%" height="100%" viewBox="0 0 480 360">
  <desc>Font Resolution Example</desc>
  <text x="240" y="230" text-anchor="middle" font-size="120"
    font-family="fontC, fontB, fontA">A</text>

  <defs>
    <font xml:id="fontA" horiz-adv-x="224" >
      <font-face
        font-family="fontA"
        units-per-em="1000"
        panose-1="0 0 0 0 0 0 0 0 0"
        ascent="917"
        descent="-250"
        alphabetic="0" />
      <missing-glyph horiz-adv-x="800" d="..." />
      <glyph unicode="A" glyph-name="A" d="..." />
    </font>

    <font xml:id="fontB" horiz-adv-x="224">
      <font-face
        font-family="fontB"
        units-per-em="1000"
        panose-1="0 0 0 0 0 0 0 0 0"
        ascent="917"
        descent="-250"
        alphabetic="0" />
    </font>
  </defs>
</svg>
```

```

<missing-glyph horiz-adv-x="800" d="..." />
<glyph unicode="A" glyph-name="B" d="..." />

</font>

<font xml:id="fontC" horiz-adv-x="224" >
  <font-face
    font-family="fontC"
    units-per-em="1000"
    panose-1="0 0 0 0 0 0 0 0 0"
    ascent="917"
    descent="-250"
    alphabetic="0" />
  <missing-glyph d="..." />
  <glyph unicode="A" glyph-name="C" d="..." />
</font>
</defs>

</svg>

```

Progressive rendering:

1. **'text'**.start element: 'A' rendered with the default font
2. **'defs'**.start element: no update
3. **fontA.font**.start element: no update
4. **fontA.font-face**.start element: no update
5. **fontA.missingGlyph**.start element: no update
6. **fontA.glyphA**.start element: no update
7. **fontA.font**.end element: 'A' rendered with fontA (represents current document state rendering)
8. **fontB.font**.start element: no update
9. **fontB.font-face**.start element: no update
10. **fontB.missingGlyph**.start element: no update
11. **fontB.glyphA**.start element: no update
12. **fontB.font**.end element: 'A' rendered with fontB (represents current document state rendering)
13. **fontC.font**.start element: no update
14. **fontC.font-face**.start element: no update
15. **fontC.missingGlyph**.start element: no update
16. **fontC.glyphA**.start element:
17. **fontC.font**.end element: 'A' rendered with fontC (represents current document state rendering)

5.9.3 The 'prefetch' element

SVG 1.1 did not specify when an [SVG User Agent](#) should begin download referenced media. This lead to implementation differences particularly when the media was not used in the initial document state (e.g. it was offscreen or hidden). SVG 1.2 does not require [SVG User Agents](#) to download referenced media that is not visual at the time the document is loaded, unless those media are contained inside a subtree for which [externalResourcesRequired](#) is set to **true**. This means there may be a pause to download the file the first time a piece of media is displayed. More advanced [SVG User Agents](#) may wish to predict that particular media streams will be needed and therefore download them in anticipation.

SVG 1.2 therefore adds functionality to allow content developers to suggest prefetching content from the server before it is needed to improve the rendering performance of the document. The ['prefetch'](#) element has been reused from Section 4.4 of SMIL 2.0 ([The PrefetchControl Module](#)) with the following modifications:

- no `percent-value` is allowed
- the `'xlink:href'` is permitted to point into the document in which the ['prefetch'](#) element appears so that this feature can be used as a hint indicating how much of the document is required before play-back can start
- in order to adequately enable [local IRI references](#), the `'mediaCharacterEncoding'` and `'mediaContentEncodings'` attributes have been added

The ['prefetch'](#) element will give a suggestion or hint to a [SVG User Agent](#) that media will be used in the future and the author would like part or all of it fetched ahead of time to make the document playback smoother. As it is a hint, user-agents may ignore ['prefetch'](#) elements, though doing so may cause an interruption in the document playback when the resource is needed. It gives authoring tools and authors the ability to schedule retrieval of resources when they think that there is available bandwidth or time to do it.

When instead of referring to external media, ['prefetch'](#) refers to the same document it occurs in, then it can only reference a top level ['g'](#) element. A top level ['g'](#) element is a ['g'](#) element that is a direct child of the ['svg'](#) element.

To enable smooth playback during progressive downloading in this scenario, it is recommended that each adjacent top level ['g'](#) element contain adjacent chronological scenes in the animation. In this case the ['prefetch'](#) element must appear in a ['defs'](#) block before all defined ['g'](#) elements in the document. In such cases, ['prefetch'](#) is used to tell the [SVG User Agent](#) how much it needs to buffer in order to be able to play content back in a smooth and predictable manner.

Schema: prefetch

```

<define name='prefetch'>
  <element name='prefetch'>
    <ref name='prefetch.AT' />
    <ref name='prefetch.CM' />
  </element>
</define>

<define name='prefetch.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.XLinkRequired.attr' />
  <optional>
    <attribute name='mediaSize' svg:animatable='false' svg:inheritable='false'>
      <ref name='NumberOrPercentage.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='mediaTime' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>

```

```

<optional>
  <attribute name='mediaCharacterEncoding' svg:animatable='false' svg:inheritable='false'><text/></attribute>
</optional>
<optional>
  <attribute name='mediaContentEncodings' svg:animatable='false' svg:inheritable='false'><text/></attribute>
</optional>
<optional>
  <attribute name='bandwidth' svg:animatable='false' svg:inheritable='false'>
    <ref name='NumberOrPercentage.datatype' />
  </attribute>
</optional>
</define>

<define name='prefetch.CM'>
  <zeroOrMore>
    <ref name='svg.Desc.group' />
  </zeroOrMore>
</define>

```

Attribute definitions:

mediaSize = "**<number>**"

Defines how much of the media to fetch in bytes as a function of the file size of the media.

When '**prefetch**' refers to a resource in the same document (i.e. a top level '**g**' element), the **mediaSize** attribute indicates the size in bytes of the '**g**' element and its children. That size corresponds to the encodings used when transmitting the document. If the document is encoded in UTF-8 [RFC 3629] and gzipped, then the size of the gzipped UTF-8 fragment applies. If that same document were decompressed and transcoded to UTF-16 [RFC 2781] the hints will become stale. Since streaming hints are to be used primarily in streaming scenarios, it is not expected that hint staleness will occur frequently.

If the attribute is not specified, the behavior is that the entire media should be fetched.

Animatable: no.

mediaTime = "**<time>**"

Defines how much of the media to fetch as a function of the duration of the media. For discrete media (non-time based media like image/png) using this attribute causes the entire resource to be fetched.

When '**prefetch**' refers to a resource in the same document (i.e. a top level '**g**' element), this is the active duration of the referenced element. In cases where the exact active duration can not be calculated before hand (e.g. end of an animation depends on user interaction), it is suggested that the content author estimate the minimum active duration for the referenced element. This estimate, even if zero, will allow the **SVG User Agent** to calculate how much of the overall document to download before beginning playback in a streaming scenario.

If the attribute is not specified, the behavior is that the entire media should be fetched.

Animatable: no.

bandwidth = "**<number>**"

Defines how much network bandwidth, in bits per second, the **SVG User Agent** should use when doing the prefetch. If the attribute is not specified, all available bandwidth should be used.

Animatable: no.

mediaCharacterEncoding = "**<string>**"

Indicates the XML character set encoding (UTF-8, ISO-8859-1, etc.) that the '**mediaSize**' attribute applies to. Tools that produce SVG should include this attribute if they specify the '**mediaSize**' attribute. The main use of this attribute is to know what character encoding was used when measuring '**mediaSize**' so that staleness of the hints may be easily detected. If the attribute is not specified, the encoding used is that returned by the server.

Animatable: no.

mediaContentEncodings = "**<list of content encodings>**"

The '**mediaContentEncodings**' attribute is a white space separated list of the content encodings as defined in section 3.5 of [HTTP/1.1] (gzip, compress, etc.) that the '**mediaSize**' attribute applies to. The order of the list is the order in which the content encodings were applied to encode the data. Note that while situations in which multiple content codings are applied are currently rare, they are allowed by HTTP/1.1 and thus that functionality is supported by SVG. Tools that produce SVG must include this attribute *if* they specify the '**mediaSize**' attribute *and* the Content-Encoding is other than the identity encoding. The main use of this attribute is to know what parameters were used when measuring '**mediaSize**' so that staleness of the hints may be easily detected. If the '**mediaContentEncodings**' attribute is not specified it is as if the '**identity**' encoding value from HTTP/1.1 has been specified. This indicates that no transformation (i.e. encodings) at all has been used.

Animatable: no.

xlink:href = "**<iri>**"

An **IRI reference** to the resource to prefetch. An empty attribute value (**xlink:href=""**) means that no prefetching will occur. If the attribute is not specified, the effect is as if an empty value ("") was specified.

Animatable: no.

When '**prefetch**' refers to external media, if both '**mediaSize**' and '**mediaTime**' are specified, then '**mediaSize**' is used and '**mediaTime**' is ignored.

When '**prefetch**' refers to a resource in the same document (i.e. a top level '**g**' element), both the '**mediaSize**' and '**mediaTime**' attributes can be used together by a more advanced **SVG User Agent** to determine how much it needs to buffer in order to be able to play content back in a smooth manner.

Below is an example of the '**prefetch**' element when it refers to external media:

Example: prefetch01.svg

```

<svg width="400" height="300" version="1.2"
  xmlns="http://www.w3.org/2000/svg" baseProfile="tiny"
  xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>
    Prefetch the large images before starting the animation
    if possible.
  </desc>

  <defs>

```

```

<prefetch xlink:href="http://www.example.com/images/hugel.png"/>
<prefetch xlink:href="http://www.example.com/images/huge2.png"/>
<prefetch xlink:href="http://www.example.com/images/huge3.png"/>
</defs>

<image x="0" y="0" width="400" height="300"
  xlink:href="http://www.example.com/images/hugel.png"
  display="none">

  <set attributeName="display" to="inline" begin="10s"/>

  <animate attributeName="xlink:href" values="
    http://www.example.com/images/hugel.png;
    http://www.example.com/images/huge2.png;
    http://www.example.com/images/huge3.png"
    begin="15s" dur="30s"/>
</image>

</svg>

```

Below is an example of the **'prefetch'** element when it refers to a resource (i.e. a top level **'g'** element in the same document):

Example: prefetch02.svg

```

<?xml version="1.0" encoding="utf-16"?>
<svg width="400" height="300" version="1.2"
  xmlns="http://www.w3.org/2000/svg" baseProfile="tiny"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  timelineBegin="onStart"
  playbackOrder="forwardOnly">

  <desc>
    Example of using SVGT 1.2 features for smooth playback
    during progressive downloading.
  </desc>

  <defs>

    <prefetch xlink:href="#scene1"
      mediaCharacterEncoding="UTF-16"
      mediaTime="5s" mediaSize="94230" />

    <prefetch xlink:href="#scene2"
      mediaCharacterEncoding="UTF-16"
      mediaTime="10s" mediaSize="283474" />

    <prefetch xlink:href="#scene3"
      mediaCharacterEncoding="UTF-16"
      mediaTime="5s" mediaSize="62763" />

  </defs>

  <g id="scene1">
    <discard begin="6s"/>
    <!-- graphics for scene 1 go here -->
  </g>

  <g id="scene2">
    <discard begin="16s"/>
    <!-- graphics for scene 2 go here -->
  </g>

  <g id="scene3">
    <discard begin="21s"/>
    <!-- graphics for scene 3 go here -->
  </g>

</svg>

```

5.10 Common attributes

5.10.1 Attributes common to all elements: 'class', 'id', 'xml:id' and 'xml:base'

The **'class'**, **'id'**, **'xml:id'** and **'xml:base'** attributes are available on all elements required by SVG Tiny 1.2:

Attribute definitions:

class = "list"

This attribute indicates membership in one or more sets. Multiple set names must be separated by white space characters.

Animatable: yes.

id = "name"

XML attribute for assigning a unique *name* to an element. Refer to the "Extensible Markup Language (XML) 1.1" Recommendation [\[XML11\]](#). It is recommended that new content should use **'xml:id'** instead.

Animatable: no.

xml:id = "name"

Standard XML attribute for assigning a unique *name* to an element. Refer to the "xml:id Version 1.0" [\[xml:id\]](#). Recommended for new content.

Animatable: no.

xml:base = "iri"

Specifies a base **IRI** other than the base **IRI** of the document or external entity. Refer to the "XML Base" specification [\[XML-BASE\]](#).

Animatable: no.

It is strongly recommended that SVG Generators only use ['xml:id'](#) to assign identity to elements. For backwards compatibility purposes, one may also specify the ['id'](#) attribute but such an approach is not without issues:

- it is expected that the ['id'](#) attribute will be deprecated in a future version of this specification, and therefore that SVG Generators should transition to using ['xml:id'](#) only as early as possible;
- there remains only one single [id](#) field on the [SVGElement](#) interface, therefore when it is updated both attributes must be updated too as a consequence;
- consequently, since [SVGElement::id](#) is updated when the attribute values are modified using the [setAttributeNS\(\)](#), [setTraitNS\(\)](#), or [setTrait\(\)](#) methods, performing an update on one of those attributes will also update the other;
- when both ['id'](#) and ['xml:id'](#) are specified on the same element but with different values, the resulting behaviour is unspecified and implementation-dependent. It is however recommended that whenever possible, implementations should give precedence to the ['xml:id'](#) attribute.

5.10.2 The ['xml:lang'](#) and ['xml:space'](#) attributes

Elements that might contain character data content have attributes ['xml:lang'](#) and ['xml:space'](#):

Schema: langspace

```
<attribute name='xml:space' svg:animatable='false' svg:inheritable='false'>
  <choice>
    <value>default</value>
    <value>preserve</value>
  </choice>
</attribute>

<attribute name='xml:lang' svg:animatable='false' svg:inheritable='false'>
  <choice>
    <ref name='LanguageCode.datatype' />
    <empty/>
  </choice>
</attribute>
```

Attribute definitions:

xml:lang = "[<languageID>](#)"

Standard XML attribute to specify the language (e.g., English) used in the contents and attribute values of particular elements. The value is either a language tag as defined in IETF Best Current Practice 47 [\[BCP 47\]](#) or the empty string, "". Refer to the "Extensible Markup Language (XML) 1.1" Recommendation [\[XML11\]](#).

Animatable: no.

xml:space = "default" | "preserve"

Standard XML attribute to specify whether white space is preserved in character data. The only possible values are **default** and **preserve**. Refer to the "Extensible Markup Language (XML) 1.1" Recommendation [\[XML11\]](#) and to the discussion [White space handling](#) in SVG.

Animatable: no.

5.11 Core Attribute Module

The Core Attribute Module contains the following attributes:

- [id](#)
- [xml:base](#)
- [xml:lang](#)
- [xml:space](#)
- [class](#)

[\[RNG\]](#) [\[Feature String\]](#)

5.12 Structure Module

The Structure Module contains the following elements:

- [svg](#)
- [g](#)
- [defs](#)
- [desc](#)
- [title](#)
- [metadata](#)
- [use](#)

[\[RNG\]](#) [\[Feature String\]](#)

5.13 Conditional Processing Module

The Conditional Processing Module contains the following element:

- [switch](#)

[RNG] [[Feature String](#)]

5.14 Conditional Processing Attribute Module

The Conditional Processing Attribute Module contains the following attributes:

- [requiredFeatures](#)
- [requiredFonts](#)
- [requiredFormats](#)
- [requiredExtensions](#)
- [systemLanguage](#)

[RNG] [[Feature String](#)]

5.15 Image Module

The Image Module contains the following element:

- [image](#)

[RNG] [[Feature String](#)]

5.16 Prefetch Module

The Prefetch Module contains the following element:

- [prefetch](#)

[RNG] [[Feature String](#)]

5.17 Discard Module

The Discard Module contains the following element:

- [discard](#)

[RNG] [[Feature String](#)]

5.18 ExternalResourcesRequired Attribute Module

The ExternalResourcesRequired Attribute Module contains the following attributes:

- [externalResourcesRequired](#)

[RNG] [[Feature String](#)]

6 Styling

Contents

- [6.1 SVG's styling properties](#)
- [6.2 Usage scenarios for styling](#)
- [6.3 Specifying properties using the presentation attributes](#)
- [6.4 Styling with XSL](#)
- [6.5 Case sensitivity of property names and values](#)
- [6.6 Facilities from CSS and XSL used by SVG](#)
- [6.7 Property inheritance and computation](#)

6.1 SVG's styling properties

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

- Parameters which are clearly visual in nature and thus lend themselves to styling. Examples include all attributes that define how an object is "painted," such as fill and stroke colors, line widths and dash styles.
- Parameters having to do with text styling such as **'font-family'** and **'font-size'**.
- Parameters for interactivity and multimedia, such as **'pointer-events'** and **'audio-level'**.

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

The following properties are shared between CSS2 and SVG. Apart from **'display'**, these properties are also defined in XSL:

- [Font properties](#):
 - **'font-family'**
 - **'font-size'**
 - **'font-style'**
 - **'font-weight'**
- Other properties for visual media:
 - **'color'** is used to provide a potential indirect value (**currentColor**) for the **'fill'**, **'stroke'**, **'stop-color'** properties.
 - **'display'**
 - **'visibility'**

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

- [Gradient](#) properties:
 - **'stop-color'**
 - **'stop-opacity'**
- [Interactivity](#) properties:
 - **'pointer-events'**
- [Multimedia](#) properties:
 - **'audio-level'**
- [Color and Painting](#) properties:
 - **'color-rendering'**

- 'fill'
- 'fill-opacity'
- 'fill-rule'
- 'image-rendering'
- 'shape-rendering'
- 'solid-color'
- 'solid-opacity'
- 'stroke'
- 'stroke-dasharray'
- 'stroke-dashoffset'
- 'stroke-linecap'
- 'stroke-linejoin'
- 'stroke-miterlimit'
- 'stroke-opacity'
- 'stroke-width'
- 'text-rendering'
- 'vector-effect'
- 'viewport-fill'
- 'viewport-fill-opacity'
- Text properties:
 - 'display-align'
 - 'line-increment'
 - 'text-anchor'

A table that lists and summarizes the styling properties can be found in the [Property Index](#).

6.2 Usage scenarios for styling

SVG has many usage scenarios, each with different needs. Here are three common usage scenarios:

1. SVG content used as an exchange format (style sheet language-independent):

In some usage scenarios, reliable interoperability of SVG content across software tools is the main goal. Since support for a particular style sheet language is not guaranteed across all implementations, it is a requirement that SVG content can be fully specified without the use of a style sheet language.

2. SVG content generated as the output from XSLT [[XSLT](#)]:

XSLT offers the ability to take a stream of arbitrary XML content as input, apply potentially complex transformations, and then generate SVG content as output. XSLT can be used to transform XML data extracted for instance from databases into an SVG graphical representation of that data. It is a requirement that fully specified SVG content can be generated from XSLT.

3. SVG content styled with CSS [[CSS2](#)]:

CSS is a widely implemented declarative language for assigning styling properties to XML content, including SVG. It represents a combination of features, simplicity and compactness that makes it very suitable for many applications of SVG. SVG Tiny 1.2 does not require support for CSS selectors applied to SVG content. Authors must not rely on the CSS cascade to style documents that are intended to be used with SVG Tiny 1.2 user agents

6.3 Specifying properties using the presentation attributes

For each styling property defined in this specification (see [Property Index](#)), there is a corresponding XML attribute (the **presentation attribute**) with the same name that is available on all relevant SVG elements. For example, SVG has a **'fill'** property that defines how to paint the interior of a shape. There is a corresponding presentation attribute with the same name (i.e., **fill**) that can be used to specify a value for the **'fill'** property on a given element.

The following example shows how the **'fill'** and **'stroke'** properties can be assigned to a rectangle using the **fill** and **stroke** presentation attributes. The rectangle will be filled with red and outlined with blue:

Example: [06_01.svg](#)

```
<?xml version="1.0"?>
<svg width="100%" height="100%" viewBox="0 0 1000 500"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <rect x="200" y="100" width="600" height="300"
    fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

The presentation attributes offer the following advantages:

- **Broad support.** All versions of [Conforming SVG Interpreters](#) and [Conforming SVG Viewers](#) are required to support the presentation attributes.
- **Simplicity.** Styling properties can be attached to elements by simply providing a value for the presentation attribute on the proper elements.
- **Restyling.** SVG content that uses the presentation attributes is highly compatible with downstream processing using XSLT [[XSLT](#)] or supplemental styling by adding CSS style rules to override some of the presentation attributes.
- **Convenient generation using XSLT [[XSLT](#)].** In some cases, XSLT can be used to generate fully styled SVG content. The presentation attributes are compatible with convenient generation of SVG from XSLT.

In some situations, SVG content that uses the presentation attributes has potential limitations versus SVG content that is styled with a style sheet language such as CSS. In other situations, such as when an XSLT style sheet generates SVG content from semantically rich XML source files, the limitations below may not apply.

- **Styling attached to content.** The presentation attributes are attached directly to particular elements, thereby diminishing potential advantages that comes from abstracting styling from content, such as the ability to restyle documents for different uses and environments.
- **Flattened data model.** In and of themselves, the presentation attributes do not offer the higher level abstractions that you get with a styling system, such as the ability to define named collections of properties which are applied to particular categories of elements. The result is that, in many cases, important higher level semantic information can be lost, potentially making document reuse and restyling more difficult.
- **Potential increase in file size.** Many types of graphics use similar styling properties across multiple elements. For example, a company organization chart might assign one collection of styling properties to the boxes around temporary workers (e.g., dashed outlines, red fill), and a different collection of styling properties to permanent workers (e.g., solid outlines, blue fill). Styling systems such as CSS allow collections of properties to be defined once in a file. With the styling attributes, it might be necessary to specify presentation attributes on each different element.

An **important** declaration within a presentation attribute definition is unsupported and causes that attribute to have an unsupported value.

Note: Animation of presentation attributes and animation of properties are related, see the [attributeType](#) attribute definition for more information.

6.4 Styling with XSL

XSL style sheets (see [[XSLT](#)]) define how to transform XML content into something else, usually other XML. When XSLT is used in conjunction with SVG, sometimes SVG content will serve as both input and output for XSL style sheets. Other times, XSL style sheets will take non-SVG content as input and generate SVG content as output.

The following example uses an external XSL style sheet to transform SVG content into modified SVG content. The style sheet sets the **'fill'** and **'stroke'** properties on all rectangles to red and blue, respectively:

mystyle.xsl

Example: 06_02.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg">
  <xsl:output
    method="xml"
    encoding="utf-8"
    doctype-public="-//W3C//DTD SVG 1.1//EN"
    doctype-system="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"/>
  <!-- Add version to topmost 'svg' element -->
  <xsl:template match="/svg:svg">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:attribute name="version">1.1</xsl:attribute>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
  <!-- Add styling to all 'rect' elements -->
  <xsl:template match="svg:rect">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:attribute name="fill">red</xsl:attribute>
      <xsl:attribute name="stroke">blue</xsl:attribute>
      <xsl:attribute name="stroke-width">3</xsl:attribute>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

SVG file to be transformed by mystyle.xsl

Example: 06_03.svg

```
<?xml version="1.0"?>
<svg width="10cm" height="5cm" viewBox="0 0 100 50"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <rect x="20" y="10" width="60" height="30"/>
</svg>
```

SVG content after applying mystyle.xsl

Example: 06_04.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="10cm" height="5cm"
  viewBox="0 0 100 50" version="1.2" baseProfile="tiny">
  <rect x="20" y="10" width="60" height="30" fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

6.5 Case sensitivity of property names and values

Property declarations via [presentation attributes](#) are expressed in XML [[XML11](#)], which is case-sensitive and must match the exact property name. When using a presentation attribute to specify a value for the **'fill'** property, the presentation attribute must be specified as **'fill'** and not **'FILL'** or **'Fill'**. Keyword values, such as **'italic'** in `font-style="italic"`, are also case-sensitive and must be specified using the exact case used in the specification which defines the given keyword. For example, the keyword **"sRGB"** must have lowercase **"s"** and uppercase **"RGB"**.

6.6 Facilities from CSS and XSL used by SVG

SVG shares various relevant properties and approaches common to CSS and XSL, plus the semantics of many of the processing rules.

SVG shares the following facilities with CSS and XSL:

- Shared properties. Many of SVG's properties are shared between CSS2, XSL and SVG. (See [list of shared properties](#)).
- Syntax rules. (The normative references are [\[CSS2 syntax and basic data types\]](#) and [\[The grammar of CSS2\]](#).)
- Allowable data types. (The normative reference is [\[CSS2 syntax and basic data types\]](#)), with the exception that SVG allows [<length>](#) values without a unit identifier. See [Units](#).)
- [Inheritance rules](#).
- The color keywords from CSS2 that correspond to the colors used by objects in the user's environment. (The normative reference is [\[CSS2 system colors\]](#).)

6.7 Property inheritance and computation

SVG supports property inheritance to child elements. Each property states whether it is inherited or not. Inheritable properties inherit the computed value, and not the specified value. For the calculation of computed values, see the definition of each property.

7 Coordinate Systems, Transformations and Units

Contents

- 7.1 [Introduction](#)
- 7.2 [The initial viewport](#)
- 7.3 [The initial coordinate system](#)
- 7.4 [Coordinate system transformations](#)
- 7.5 [Nested transformations](#)
- 7.6 [The transform attribute](#)
 - 7.6.1 [The TransformList value](#)
- 7.7 [Constrained Transformations](#)
 - 7.7.1 [The User Transform](#)
 - 7.7.2 [ViewBox to Viewport transformation](#)
 - 7.7.3 [Element Transform Stack](#)
 - 7.7.4 [The Current Transform Matrix](#)
 - 7.7.5 [The ref\(\) transform value](#)
- 7.8 [The viewBox attribute](#)
- 7.9 [The preserveAspectRatio attribute](#)
- 7.10 [Establishing a new viewport](#)
- 7.11 [Units](#)
- 7.12 [Object bounding box units](#)
- 7.13 [Intrinsic Sizing Properties of the Viewport of SVG content](#)
- 7.14 [Geographic Coordinate Systems](#)

7.1 Introduction

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

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

- a number (usually an integer) that represents the width in "pixels" of the viewport
- a number (usually an integer) that represents the height in "pixels" of the viewport
- (highly desirable but not required) a real number value that indicates the size in real world units, such as millimeters, of a "pixel" (i.e., a *px* unit as defined in [[CSS2 lengths](#)])

Using the above information, the SVG user agent determines the **viewport**, an initial **viewport coordinate system** and an initial **user coordinate system** such that the two coordinate systems are identical. Both coordinate systems are established such that the origin matches the origin of the **viewport** (for the rootmost **viewport**, the **viewport** origin is at the top/left corner), and one unit in the initial coordinate system equals one "pixel" in the **viewport**. (See [Initial coordinate system](#).) The **viewport** coordinate system is also called **viewport space** and the user coordinate system is also called **user space**.

Lengths in SVG are specified as values in user space (e.g. "15") and in a couple of special cases with an additional absolute unit measure (e.g. "15mm"). See [Units](#) for details.

A new user space (i.e., a new current coordinate system) can be established at any place within an SVG document fragment by specifying **transformations** in the form of **transformation matrices** or simple transformation operations such as rotation, skewing, scaling and translation. Establishing new user spaces via [coordinate system transformations](#) are fundamental operations to 2D graphics and represent the usual method of controlling the size, position, rotation and skew of graphic objects.

New viewports also can be established. By [establishing a new viewport](#), you can provide a new reference rectangle for "fitting" a graphic into a particular rectangular area. ("Fit" means that a given graphic is transformed in such a way that its bounding box in user space aligns exactly with the edges of a given [viewport](#).)

7.2 The initial viewport

The SVG user agent negotiates with its parent user agent to determine the [viewport](#) into which the SVG user agent can render the document. In some circumstances, SVG content will be embedded ([by reference or inline](#)) within a containing document. This containing document might include attributes, properties and/or other parameters (explicit or implicit) which specify or provide hints about the dimensions of the [viewport](#) for the SVG content. SVG content itself optionally can provide information about the appropriate [viewport](#) region for the content via the [width](#) and [height](#) XML attributes on the ['svg'](#) element. The negotiation process uses any information provided by the containing document and the SVG content itself to choose the [viewport](#) location and size.

The [width](#) attribute on the ['svg'](#) element establishes the [viewport's](#) width, unless all of the following conditions are met:

- the SVG content is a separately stored resource that is embedded by reference (such as the ['object'](#) element in [\[XHTML\]](#)), or the SVG content is embedded inline within a parent document;
- the parent document is styled using CSS [\[CSS2\]](#) or XSL [\[XSL\]](#);
- CSS-compatible width properties [see [CSS2-VISUDET](#)] or corresponding XSL properties are specified on the referencing element (or [rootmost svg element](#) for inline SVG content).

Under these conditions, the ['svg'](#) element must be treated as a replaced element in a CSS context, so the positioning properties establish the [viewport's](#) width.

Similarly, if CSS-compatible height properties or corresponding XSL properties on the referencing element (or [rootmost svg element](#) for inline SVG content), then these positioning properties establish the [viewport's](#) height; otherwise, the [height](#) attribute on the ['svg'](#) element establishes the viewport's height.

If the [width](#) or [height](#) attributes on the ['svg'](#) element are in [user units](#) (i.e., no unit identifier has been provided), then the value is assumed to be equivalent to the same number of "px" units (see [Units](#)).

In the following example, an SVG graphic is embedded inline within a parent XML document which is formatted using CSS layout rules. The [width="100px"](#) and [height="200px"](#) attributes determine the size of the initial [viewport](#):

Example: 07_01.xml

```
<?xml version="1.0"?>
<parent xmlns="http://www.example.org">
  <!-- SVG graphic -->
  <svg xmlns='http://www.w3.org/2000/svg'
    width="100px" height="200px" version="1.2" baseProfile="tiny">
    <path d="M100,100 Q200,400,300,100"/>
    <!-- rest of SVG graphic would go here -->
  </svg>
</parent>
```

7.3 The initial coordinate system

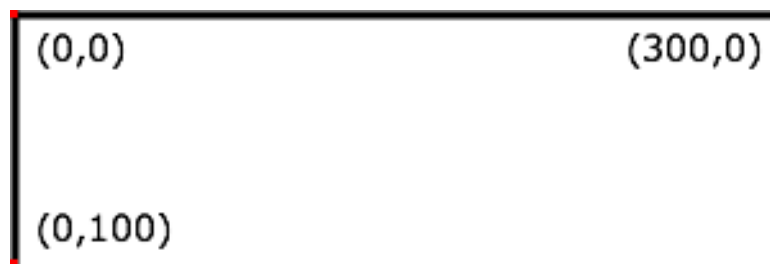
For the ['svg'](#) element, the SVG user agent determines an initial [viewport coordinate system](#) and an initial [user coordinate system](#) such that the two coordinate systems are identical. The origin of both coordinate systems is at the origin of the [viewport](#), and one unit in the initial coordinate system equals one "pixel" (i.e., a *px* unit as defined in [\[CSS2 lengths\]](#)) in the [viewport](#). In most cases, such as stand-alone SVG documents or SVG document fragments embedded ([by reference or inline](#)) within XML parent documents where the parent's layout is determined by CSS [\[CSS2\]](#) or XSL [\[XSL\]](#), the initial [viewport](#) coordinate system (and therefore the initial user coordinate system) has its origin at the top/left of the [viewport](#), with the positive x-axis pointing towards the right, the positive y-axis pointing down, and text rendered with an "upright" orientation, which means glyphs are oriented such that Roman characters and full-size ideographic characters for Asian scripts have the top edge of the corresponding glyphs oriented upwards and the right edge of the corresponding glyphs oriented to the right.

If the SVG implementation is part of a user agent which supports styling XML documents using CSS2-compatible *px* units, then the SVG user agent should get its initial value for the size of a *px* unit in real world units to match the value used for other XML styling operations; otherwise, if the user agent can determine the size of a *px* unit from its environment, it should use that value; otherwise, it should choose an appropriate size for one *px* unit. In all cases, the size of a *px* must be in conformance with the rules described in [[CSS2 lengths](#)].

Example 07_02 below shows that the initial coordinate system has the origin at the top/left with the x-axis pointing to the right and the y-axis pointing down. The initial user coordinate system has one user unit equal to the parent (implicit or explicit) user agent's "pixel".

Example: 07_02.svg

```
<?xml version="1.0"?>
<svg width="300px" height="100px" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example InitialCoords - SVG's initial coordinate system</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <line x1="0" y1="1.5" x2="300" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="100" />
  </g>
  <g fill="red" stroke="none" >
    <rect x="0" y="0" width="3" height="3" />
    <rect x="297" y="0" width="3" height="3" />
    <rect x="0" y="97" width="3" height="3" />
  </g>
  <g font-size="14" font-family="Verdana" >
    <text x="10" y="20">(0,0)</text>
    <text x="240" y="20">(300,0)</text>
    <text x="10" y="90">(0,100)</text>
  </g>
</svg>
```



7.4 Coordinate system transformations

A new user space (i.e., a new current coordinate system) can be established by specifying **transformations** in the form of a [transform](#) attribute on a container element or graphics element or a [viewBox](#) attribute on the **'svg'** element. The [transform](#) and [viewBox](#) attributes transform user space coordinates and lengths on sibling attributes on the given element (see [effect of the transform attribute on sibling attributes](#) and [effect of the viewBox attribute on sibling attributes](#)) and all of its descendants.

Transformations can be nested, in which case the effect of the transformations are cumulative.

Example 07_03 below shows a document without transformations. The text string is specified in the [initial coordinate system](#).

Example: 07_03.svg

```
<?xml version="1.0"?>
<svg width="400px" height="150px" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example OrigCoordSys - Simple transformations: original picture</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (orig coord system)
    </text>
  </g>
</svg>
```

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



ABC (orig coord system)

A diagram showing a coordinate system with a black L-shaped axis. The text "ABC (orig coord system)" is positioned in the upper right area of the coordinate system.

Example 07_04 establishes a new user coordinate system by specifying **transform="translate(50,50)"** on the third 'g' element below. The new user coordinate system has its origin at location (50,50) in the original coordinate system. The result of this transformation is that the coordinate (30,30) in the new user coordinate system gets mapped to coordinate (80,80) in the original coordinate system (i.e., the coordinates have been translated by 50 units in X and 50 units in Y).

Example: 07_04.svg

```
<?xml version="1.0"?>
<svg width="400px" height="150px" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example NewCoordSys - New user coordinate system</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (orig coord system)
    </text>
  </g>
  <!-- Establish a new coordinate system, which is
    shifted (i.e., translated) from the initial coordinate
    system by 50 user units along each axis. -->
  <g transform="translate(50,50)">
    <g fill="none" stroke="red" stroke-width="3" >
      <!-- Draw lines of length 50 user units along
        the axes of the new coordinate system -->
      <line x1="0" y1="0" x2="50" y2="0" stroke="red" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (translated coord system)
    </text>
  </g>
</svg>
```



ABC (orig coord system)

A diagram showing two coordinate systems. The original system is a black L-shaped axis with the text "ABC (orig coord system)". A second system, the translated system, is a red L-shaped axis with the text "ABC (translated coord system)". The red axis is shifted 50 units to the right and 50 units down from the origin of the black axis.

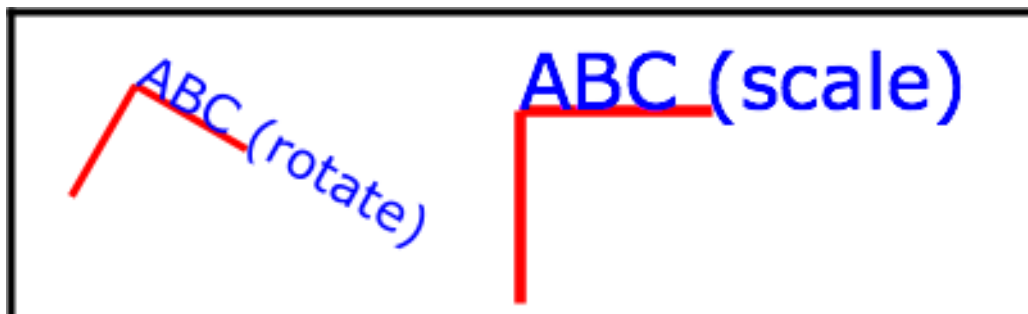
ABC (translated coord system)

Example 07_05 illustrates simple **rotate** and **scale** transformations. The example defines two new coordinate systems:

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

Example: 07_05.svg

```
<?xml version="1.0"?>
<svg width="400px" height="120px" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example RotateScale - Rotate and scale transforms</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="120" />
  </g>
  <!-- Establish a new coordinate system whose origin is at (50,30)
    in the initial coord. system and which is rotated by 30 degrees. -->
  <g transform="translate(50,30)">
    <g transform="rotate(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (rotate)
      </text>
    </g>
  </g>
  <!-- Establish a new coordinate system whose origin is at (200,40)
    in the initial coord. system and which is scaled by 1.5. -->
  <g transform="translate(200,40)">
    <g transform="scale(1.5)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (scale)
      </text>
    </g>
  </g>
</svg>
```



Example 07_06 defines two coordinate systems which are **skewed** relative to the origin coordinate system.

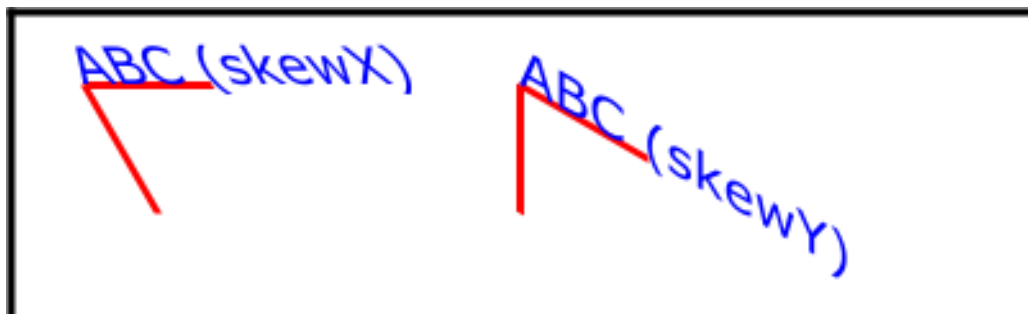
Example: 07_06.svg

```
<?xml version="1.0"?>
<svg width="400px" height="120px" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example Skew - Show effects of skewX and skewY</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="120" />
  </g>
  <!-- Establish a new coordinate system whose origin is at (30,30)
    in the initial coord. system and which is skewed in X by 30 degrees. -->
```

```

<g transform="translate(30,30)">
  <g transform="skewX(30)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
      ABC (skewX)
    </text>
  </g>
</g>
<!-- Establish a new coordinate system whose origin is at (200,30)
      in the initial coord. system and which is skewed in Y by 30 degrees. -->
<g transform="translate(200,30)">
  <g transform="skewY(30)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
      ABC (skewY)
    </text>
  </g>
</g>
</svg>

```



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

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

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

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

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

Simple transformations are represented in matrix form as follows:

- Translation is equivalent to the matrix

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

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

- Scaling is equivalent to the matrix

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

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

- Rotation about the origin is equivalent to the matrix

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or $[\cos(a) \ \sin(a) \ -\sin(a) \ \cos(a) \ 0 \ 0]$, which has the effect of rotating the coordinate system axes by angle a .

- A skew transformation along the x -axis is equivalent to the matrix

$$\begin{bmatrix} 1 & \tan(a) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or $[1 \ 0 \ \tan(a) \ 1 \ 0 \ 0]$, which has the effect of skewing X coordinates by angle a .

- A skew transformation along the y -axis is equivalent to the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ \tan(a) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or $[1 \ \tan(a) \ 0 \ 1 \ 0 \ 0]$, which has the effect of skewing Y coordinates by angle a .

7.5 Nested transformations

Transformations can be nested to any level. The effect of nested transformations is to post-multiply (i.e., concatenate) the

subsequent transformation matrices onto previously defined transformations:

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

For each given element, the accumulation of all transformations that have been defined on the given element and all of its ancestors up to and including the element that established the current **viewport** (usually, the **'svg'** element which is the most immediate ancestor to the given element) is called the **current transformation matrix** or **CTM**. The CTM thus represents the mapping of current user coordinates to viewport coordinates:

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

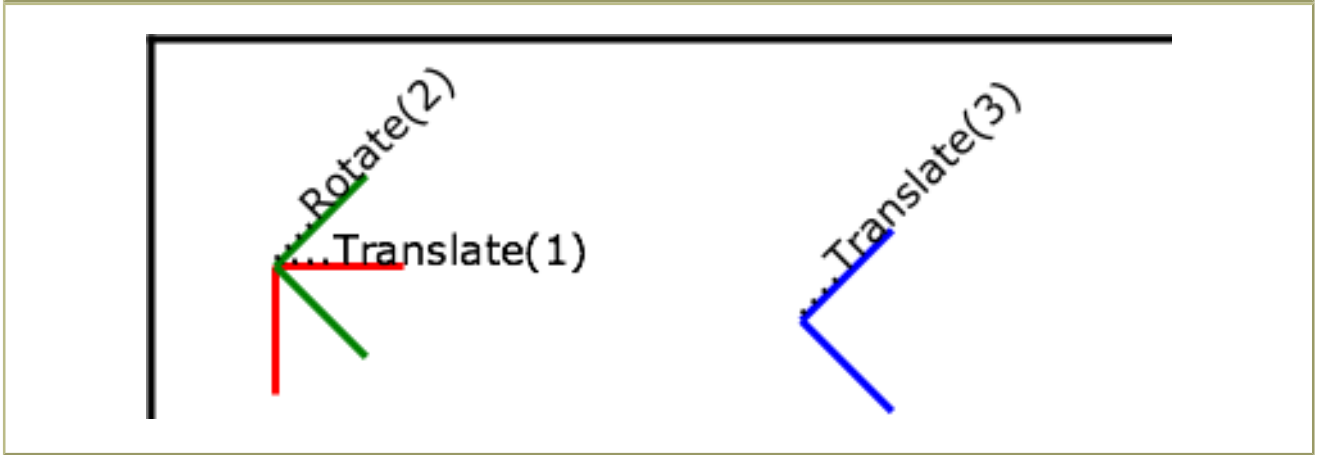
$$\begin{bmatrix} X_{\text{viewport}} \\ Y_{\text{viewport}} \\ 1 \end{bmatrix} = \text{CTM} \cdot \begin{bmatrix} X_{\text{userspace}} \\ Y_{\text{userspace}} \\ 1 \end{bmatrix}$$

Example 07_07 illustrates nested transformations.

Example: 07_07.svg

```
<?xml version="1.0"?>
<svg width="400px" height="150px" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example Nested - Nested transformations</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <!-- First, a translate -->
  <g transform="translate(50,90)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="16" font-family="Verdana" >
      ...Translate(1)
    </text>
    <!-- Second, a rotate -->
    <g transform="rotate(-45)">
      <g fill="none" stroke="green" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="16" font-family="Verdana" >
        ...Rotate(2)
      </text>
    <!-- Third, another translate -->
    <g transform="translate(130,160)">
      <g fill="none" stroke="blue" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="16" font-family="Verdana" >
        ...Translate(3)
      </text>
    </g>
  </g>
</svg>
```

</g>
</svg>



In the example above, the CTM within the third nested transformation (i.e., the `transform="translate(130,160)"`) consists of the concatenation of the three transformations, as follows:

$$\begin{aligned} \text{CTM} &= \text{translate}(50,90), \text{rotate}(-45), \text{translate}(130,160) \\ &= \begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 90 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} .707 & .707 & 0 \\ -.707 & .707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 130 \\ 0 & 1 & 160 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} .707 & .707 & 255.03 \\ -.707 & .707 & 111.21 \\ 0 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} X_{\text{initial}} \\ Y_{\text{initial}} \\ 1 \end{bmatrix} &= \text{CTM} \cdot \begin{bmatrix} X_{\text{userspace}} \\ Y_{\text{userspace}} \\ 1 \end{bmatrix} \end{aligned}$$

7.6 The transform attribute

`transform = <transform-list> | ref(svg [, <x>, <y>])`

For the `transform` attribute:

Animatable: yes.

See the ['animateTransform'](#) element for information on animating transformations.

The `<transform-list>` attribute type is defined below. The `'ref()'` attribute type is defined in the [Constrained Transformations](#) section.

7.6.1 The TransformList value

A `<transform-list>` is defined as a list of transform definitions, which are applied in the order provided. The individual transform

definitions are separated by whitespace and/or a comma. The available types of transform definitions include:

- **matrix(<a> <c> <d> <e> <f>)**, which specifies a transformation in the form of a [transformation matrix](#) of six values. **matrix(a,b,c,d,e,f)** is equivalent to applying the transformation matrix **[a b c d e f]**.
- **translate(<tx> [<ty>])**, which specifies a [translation](#) by *tx* and *ty*. If *<ty>* is not provided, it is assumed to be zero.
- **scale(<sx> [<sy>])**, which specifies a [scale](#) operation by *sx* and *sy*. If *<sy>* is not provided, it is assumed to be equal to *<sx>*.
- **rotate(<rotate-angle> [<cx> <cy>])**, which specifies a [rotation](#) by *<rotate-angle>* degrees about a given point.

If optional parameters *<cx>* and *<cy>* are not supplied, the rotate is about the origin of the current user coordinate system. The operation corresponds to the matrix **[cos(a) sin(a) -sin(a) cos(a) 0 0]**.

If optional parameters *<cx>* and *<cy>* are supplied, the rotate is about the point (*<cx>*, *<cy>*). The operation represents the equivalent of the following specification: **translate(<cx>, <cy>) rotate(<rotate-angle>) translate(-<cx>, -<cy>)**.

- **skewX(<skew-angle>)**, which specifies a [skew transformation along the x-axis](#).
- **skewY(<skew-angle>)**, which specifies a [skew transformation along the y-axis](#).

All numeric values are real [<number>](#)s.

A matrix with all values = 0 (**matrix(0,0,0,0,0,0)**) disables the rendering of the element, it does not create an error.

If a list of transforms is provided, then the net effect is as if each transform had been specified separately in the order provided. For example,

Example: 07_08.svg

```
<g transform="translate(-10,-20) scale(2) rotate(45) translate(5,10)">
  <!-- graphics elements go here -->
</g>
```

is functionally equivalent to:

Example: 07_09.svg

```
<g transform="translate(-10,-20)">
  <g transform="scale(2)">
    <g transform="rotate(45)">
      <g transform="translate(5,10)">
        <!-- graphics elements go here -->
      </g>
    </g>
  </g>
</g>
```

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

Example: 07_10.svg

```
<rect x="10" y="10" width="20" height="20" transform="scale(2)"/>
```

the *x*, *y*, *width* and *height* values are processed after the current coordinate system has been scaled uniformly by a factor of 2 by the **transform** attribute. Attributes *x*, *y*, *width* and *height* (and any other attributes or properties) are treated as values in the new user coordinate system, not the previous user coordinate system. Thus, the above **'rect'** element is functionally equivalent to:

Example: 07_11.svg

```
<g transform="scale(2)">
  <rect x="10" y="10" width="20" height="20"/>
</g>
```

The following is the Backus-Naur Form (BNF) for values for <transform-list>. The following notation is used:

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

```
transform-list:
  wsp* transforms? wsp*
transforms:
  transform
  | transform comma-wsp+ transforms
transform:
  matrix
  | translate
  | scale
  | rotate
  | skewX
  | skewY
matrix:
  "matrix" wsp* "(" wsp*
    number comma-wsp
    number comma-wsp
    number comma-wsp
    number comma-wsp
    number comma-wsp
    number wsp* ")"
translate:
  "translate" wsp* "(" wsp* number ( comma-wsp number )? wsp* ")"
scale:
  "scale" wsp* "(" wsp* number ( comma-wsp number )? wsp* ")"
rotate:
  "rotate" wsp* "(" wsp* number ( comma-wsp number comma-wsp number )? wsp* ")"
skewX:
  "skewX" wsp* "(" wsp* number wsp* ")"
skewY:
  "skewY" wsp* "(" wsp* number wsp* ")"
number:
  sign? integer-constant
  | sign? floating-point-constant
comma-wsp:
  (wsp+ comma? wsp*) | (comma wsp*)
comma:
  ","
integer-constant:
  digit-sequence
floating-point-constant:
  fractional-constant exponent?
  | digit-sequence exponent
fractional-constant:
  digit-sequence? "." digit-sequence
  | digit-sequence "."
exponent:
  ( "e" | "E" ) sign? digit-sequence
sign:
  "+" | "-"
digit-sequence:
  digit
  | digit digit-sequence
digit:
  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
wsp:
  (#x20 | #x9 | #xD | #xA)
```

7.7 Constrained Transformations

SVG 1.2 extends the coordinate system transformations allowed on groups and graphical elements to provide a method by

which graphical objects can remain fixed in the [viewport](#) without being scaled or rotated. Use cases include thin lines that do not become fatter on zooming in, map symbols or icons of a constant size, and so forth.

The following summarizes the different transforms that are applied to a graphical object as it is rendered.

7.7.1 The User Transform

The User Transform is the transformation that the user agent positioning controls apply to the viewport coordinate system. This transform can be considered to be applied to a group that surrounds the svg element of the document.

The user agent positioning controls consist of a translation (commonly referred to as the "pan"), a scale (commonly referred to as the "zoom") and a rotate.

```
US = User Scale (currentScale on SVGSVGElement)
UP = User Pan (currentTranslate on SVGSVGElement)
UR = User Rotate (currentRotate on SVGSVGElement)
```

The User Transform is the product of these component transformations.

```
U = User Transform
  = UP.US.UR
```

7.7.2 ViewBox to Viewport transformation

SVG elements, such as the [rootmost svg element](#), create their own [viewport](#). The [viewBox](#) to [viewport](#) transformation is the transformation on an svg element that adjusts the coordinate system to take the [viewBox](#) and preserveAspectRatio attributes into account.

We use the following notation for a [viewBox](#) to [viewport](#) transformation:

```
VB(svgId)
```

The 'svgId' parameter is the value of the id or xml:id attribute on a given svg element.

7.7.3 Element Transform Stack

All elements in an SVG document have a transform stack. This is the list of transforms that manipulate the coordinate system between the element and its nearest ancestor svg element, i.e. in this specification, the root element.

We use the following notation for the Element Transform stack on a given element:

```
TS(id)
```

The 'id' parameter is the value of the id or xml:id attribute on a given element.

Similarly, we use the following notation for the the transform defined by the transform attribute on the given element with identifier 'elt'.

```
Txf(id)
```

With the above definition, the transformation TS of an element is equal to the product of all the transformations Txf from that element to its nearest ancestor svg.

```
TS(id) = Txf(id.nearestViewportElement).[...].Txf(id.parentElement).Txf(id)
```

Example: Element transform stack

```

<svg id="root" version="1.2" baseProfile="tiny">
  <g id="g" transform="scale(2)">
    <rect id="r" transform="scale(4)"/>
    <g id="g2">
      <rect id="r2" transform="scale(0.5)" />
    </g>
  </g>
</svg>

```

In this example, the transforms are:

```

TS(g) = scale(2)
TS(r) = TS(g) . scale(4) = scale(8)
TS(g2) = TS(g) . I = scale(4) (I is the identity matrix)
TS(r2) = TS(g) . scale(0.5) = scale(1)

```

7.7.4 The Current Transform Matrix

Each element in the [rendering tree](#) has the concept of a Current Transform Matrix, or CTM. This is the product of all coordinate system transformations that apply to an element, effectively mapping the element into a coordinate system that is then transformed into device units by the user agent.

Consider the following example, with a rectangle having a set of ancestor g elements with ids "g-0" to "g-n".

Example: Current transform matrix

```

<svg id="root" version="1.2" baseProfile="tiny">
  ...
  <g id="g-n">
    ...
    <g id="g-2">
      ...
      <g id="g-1">
        ...
        <g id="g-0">
          ...
          <rect id="elt" .../>
        </g>
      </g>
    </g>
  </g>
</svg>

```

With the above definitions for [U](#), [VB](#), and [TS](#), the CTM for the rectangle with id "elt" is:

$$\begin{aligned}
 \text{CTM}(\text{elt}) &= \text{U.VB}(\text{root}).\text{TS}(\text{elt}) \\
 &= \text{U.VB}(\text{root}).\text{Txf}(\text{g-n}).[\dots].\text{Txf}(\text{g-0}).\text{Txf}(\text{elt})
 \end{aligned}$$

Example: n=2

```

<svg id="root" version="1.2" baseProfile="tiny">
  ...
  <g id="g-1">
    ...
    <g id="g-0">
      ...
      <rect id="elt" .../>
    </g>
  </g>
</svg>

```

This produces the following transformations:

$$\text{CTM}(\text{elt}) = \text{U.VB}(\text{root}).\text{Txf}(\text{g-1}).\text{Txf}(\text{g-0}).\text{Txf}(\text{elt})$$

Note the important relationship between an element's CTM and its parent CTM, for elements which do not define a viewport:

```
CTM(elt) = CTM(elt.parentElement).Txf(elt)
```

7.7.5 The ref() transform value

By using the "ref()" attribute value on the [transform attribute](#) it is possible to provide simple constrained transformations.

The 'ref(svg, x, y)' transform evaluates to the inverse of the element's parent's CTM multiplied by the svg element's CTM but exclusive of the svg element's zoom/pan/rotate user transform, if any.

The x and y parameters are optional. If they are specified an additional translation is appended to the transform so that (0, 0) in the element's user space maps to (x, y) in the svg element's user space. If no x and y parameters are specified, no additional translation is applied.

Using the definitions provided above:

```
Inverse of the parent's CTM: inv(CTM(elt.parentElement))

The svg element's user transform, exclusive of zoom,
pan and rotate transforms:
  CTM(svg[0].parentElement).VB(svg[0])

CTM(svg[0].parentElement) evaluates to Identity since there
is no svg[0].parentElement element.
```

In addition, the T(x, y) translation is such that:

```
CTM(elt).(0, 0) = CTM(svg[0]).(x, y)
```

So the transform evaluates to:

```
Txf(elt) = inv(CTM(elt.parentElement)).CTM(svg[0].parentElement).VB(svg[0]).T(x, y)
```

The element's CTM is:

```
CTM(elt) = CTM(elt.parentElement).Txf(elt)
          = CTM(svg[0].parentElement).VB(svg[0]).T(x,y)
```

Example: ref() transform

A small rectangle initially marks the middle of a line. The user agent [viewport](#) is a square with sides of 200 units.

```
<svg id="root" viewBox="0 0 100 100" version="1.2" baseProfile="tiny">
<line x1="0" x2="100" y1="0" y2="100"/>
<rect id="r" transform="ref(svg)"
x="45" y="45" width="10" height="10"/>
</svg>
```

In this case:

```
Txf(r) = inv(CTM(r.parent)).CTM(root.parentElement).VB(root).T(x, y)
CTM(root.parentElement) evaluates to Identity.
T(x, y) evaluates to Identity because (x, y) is not specified
CTM(r) = CTM(r.parent).Txf(r)
        = CTM(r.parent).inv(CTM(r.parent)).VB(root)
        = VB(root)
        = scale(2)
```

Consequently, regardless of the user transform (currentTranslate, currentScale, currentRotate) the rectangle's coordinates in viewport space will *always* be: (45, 45, 10, 10)*scale(2) = (90, 90, 20, 20). Initially, the line is from (0, 0) to (200, 200) in the [viewport](#) coordinate system. If we apply a user agent zoom of 3 (currentScale = 3), the rectangle is still (90, 90, 20, 20) but the line is (0, 0, 600, 600) and the marker no longer marks the middle of the line.

Example: ref() transform

A small rectangle always marks the middle of a line. Again, the user agent [viewport](#) is a square with sides of 200 units.

```
<svg id="root" baseProfile="tiny" viewBox="0 0 100 100" version="1.2">
<line x1="0" x2="100" y1="0" y2="100"/>
<g id="g" transform="ref(svg, 50, 50)">
<rect id="r" x="-5" y="-5" width="10" height="10"/>
</g>
</svg>
```

In this case:

```
Txf(g) = inv(CTM(g.parent)).CTM(root.parentElement).VB(root).T(x,y)
CTM(root.parentElement) evaluates to Identity.
CTM(g) = CTM(g.parent).Txf(r)
= CTM(g.parent).inv(CTM(g.parent)).VB(root).T(x,y)
= VB(root).T(x,y)
= scale(2).T(x,y)
```

Initially, (50, 50) in the svg user space is (100, 100) in [viewport](#) space. Therefore:

```
CTM(g).[0, 0] = CTM(root).[50, 50]
= scale(2).[50, 50]
= [100, 100]
and
scale(2).T(x,y) = [100, 100]
T(x,y) = translate(50, 50)
```

If the user agent pan was (50, 80) (modifying currentTranslate) then we now have (50, 50) in the svg element's user space located at (150, 180) in the [viewport](#) space. This produces:

```
CTM(g).[0, 0] = CTM(root).[50, 50]
= translate(50, 80).scale(2).[50, 50]
= [150, 180]
and
scale(2).T(x,y) = [150, 180]
T(x, y) = translate(75, 90)
```

Therefore, regardless of the user transform, the rectangle will always overlap the middle of the line. Note that the rectangle will not rotate with the line (e.g., if currentRotate is set) and it will not scale either.

7.8 The viewBox attribute

It is often desirable to specify that a given set of graphics stretch to fit a particular container element. The [viewBox](#) attribute provides this capability. All elements that establish a new [viewport](#) (see [elements that establish viewports](#)) have attribute [viewBox](#).

Attribute definition:

viewBox = "<list> | 'none'
<list>

A list of four numbers <min-x>, <min-y>, <width> and <height>, separated by whitespace and/or a comma, which specify a rectangle in user space which should be mapped to the bounds of the viewport established by the given element, taking into account attribute [preserveAspectRatio](#). If specified, an additional transformation is applied to all descendants of the given element to achieve the specified effect.

'none'

Specifying a value of 'none' indicates that no [viewBox](#) should be used. This is exactly the same as not setting the [viewBox](#) at all.

[Animatable](#): yes.

A negative value for <width> or <height> is unsupported. A value of zero disables rendering of the element.

Example 07_12 illustrates the use of the [viewBox](#) attribute on the `'svg'` element to specify that the SVG content should stretch to fit bounds of the [viewport](#).

Example: 07_12.svg

```
<?xml version="1.0"?>
<svg width="300px" height="200px" version="1.2" baseProfile="tiny"
  viewBox="0 0 1500 1000" preserveAspectRatio="none"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Example viewBox - uses the viewBox
  attribute to automatically create an initial user coordinate
  system which causes the graphic to scale to fit into the
  viewport no matter what size the viewport is.</desc>
  <!-- This rectangle goes from (0,0) to (1500,1000) in user space.
  Because of the viewBox attribute above,
  the rectangle will end up filling the entire area
  reserved for the SVG content. -->
  <rect x="0" y="0" width="1500" height="1000"
    fill="yellow" stroke="blue" stroke-width="12" />
  <!-- A large, red triangle -->
  <path fill="red" d="M 750,100 L 250,900 L 1250,900 z"/>
  <!-- A text string that spans most of the viewport -->
  <text x="100" y="600" font-size="200" font-family="Verdana" >
    Stretch to fit
  </text>
</svg>
```



Rendered into [viewport](#) with
width=300px, height=200px



width=150px,
height=200px

The effect of the [viewBox](#) attribute is that the user agent automatically supplies the appropriate transformation matrix to map the specified rectangle in user space to the bounds of a designated region (often, the viewport). To achieve the effect of the example on the left, with [viewport](#) dimensions of 300 by 200 pixels, the user agent needs to automatically insert a transformation which scales both X and Y by 0.2. The effect is equivalent to having a [viewport](#) of size 300px by 200px and the following supplemental transformation in the document, as follows:

```
<svg version="1.2" baseProfile="tiny" width="300px" height="200px">
  <g transform="scale(0.2)">
    <!-- Rest of document goes here -->
  </g>
</svg>
```

To achieve the effect of the example on the right, with [viewport](#) dimensions of 150 by 200 pixels, the user agent needs to automatically insert a transformation which scales X by 0.1 and Y by 0.2. The effect is equivalent to having a [viewport](#) of size 150px by 200px and the following supplemental transformation in the document, as follows:

```
<svg version="1.2" baseProfile="tiny" width="150px" height="200px">
  <g transform="scale(0.1 0.2)">
    <!-- Rest of document goes here -->
  </g>
</svg>
```

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

Unlike the **transform** attribute (see [effect of the transform on sibling attributes](#)), the automatic transformation that is created due to a **viewBox** does not affect the **x**, **y**, **width** and **height** attributes on the element with the **viewBox** attribute. Thus, in the example above which shows an **'svg'** element which has attributes **width**, **height** and **viewBox**, the **width** and **height** attributes represent values in the coordinate system that exists *before* the **viewBox** transformation is applied. On the other hand, like the **transform** attribute, it does establish a new coordinate system for all other attributes and for descendant elements.

7.9 The **preserveAspectRatio** attribute

In some cases, typically when using the **viewBox** attribute, it is desirable that the graphics stretch to fit non-uniformly to take up the entire **viewport**. In other cases, it is desirable that uniform scaling be used for the purposes of preserving the aspect ratio of the graphics.

Attribute **preserveAspectRatio="[defer] <align> [<meet>]"**, which is available for all elements that establish a new **viewport** (see [elements that establish viewports](#)), indicates whether or not to force uniform scaling.

preserveAspectRatio only applies when a value has been provided for **viewBox** on the same element. Or, in some cases, if an implicit **viewBox** value can be established for the element (see each element description for details on this). If a **viewBox** value can not be determined then **preserveAspectRatio** is ignored.

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

The **<align>** parameter indicates whether to force uniform scaling and, if so, the alignment method to use in case the aspect ratio of the **viewBox** doesn't match the aspect ratio of the **viewport**. The **<align>** parameter must be one of the following strings:

- **none** - Do not force uniform scaling. Scale the graphic content of the given element non-uniformly if necessary such that the element's bounding box exactly matches the **viewport** rectangle.
- **xMinYMin** - Force uniform scaling.
Align the **<min-x>** of the element's **viewBox** with the smallest X value of the **viewport**.
Align the **<min-y>** of the element's **viewBox** with the smallest Y value of the **viewport**.
- **xMidYMin** - Force uniform scaling.
Align the midpoint X value of the element's **viewBox** with the midpoint X value of the **viewport**.
Align the **<min-y>** of the element's **viewBox** with the smallest Y value of the **viewport**.
- **xMaxYMin** - Force uniform scaling.
Align the **<min-x>+<width>** of the element's **viewBox** with the maximum X value of the **viewport**.
Align the **<min-y>** of the element's **viewBox** with the smallest Y value of the **viewport**.
- **xMinYMid** - Force uniform scaling.
Align the **<min-x>** of the element's **viewBox** with the smallest X value of the **viewport**.
Align the midpoint Y value of the element's **viewBox** with the midpoint Y value of the **viewport**.
- **xMidYMid** (the default) - Force uniform scaling.
Align the midpoint X value of the element's **viewBox** with the midpoint X value of the **viewport**.
Align the midpoint Y value of the element's **viewBox** with the midpoint Y value of the **viewport**.

- **xMaxYMid** - Force uniform scaling.

Align the `<min-x>+<width>` of the element's [viewBox](#) with the maximum X value of the [viewport](#).

Align the midpoint Y value of the element's [viewBox](#) with the midpoint Y value of the [viewport](#).

- **xMinYMax** - Force uniform scaling.

Align the `<min-x>` of the element's [viewBox](#) with the smallest X value of the [viewport](#).

Align the `<min-y>+<height>` of the element's [viewBox](#) with the maximum Y value of the [viewport](#).

- **xMidYMax** - Force uniform scaling.

Align the midpoint X value of the element's [viewBox](#) with the midpoint X value of the [viewport](#).

Align the `<min-y>+<height>` of the element's [viewBox](#) with the maximum Y value of the [viewport](#).

- **xMaxYMax** - Force uniform scaling.

Align the `<min-x>+<width>` of the element's [viewBox](#) with the maximum X value of the [viewport](#).

Align the `<min-y>+<height>` of the element's [viewBox](#) with the maximum Y value of the [viewport](#).

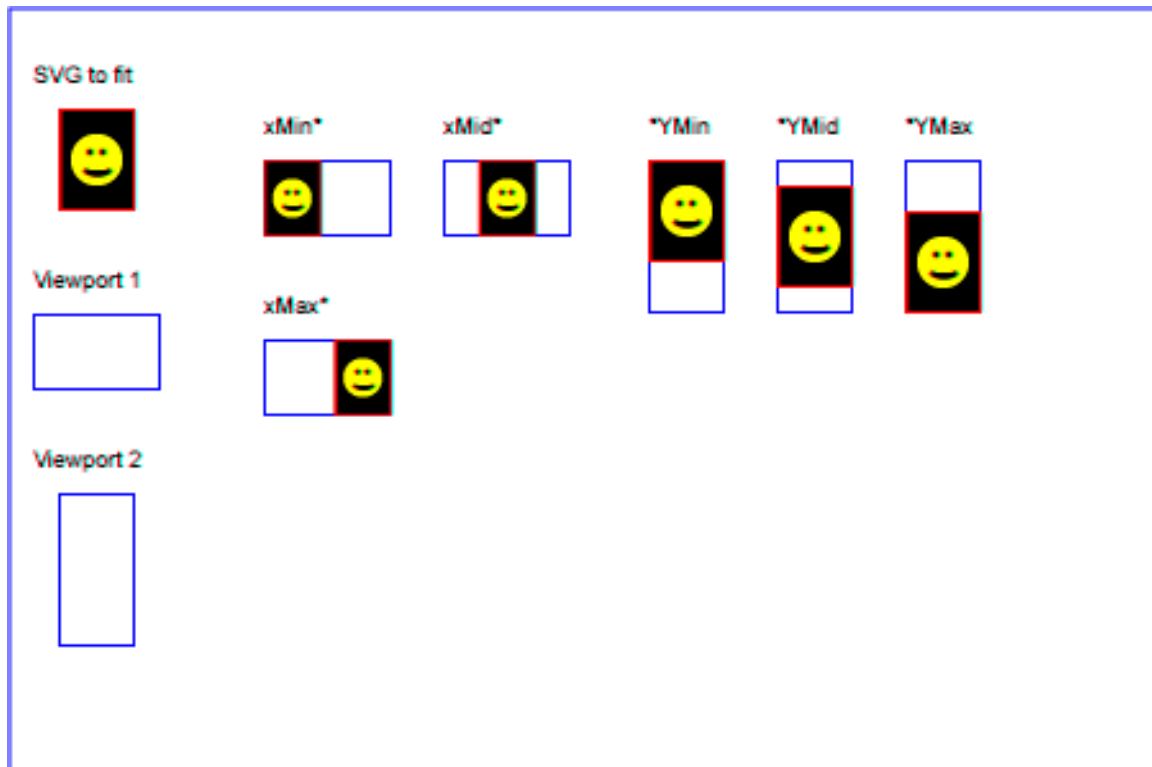
The `<meet>` parameter is optional and is only available due to historical reasons. The `<meet>` is separated from the `<align>` value by one or more spaces and must equal the string `meet`. This is also the default value and therefore it is recommended that content do not specify this parameter since it adds no additional value.

`meet` indicates to scale the graphic such that:

- aspect ratio is preserved
- the entire [viewBox](#) is visible within the viewport
- the [viewBox](#) is scaled up as much as possible, while still meeting the other criteria

In this case, if the aspect ratio of the graphic does not match the viewport, some of the [viewport](#) will extend beyond the bounds of the [viewBox](#) (i.e., the area into which the [viewBox](#) will draw will be smaller than the viewport).

Example `PreserveAspectRatio` illustrates the various options on `preserveAspectRatio`. The example creates several new viewports by including `'animation'` elements (see [Establishing a new viewport](#)).



Example `PreserveAspectRatio`

For the `preserveAspectRatio` attribute:

[Animatable](#): yes.

7.10 Establishing a new viewport

Some elements establish a new [viewport](#). By establishing a new viewport, you implicitly establish a new [viewport](#) coordinate system and a new user coordinate system. Additionally, there is a new meaning for percentage units defined to be relative to the current [viewport](#) since a new [viewport](#) has been established (see [Units](#))

['viewport-fill'](#) and ['viewport-fill-opacity'](#) properties can be applied on the new [viewport](#).

The bounds of the new [viewport](#) are defined by the `x`, `y`, `width` and `height` attributes on the element establishing the new viewport, such as an ['animation'](#) element. Both the new [viewport](#) coordinate system and the new user coordinate system have their origins at `(x, y)`, where `x` and `y` represent the value of the corresponding attributes on the element establishing the [viewport](#). The orientation of the new [viewport](#) coordinate system and the new user coordinate system correspond to the orientation of the current user coordinate system for the element establishing the [viewport](#). A single unit in the new [viewport](#) coordinate system and the new user coordinate system are the same size as a single unit in the current user coordinate system for the element establishing the [viewport](#).

For an extensive example of creating new viewports, see [Example PreserveAspectRatio](#).

The following elements establish new viewports:

- The ['svg'](#) element establishes the root [viewport](#) for the document.
- The ['animation'](#) element.
- The ['image'](#) element.
- The ['video'](#) element.

7.11 Units

Besides the exceptions listed below all coordinates and lengths in SVG must be specified in user units, which means that **unit identifiers** are not allowed.

Two exceptions exist:

- **unit identifiers** are allowed on the `'width'` and `'height'` attributes of the `'svg'` element.
- [Object bounding box units](#) are allowed on `'linearGradient'` and `'radialGradient'` elements.

A user unit is a value in the current user coordinate system. For example:

Example: 07_17.svg

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

For the `svg` element's `'width'` and `'height'` attributes a coordinate or length value can be expressed as a number following by a unit identifier (e.g., "25cm" or "100%"). The list of unit identifiers in SVG are the following CSS unit identifiers: in, cm, mm, pt, pc, px and percentages (%). The following describes how the various unit identifiers are processed:

- One `px` unit is defined to be equal to one user unit. Thus, a length of "5px" is the same as a length of "5".
- The other absolute unit identifiers from CSS (i.e., pt, pc, cm, mm, in) are all defined as an appropriate multiple of one `px` unit (which, according to the previous item, is defined to be equal to one user unit), based on what the SVG user agent determines is the size of a `px` unit (possibly passed from the parent processor or environment at initialization time). For example, suppose that the user agent can determine from its environment that "1px" corresponds to "0.282222mm" (i.e., 90dpi). Then, for all processing:

- "1pt" equals "1.25px" (and therefore 1.25 user units)
- "1pc" equals "15px" (and therefore 15 user units)
- "1mm" would be "3.543307px" (3.543307 user units)
- "1cm" equals "35.43307px" (and therefore 35.43307 user units)
- "1in" equals "90px" (and therefore 90 user units)

Percentage values on 'width' and 'height' attributes mandates how much space the SVG [viewport](#) should take of the available initial [viewport](#). See list below and [initial viewport](#) for details.

- For any width value expressed as a percentage of the [viewport](#), the value to use is the specified percentage of the *actual-width* in user units for the nearest containing [viewport](#), where *actual-width* is the width dimension of the [viewport](#) element within the user coordinate system for the [viewport](#) element.
- For any height value expressed as a percentage of the [viewport](#), the value to use is the specified percentage of the *actual-height* in user units for the nearest containing [viewport](#), where *actual-height* is the height dimension of the [viewport](#) element within the user coordinate system for the [viewport](#) element.

7.12 Object bounding box units

The following elements offer the option of expressing coordinate values and lengths as fractions of the **bounding box** (via keyword `objectBoundingBox`) on a given element:

Element	Attribute	Effect
linearGradient	<code>gradientUnits="objectBoundingBox"</code>	Indicates that the attributes which specify the gradient vector (x1 , y1 , x2 , y2) represent fractions of the bounding box of the element to which the gradient is applied.
radialGradient	<code>gradientUnits="objectBoundingBox"</code>	Indicates that the attributes which specify the center (cx , cy) and the radius (r) represent fractions of the bounding box of the element to which the gradient is applied.

In the discussion that follows, the term **applicable element** is the element to which the given effect applies. For gradients the applicable element is the [graphics element](#) which has its `'fill'` or `'stroke'` property referencing the given gradient. (See [Inheritance of Painting Properties](#). For special rules concerning [text elements](#), see the discussion of [object bounding box units and text elements](#).)

When keyword `objectBoundingBox` is used, then the effect is as if a supplemental transformation matrix were inserted into the list of nested transformation matrices to create a new user coordinate system.

First, the (`minx,miny`) and (`maxx,maxy`) coordinates are determined for the applicable element and all of its descendants. The values `minx`, `miny`, `maxx` and `maxy` are determined by computing the maximum extent of the shape of the element in X and Y with respect to the user coordinate system for the applicable element. The bounding box is the tightest fitting rectangle aligned with the axes of the applicable element's user coordinate system that entirely encloses the applicable element and its descendants. The bounding box is computed exclusive of any values for opacity and stroke-width. For curved shapes, the bounding box encloses all portions of the shape, not just end points. For `'text'` elements, for the purposes of the bounding box calculation, each glyph is treated as a separate graphics element. The calculations assume that all glyphs occupy the full glyph cell. For example, for horizontal text, the calculations assume that each glyph extends vertically to the full ascent and descent values for the font.

Then, coordinate (0,0) in the new user coordinate system is mapped to the (minx,miny) corner of the tight bounding box within the user coordinate system of the applicable element and coordinate (1,1) in the new user coordinate system is mapped to the (maxx,maxy) corner of the tight bounding box of the applicable element. In most situations, the following transformation matrix produces the correct effect:

```
[ (maxx-minx) 0 0 (maxy-miny) minx miny ]
```

Any numeric value can be specified for values expressed as a fraction of object bounding box units. In particular, fractions less than zero or greater than one can be specified.

Keyword **objectBoundingBox** should not be used when the geometry of the applicable element has no width or no height, such as the case of a horizontal or vertical line, even when the line has actual thickness when viewed due to having a non-zero stroke width since stroke width is ignored for bounding box calculations. When the geometry of the applicable element has no width or height and **objectBoundingBox** is specified, then the given effect (e.g., a gradient) will be ignored.

7.13 Intrinsic Sizing Properties of the Viewport of SVG content

SVG needs to specify how to calculate some intrinsic sizing properties to enable inclusion within other languages. The intrinsic width and height of the **viewport** of SVG content must be determined from the width and height attributes. If these are not specified, the default values of 100% must be used.

The intrinsic aspect ratio of the **viewport** of SVG content is necessary for example, when including SVG from an object element in XHTML styled with CSS. The intrinsic aspect ratio must be calculated based upon the following rules:

- If the width and height of the [rootmost svg element](#) are both specified in absolute units (in, mm, cm, pt, pc, px, user units) then the aspect ratio is calculated from the width and height after resolving both values to user units.
- If either/both of the width and height of the [rootmost svg element](#) are in percentage units, the aspect ratio is calculated from the width and height of the [viewBox](#) specified for the current SVG document fragment. If the [viewBox](#) is not correctly specified, or set to 'none', the intrinsic aspect ratio cannot be calculated and is considered unspecified.

Examples:

Example: Intrinsic Aspect Ratio 1

```
<svg version="1.2" baseProfile="tiny" width="10cm" height="5cm">
...
</svg>
```

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

Example: Intrinsic Aspect Ratio 2

```
<svg version="1.2" baseProfile="tiny" width="100%" height="50%" viewBox="0 0 200 200">
...
</svg>
```

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

Example: Intrinsic Aspect Ratio 3

```
<svg version="1.2" baseProfile="tiny" width="10cm" viewBox="0 0 200 200">
...
</svg>
```

In this case the intrinsic aspect ratio is 1:1.

Example: Intrinsic Aspect Ratio 4

```
<svg version="1.2" baseProfile="tiny" width="75%" height="10cm" viewBox="0 0 200 200">
...
</svg>
```

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

7.14 Geographic Coordinate Systems

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

Such metadata should be added under the **'metadata'** element of the topmost **'svg'** element describing the map. They consist of an RDF description of the Coordinate Reference System definition used to generate the SVG map. Note that the presence of this metadata does not affect the rendering of the SVG in any way; it merely provides added semantic value for applications that use of combine maps.

The definition should be conformant to the XML grammar described in the OpenGIS Recommendation on the Definition of Coordinate Reference System [[OpenGIS Coordinate Systems](#)]. In order to correctly map the 2-dimensional data used by SVG, the CRS must be of subtype ProjectedCRS or Geographic2dCRS. The first axis of the described CRS maps the SVG x-axis and the second axis maps the SVG y-axis. Optionally, an additional affine transformation may have been applied during this mapping. This additional transformation is described by an SVG **transform** attribute that can be added to the OpenGIS **'CoordinateReferenceSystem'** element. Note that the **transform** attribute on the **'CoordinateReferenceSystem'** does not indicate that a transformation should be applied to the data within the file, it simply describes the transformation that was applied to the data when being encoded in SVG.

There are three typical uses for the SVG transform attribute. These are described below and used in the examples.

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

The main purpose of such metadata is to indicate to the User Agent that two or more SVG documents can be overlaid or merged into a single document. Obviously, if two maps reference the same Coordinate Reference System definition and have the same SVG transform attribute value then they can be overlaid without reprojecting the data. If the maps reference different Coordinate Reference Systems and/or have different SVG transform attribute values, then a specialized cartographic User Agent may choose to transform the coordinate data to overlay the data. However, typical SVG user agents are not required to perform these types of transformations, or even recognize the metadata.

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

Example: 07_19.svg

```

<?xml version="1.0"?>
<svg width="100" height="100" viewBox="0 0 1000 1000" version="1.2"
  xmlns="http://www.w3.org/2000/svg" baseProfile="tiny">
  <desc>An example that references co-ordinate data.</desc>
  <metadata>
    <rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:crs = "http://www.ogc.org/crs"
      xmlns:svg="http://www.w3.org/2000/svg">
      <rdf:Description>
        <!-- The Coordinate Reference System is described
          through an URI. -->
        <crs:CoordinateReferenceSystem transform="rotate(-90)"
          rdf:resource="http://www.example.org/srs/epsg.xml#4326"/>
      </rdf:Description>
    </rdf:RDF>
  </metadata>
  <!-- The actual map content -->
</svg>

```

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

Example: 07_20.svg

```

<?xml version="1.0"?>
<svg width="100" height="100" viewBox="0 0 1000 1000" version="1.2"
  xmlns="http://www.w3.org/2000/svg" baseProfile="tiny">
  <desc>Example using a well known co-ordinate system.</desc>
  <metadata>
    <rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:crs = "http://www.ogc.org/crs"
      xmlns:svg="http://www.w3.org/2000/svg">
      <rdf:Description>
        <!-- In case of a well-known Coordinate Reference System
          an 'Identifier' is enough to describe the CRS -->
        <crs:CoordinateReferenceSystem transform="rotate(-90) scale(100, 100)">
          <crs:Identifier>
            <crs:code>4326</crs:code>
            <crs:codeSpace>EPSG</crs:codeSpace>
            <crs:edition>5.2</crs:edition>
          </crs:Identifier>
        </crs:CoordinateReferenceSystem>
      </rdf:Description>
    </rdf:RDF>
  </metadata>
  <!-- The actual map content -->
</svg>

```

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

Example: 07_21.svg

```

<?xml version="1.0"?>
<svg width="100" height="100" viewBox="0 0 1000 1000" version="1.2"
  xmlns="http://www.w3.org/2000/svg" baseProfile="tiny">
  <desc>Co-ordinate Metadata defined within the SVG Document</desc>
  <metadata>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:crs="http://www.ogc.org/crs"
      xmlns:svg="http://www.w3.org/2000/svg">
      <rdf:Description>
        <!-- For other CRS it should be entirely defined -->
        <crs:CoordinateReferenceSystem transform="scale(1,-1)">
          <crs:NameSet>
            <crs:name>Mercator projection of WGS84</crs:name>
          </crs:NameSet>
          <crs:ProjectedCRS>
            <!-- The actual definition of the CRS -->
            <crs:CartesianCoordinateSystem>
              <crs:dimension>2</crs:dimension>
              <crs:CoordinateAxis>
                <crs:axisDirection>north</crs:axisDirection>

```

```

    <crs:AngularUnit>
      <crs:Identifier>
        <crs:code>9108</crs:code>
        <crs:codeSpace>EPSG</crs:codeSpace>
        <crs:edition>5.2</crs:edition>
      </crs:Identifier>
    </crs:AngularUnit>
  </crs:CoordinateAxis>
<crs:CoordinateAxis>
  <crs:axisDirection>east</crs:axisDirection>
  <crs:AngularUnit>
    <crs:Identifier>
      <crs:code>9108</crs:code>
      <crs:codeSpace>EPSG</crs:codeSpace>
      <crs:edition>5.2</crs:edition>
    </crs:Identifier>
  </crs:AngularUnit>
</crs:CoordinateAxis>
</crs:CartesianCoordinateSystem>
<crs:CoordinateReferenceSystem>
  <!-- the reference system of that projected system is
        WGS84 which is EPSG 4326 in EPSG codeSpace -->
  <crs:NameSet>
    <crs:name>WGS 84</crs:name>
  </crs:NameSet>
  <crs:Identifier>
    <crs:code>4326</crs:code>
    <crs:codeSpace>EPSG</crs:codeSpace>
    <crs:edition>5.2</crs:edition>
  </crs:Identifier>
</crs:CoordinateReferenceSystem>
<crs:CoordinateTransformationDefinition>
  <crs:sourceDimensions>2</crs:sourceDimensions>
  <crs:targetDimensions>2</crs:targetDimensions>
  <crs:ParameterizedTransformation>
    <crs:TransformationMethod>
      <!-- the projection is a Mercator projection which is
            EPSG 9805 in EPSG codeSpace -->
      <crs:NameSet>
        <crs:name>Mercator</crs:name>
      </crs:NameSet>
      <crs:Identifier>
        <crs:code>9805</crs:code>
        <crs:codeSpace>EPSG</crs:codeSpace>
        <crs:edition>5.2</crs:edition>
      </crs:Identifier>
      <crs:description>Mercator (2SP)</crs:description>
    </crs:TransformationMethod>
    <crs:Parameter>
      <crs:NameSet>
        <crs:name>Latitude of 1st standart parallel</crs:name>
      </crs:NameSet>
      <crs:Identifier>
        <crs:code>8823</crs:code>
        <crs:codeSpace>EPSG</crs:codeSpace>
        <crs:edition>5.2</crs:edition>
      </crs:Identifier>
      <crs:value>0</crs:value>
    </crs:Parameter>
    <crs:Parameter>
      <crs:NameSet>
        <crs:name>Longitude of natural origin</crs:name>
      </crs:NameSet>
      <crs:Identifier>
        <crs:code>8802</crs:code>
        <crs:codeSpace>EPSG</crs:codeSpace>
        <crs:edition>5.2</crs:edition>
      </crs:Identifier>
      <crs:value>0</crs:value>
    </crs:Parameter>
    <crs:Parameter>
      <crs:NameSet>
        <crs:name>False Easting</crs:name>
      </crs:NameSet>
      <crs:Identifier>
        <crs:code>8806</crs:code>
        <crs:codeSpace>EPSG</crs:codeSpace>
        <crs:edition>5.2</crs:edition>
      </crs:Identifier>
      <crs:value>0</crs:value>
    </crs:Parameter>
    <crs:Parameter>
      <crs:NameSet>

```

```
        <crs:name>False Northing</crs:name>
      </crs:NameSet>
      <crs:Identifier>
        <crs:code>8807</crs:code>
        <crs:codeSpace>EPSG</crs:codeSpace>
        <crs:edition>5.2</crs:edition>
      </crs:Identifier>
      <crs:value>0</crs:value>
    </crs:Parameter>
  </crs:ParameterizedTransformation>
</crs:CoordinateTransformationDefinition>
</crs:ProjectedCRS>
</crs:CoordinateReferenceSystem>
</rdf:Description>
</rdf:RDF>
</metadata>
<!-- the actual map content -->
</svg>
```


8 Paths

Contents

- 8.1 [Introduction](#)
- 8.2 [The 'path' element](#)
 - 8.2.1 [Animating path data](#)
- 8.3 [Path data](#)
 - 8.3.1 [General information about path data](#)
 - 8.3.2 [The "moveto" commands](#)
 - 8.3.3 [The "closepath" command](#)
 - 8.3.4 [The "lineto" commands](#)
 - 8.3.5 [The Curve commands](#)
 - 8.3.6 [The Cubic Bézier curve commands](#)
 - 8.3.7 [The Quadratic Bézier curve commands](#)
 - 8.3.8 [The grammar for path data](#)
- 8.4 [Distance along a path](#)

8.1 Introduction

Paths represent the outline of a shape which can be filled or stroked. (See [Filling, Stroking and Paint Servers](#).)

A path is described using the concept of a current point. In an analogy with drawing on paper, the current point can be thought of as the location of the pen. The position of the pen can be changed, and the outline of a shape (open or closed) can be traced by dragging the pen in either straight lines or curves.

Paths represent the geometry of the outline of an object, defined in terms of *moveto* (set a new current point), *lineto* (draw a straight line), *curveto* (draw a curve using a cubic Bézier) and *closepath* (close the current shape by drawing a line to the last *moveto*) elements. Compound paths (i.e., a path with multiple subpaths) are possible to allow effects such as "donut holes" in objects.

This chapter describes the syntax and behavior for SVG paths. Various implementation notes for SVG paths can be found in ['path' element implementation notes](#).

A path is defined in SVG using the ['path'](#) element.

8.2 The 'path' element

Schema: path

```
<define name='path'>
  <element name='path'>
    <ref name='path.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='path.AT' combine='interleave'>
  <ref name='svg.ShapeCommon.attr' />
  <ref name='svg.D.attr' />
  <optional>
    <attribute name='pathLength' svg:animatable='true' svg:inheritable='false'>
      <ref name='Number.datatype' />
    </attribute>
  </optional>
</define>
```

Attribute definitions:

d = "path data"

The definition of the outline of a shape. See [Path data](#). An empty attribute value (d="") disables rendering of the element. If the attribute is not specified, the effect is as if an empty string ("") was specified.

Animatable: yes, but see restrictions described in [Animating path data](#)

pathLength = "<number>"

The authoring length of the path, in user units. This value is used to calibrate the user agent's own [distance-along-a-path](#) calculations with that of the author. The user agent shall scale all distance-along-a-path computations by the ratio of **pathLength** to the user agent's own computed value for total path length. **pathLength** potentially affects calculations for [motion animation](#) and various [stroke operations](#).

A negative value shall be treated as unsupported.

Animatable: yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

8.2.1 Animating path data

Interpolated path data animation is only possible when each [normalized](#) path data specification within an animation specification has exactly the same list of path data commands as the **d** attribute after [normalization](#). This means that each path data specification and the **d** attribute would have the exact same list of commands if normalized as defined in [Path Normalization](#). If an animation is specified and the list of path data commands is not the same, then the animation specification must be ignored as unsupported. The animation engine shall interpolate each parameter to each path data command separately based upon the attributes to the given animation element.

Non-interpolated (i.e. the [calcMode](#) is **discrete**) path data animation is always possible.

8.3 Path data

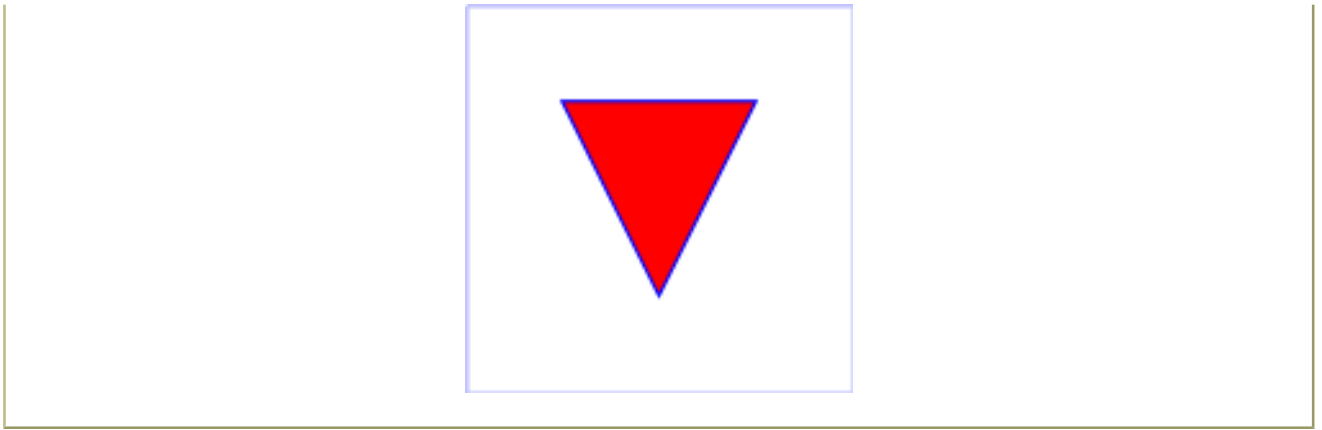
8.3.1 General information about path data

A path is defined by including a **'path'** element which contains a **d** attribute, where the **d** attribute contains the *moveto*, *line*, *curve* (both cubic and quadratic Béziers) and *closepath* instructions.

Example 08_01 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*).

Example: 08_01.svg

```
<?xml version="1.0"?>
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <title>Example triangle01- simple example of a 'path'</title>
  <desc>A path that draws a triangle</desc>
  <rect x="1" y="1" width="398" height="398"
    fill="none" stroke="blue" />
  <path d="M 100 100 L 300 100 L 200 300 z"
    fill="red" stroke="blue" stroke-width="3" />
</svg>
```



Path data can contain newline characters and thus can be broken up into multiple lines to improve readability.

The syntax of path data is concise in order to allow for minimal file size and efficient downloads, since many SVG files will be dominated by their path data. Some of the ways that SVG attempts to minimize the size of path data are as follows:

- All instructions are expressed as one character (e.g., a *moveto* is expressed as an **M**).
- Superfluous white space and separators such as commas can be eliminated (e.g., "M 100 100 L 200 200" contains unnecessary spaces and could be expressed more compactly as "M100 100L200 200").
- The command letter can be eliminated on subsequent commands if the same command is used multiple times in a row (e.g., you can drop the second "L" in "M 100 200 L 200 100 L -100 -200" and use "M 100 200 L 200 100 -100 -200" instead).
- Relative versions of all commands are available (uppercase means absolute coordinates, lowercase means relative coordinates).
- Alternate forms of *lineto* are available to optimize the special cases of horizontal and vertical lines (absolute and relative).
- Alternate forms of *curve* are available to optimize the special cases where some of the control points on the current segment can be determined automatically from the control points on the previous segment.

The path data syntax is a prefix notation (i.e., commands followed by parameters). The only allowable decimal point is a Unicode [\[UNICODE\]](#) FULL STOP (".") character (also referred to in Unicode as PERIOD, dot and decimal point) and no other delimiter characters are allowed. (For example, the following is an invalid numeric value in a path data stream: "13,000.56". Instead, say: "13000.56".)

For the relative versions of the commands, all coordinate values shall be relative to the current point at the start of the command.

In the tables below, the following notation is used:

- **()**: grouping of parameters
- **+**: 1 or more of the given parameter(s) is required
- Coordinates following commands in uppercase (e.g., **M**) shall be treated as absolute coordinates.
- Coordinates following commands in lowercase (e.g., **m**) shall be treated as relative coordinates.

The following sections list the commands.

8.3.2 The "moveto" commands

The '[moveto](#)' commands (**M** or **m**) establish a new current point. The effect is as if the "pen" were lifted and moved to a new location. A path data segment (if there is one) must begin with a '[moveto](#)' command. Subsequent '[moveto](#)' commands (i.e., when the '[moveto](#)' is not the first command) represent the start of a new *subpath*:

Command	Name	Parameters	Description

M (absolute) m (relative)	moveto	(x y)+	A new sub-path at the given (x,y) coordinate shall be started. This shall also establish a new current point at the given coordinate. If a relative 'moveto' (m) appears as the first element of the 'path' , then it shall be treated as a pair of absolute coordinates. If a 'moveto' is followed by multiple pairs of coordinates, the subsequent pairs shall be treated as implicit 'lineto' commands.
--	--------	--------	---

8.3.3 The "closepath" command

A straight line shall be drawn from the current point to the initial point of the current subpath, and shall end the current subpath. If a ['closepath'](#) (**Z** or **z**) is followed immediately by any other command, then the next subpath must start at the same initial point as the current subpath.

When a subpath ends in a ['closepath'](#), it differs in behavior from what happens when "manually" closing a subpath via a ['lineto'](#) command in how ['stroke-linejoin'](#) and ['stroke-linecap'](#) are implemented. With ['closepath'](#), the end of the final segment of the subpath shall be "joined" with the start of the initial segment of the subpath using the current value of ['stroke-linejoin'](#). If instead the subpath is closed "manually" via a ['lineto'](#) command, the start of the first segment and the end of the last segment are not joined but instead shall each be capped using the current value of ['stroke-linecap'](#). At the end of the command, the new current point shall be set to the initial point of the current subpath.

Command	Name	Parameters	Description
Z or z	closepath	(none)	The current subpath shall be closed by drawing a straight line from the current point to current subpath's initial point, which then shall become the new current point.

8.3.4 The "lineto" commands

The various ['lineto'](#) commands draw straight lines from the current point to a new point:

Command	Name	Parameters	Description

L (absolute) l (relative)	lineto	(x y)+	A line shall be drawn from the current point to the given (x,y) coordinate, which then shall become the new current point. If more than one coordinate pair is specified, a polyline shall be drawn. At the end of the command, the new current point shall be set to the final set of coordinates provided.
H (absolute) h (relative)	horizontal lineto	x+	A horizontal line shall be drawn from the current point (cpx, cpy) to (x, cpy). If more than one x value is specified, multiple horizontal lines shall be drawn (although usually this doesn't make sense). At the end of the command, the new current point shall be (x, cpy) for the final value of x.
V (absolute) v (relative)	vertical lineto	y+	A vertical line shall be drawn from the current point (cpx, cpy) to (cpx, y). If more than one y value is specified, multiple vertical lines shall be drawn (although usually this doesn't make sense). At the end of the command, the new current point shall be (cpx, y) for the final value of y.

8.3.5 The Curve commands

These groups of commands draw curves:

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

8.3.6 The Cubic Bézier curve commands

The '[Cubic Bézier](#)' commands are as follows:

Command	Name	Parameters	Description

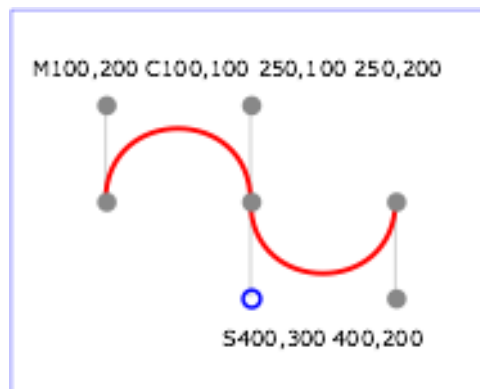
<p>C (absolute) c (relative)</p>	<p>curveto</p>	<p>(x1 y1 x2 y2 x y)+</p>	<p>A cubic Bézier curve shall be drawn from the current point to (x, y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. If multiple sets of coordinates are specified, a polybézier shall be drawn. At the end of the command, the new current point shall be the final (x,y) coordinate pair used in the polybézier.</p>
<p>S (absolute) s (relative)</p>	<p>shorthand/smooth curveto</p>	<p>(x2 y2 x y)+</p>	<p>A cubic Bézier curve shall be drawn from the current point to (x, y). The first control point shall 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, the first control point shall be coincident with the current point.) (x2,y2) shall be used as the second control point (i.e., the control point at the end of the curve). If multiple sets of coordinates are specified, a polybézier shall be drawn. At the end of the command, the</p>

new current point shall be the final (x,y) coordinate pair used in the polybézier.

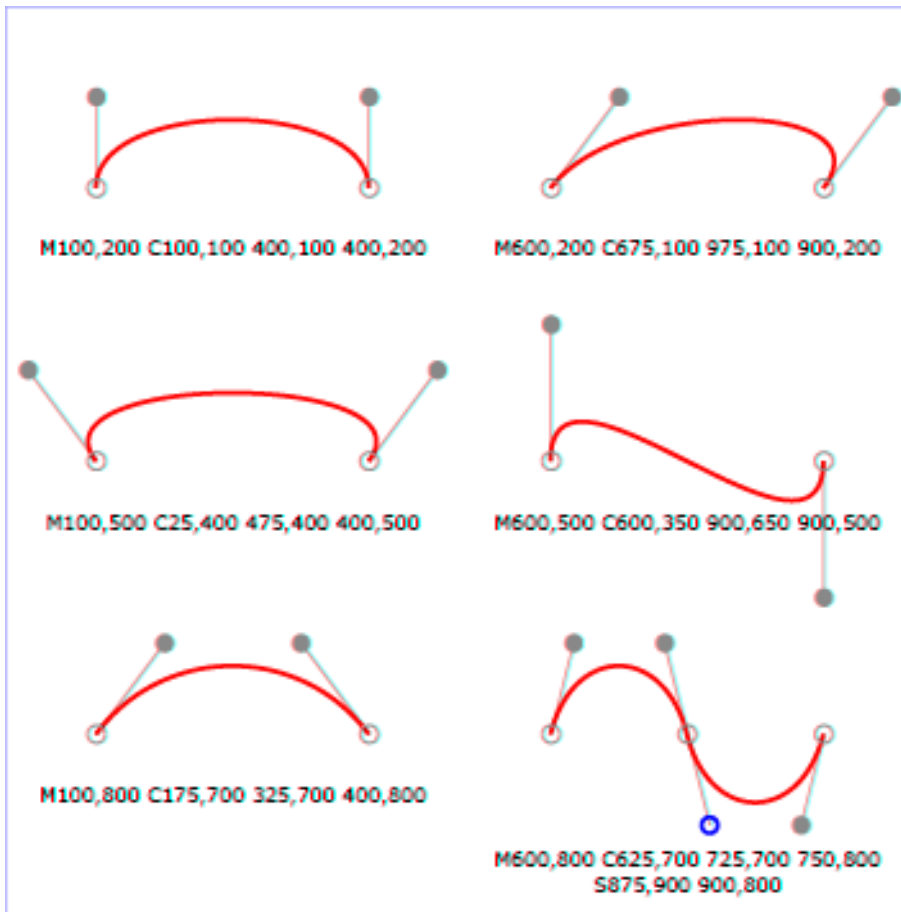
Example 08_02 shows some simple uses of 'Cubic Bézier' commands within a 'path'. Note that the control point for the "S" command is computed automatically as the reflection of the control point for the previous "C" command relative to the start point of the "S" command.

Example: 08_02.svg

```
<?xml version="1.0"?>
<svg width="5cm" height="4cm" viewBox="0 0 500 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <title>Example cubic01- cubic Bézier commands in path data</title>
  <desc>Picture showing a simple example of path data
    using both a "C" and an "S" command,
    along with annotations showing the control points
    and end points</desc>
  <rect fill="none" stroke="blue" stroke-width="1" x="1" y="1" width="498" height="398" />
  <polyline fill="none" stroke="#888888" stroke-width="1" points="100,200 100,100" />
  <polyline fill="none" stroke="#888888" stroke-width="1" points="250,100 250,200" />
  <polyline fill="none" stroke="#888888" stroke-width="1" points="250,200 250,300" />
  <polyline fill="none" stroke="#888888" stroke-width="1" points="400,300 400,200" />
  <path fill="none" stroke="red" stroke-width="5" d="M100,200 C100,100 250,100 250,200
    S400,300 400,200" />
  <circle fill="#888888" stroke="none" stroke-width="2" cx="100" cy="200" r="10" />
  <circle fill="#888888" stroke="none" stroke-width="2" cx="250" cy="200" r="10" />
  <circle fill="#888888" stroke="none" stroke-width="2" cx="400" cy="200" r="10" />
  <circle fill="#888888" stroke="none" cx="100" cy="100" r="10" />
  <circle fill="#888888" stroke="none" cx="250" cy="100" r="10" />
  <circle fill="#888888" stroke="none" cx="400" cy="300" r="10" />
  <circle fill="none" stroke="blue" stroke-width="4" cx="250" cy="300" r="9" />
  <text font-size="22" font-family="Verdana" x="25" y="70">M100,200 C100,100 250,100 250,200</text>
  <text font-size="22" font-family="Verdana" x="325" y="350"
    text-anchor="middle">S400,300 400,200</text>
</svg>
```



The following picture shows some how cubic Bézier curves change their shape depending on the position of the control points. The first five examples illustrate a single cubic Bézier path segment. The example at the lower right shows a "C" command followed by an "S" command.



[View this example as SVG \(SVG-enabled browsers only\)](#)

8.3.7 The Quadratic Bézier curve commands

The '[Quadratic Bézier](#)' commands are as follows:

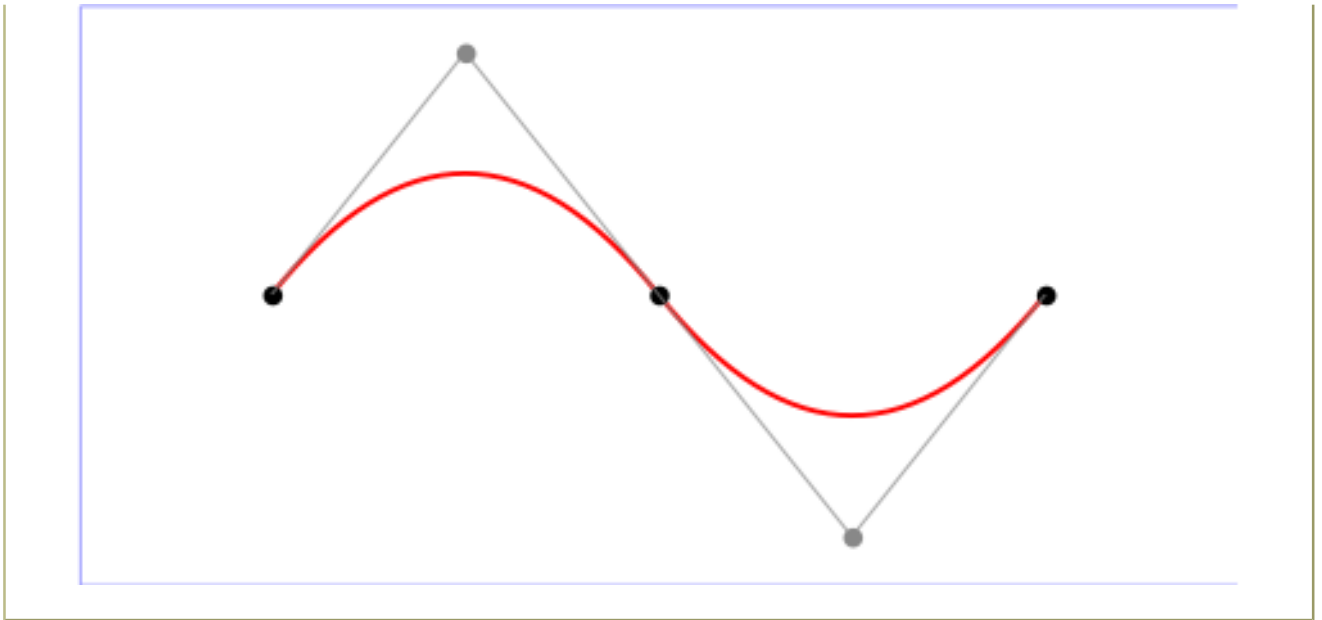
Command	Name	Parameters	Description
Q (absolute) q (relative)	quadratic Bézier curveto	$(x_1\ y_1\ x\ y)^+$	A quadratic Bézier curve is drawn from the current point to (x,y) using (x_1,y_1) as the control point. If multiple sets of coordinates are specified, a polybézier shall be drawn. At the end of the command, the new current point shall be the final (x,y) coordinate pair used in the polybézier.

<p>T (absolute) t (relative)</p>	<p>Shorthand/smooth quadratic Bézier curveto</p>	<p>(x y)+</p>	<p>A quadratic Bézier curve is drawn from the current point to (x,y). The control point shall be the reflection of the control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not a Q, q, T or t, the control point shall be current point.) If multiple sets of coordinates are specified, a polybézier shall be drawn. At the end of the command, the new current point shall be the final (x,y) coordinate pair used in the polybézier.</p>
--	--	---------------	--

Example quad01 shows some simple uses of ['Quadratic Bézier'](#) commands within a path. Note that the control point for the "T" command is computed automatically as the reflection of the control point for the previous "Q" command relative to the start point of the "T" command.

Example: [08_03.svg](#)

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="6cm" viewBox="0 0 1200 600"
xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
<title>Example quad01 - quadratic Bezier commands in path data</title>
<desc>Picture showing a "Q" a "T" command,
along with annotations showing the control points
and end points</desc>
<rect x="1" y="1" width="1198" height="598"
fill="none" stroke="blue" stroke-width="1" />
<path d="M200,300 Q400,50 600,300 T1000,300"
fill="none" stroke="red" stroke-width="5" />
<!-- End points -->
<g fill="black" >
<circle cx="200" cy="300" r="10"/>
<circle cx="600" cy="300" r="10"/>
<circle cx="1000" cy="300" r="10"/>
</g>
<!-- Control points and lines from end points to control points -->
<g fill="#888888" >
<circle cx="400" cy="50" r="10"/>
<circle cx="800" cy="550" r="10"/>
</g>
<path d="M200,300 L400,50 L600,300
L800,550 L1000,300"
fill="none" stroke="#888888" stroke-width="2" />
</svg>
```



8.3.8 The grammar for path data

The following description of the grammar for path data uses Extended Backus-Naur Form [\[EBNF\]](#) :

```

svg-path:
  wsp* moveto-drawto-command-groups? wsp*
moveto-drawto-command-groups:
  moveto-drawto-command-group
  | moveto-drawto-command-group wsp* moveto-drawto-command-groups
moveto-drawto-command-group:
  moveto wsp* drawto-commands?
drawto-commands:
  drawto-command
  | drawto-command wsp* drawto-commands
drawto-command:
  closepath
  | lineto
  | horizontal-lineto
  | vertical-lineto
  | curveto
  | smooth-curveto
  | quadratic-bezier-curveto
  | smooth-quadratic-bezier-curveto
moveto:
  ( "M" | "m" ) wsp* moveto-argument-sequence
moveto-argument-sequence:
  coordinate-pair
  | coordinate-pair comma-wsp? lineto-argument-sequence
closepath:
  ( "Z" | "z" )
lineto:
  ( "L" | "l" ) wsp* lineto-argument-sequence
lineto-argument-sequence:
  coordinate-pair
  | coordinate-pair comma-wsp? lineto-argument-sequence
horizontal-lineto:
  ( "H" | "h" ) wsp* horizontal-lineto-argument-sequence
horizontal-lineto-argument-sequence:
  coordinate
  | coordinate comma-wsp? horizontal-lineto-argument-sequence
vertical-lineto:
  ( "V" | "v" ) wsp* vertical-lineto-argument-sequence
vertical-lineto-argument-sequence:
  coordinate
  | coordinate comma-wsp? vertical-lineto-argument-sequence
curveto:
  ( "C" | "c" ) wsp* curveto-argument-sequence
curveto-argument-sequence:
  curveto-argument
  | curveto-argument comma-wsp? curveto-argument-sequence
curveto-argument:
  coordinate-pair comma-wsp? coordinate-pair comma-wsp? coordinate-pair
smooth-curveto:
  ( "S" | "s" ) wsp* smooth-curveto-argument-sequence

```

```

smooth-curve-to-argument-sequence:
  smooth-curve-to-argument
  | smooth-curve-to-argument comma-wsp? smooth-curve-to-argument-sequence
smooth-curve-to-argument:
  coordinate-pair comma-wsp? coordinate-pair
quadratic-bezier-curve-to:
  ( "Q" | "q" ) wsp* quadratic-bezier-curve-to-argument-sequence
quadratic-bezier-curve-to-argument-sequence:
  quadratic-bezier-curve-to-argument
  | quadratic-bezier-curve-to-argument comma-wsp?
  quadratic-bezier-curve-to-argument-sequence
quadratic-bezier-curve-to-argument:
  coordinate-pair comma-wsp? coordinate-pair
smooth-quadratic-bezier-curve-to:
  ( "T" | "t" ) wsp* smooth-quadratic-bezier-curve-to-argument-sequence
smooth-quadratic-bezier-curve-to-argument-sequence:
  coordinate-pair
  | coordinate-pair comma-wsp? smooth-quadratic-bezier-curve-to-argument-sequence
coordinate-pair:
  coordinate comma-wsp? coordinate
coordinate:
  number
nonnegative-number:
  integer-constant
  | floating-point-constant
number:
  sign? integer-constant
  | sign? floating-point-constant
flag:
  "0" | "1"
comma-wsp:
  (wsp+ comma? wsp*) | (comma wsp*)
comma:
  ","
integer-constant:
  digit-sequence
floating-point-constant:
  fractional-constant exponent?
  | digit-sequence exponent
fractional-constant:
  digit-sequence? "." digit-sequence
  | digit-sequence "."
exponent:
  ( "e" | "E" ) sign? digit-sequence
sign:
  "+" | "-"
digit-sequence:
  digit
  | digit digit-sequence
digit:
  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
wsp:
  (#x20 | #x9 | #xD | #xA)

```

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

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

Note that the EBNF allows the path 'd' attribute to be empty. This is not an error, instead it disables rendering of the path. Values of the 'd' that do not match the EBNF are treated as unsupported.

8.4 Distance along a path

Various operations, including [motion animation](#) and various [stroke operations](#), require that the user agent compute the distance along the geometry of a graphics element, such as a 'path'.

Exact mathematics exist for computing distance along a path, but the formulas are highly complex and require substantial computation. It is recommended that authoring products and user agents employ algorithms that produce as precise results as

possible; however, to accommodate implementation differences and to help distance calculations produce results that approximate author intent, the [pathLength](#) attribute can be used to provide the author's computation of the total length of the path so that the user agent can scale distance-along-a-path computations by the ratio of [pathLength](#) to the user agent's own computed value for total path length.

A "moveto" operation within a **'path'** element is defined to have zero length. Only the various "lineto" and "curveto" commands contribute to path length calculations.

9 Basic Shapes

Contents

- 9.1 [Introduction](#)
- 9.2 [The 'rect' element](#)
- 9.3 [The 'circle' element](#)
- 9.4 [The 'ellipse' element](#)
- 9.5 [The 'line' element](#)
- 9.6 [The 'polyline' element](#)
- 9.7 [The 'polygon' element](#)
 - 9.7.1 [The grammar for points specifications in 'polyline' and 'polygon' elements](#)
- 9.8 [Shape Module](#)

9.1 Introduction

SVG contains the following set of basic shape elements:

- [rectangles](#) (including optional rounded corners)
- [circles](#)
- [ellipses](#)
- [lines](#)
- [polylines](#)
- [polygons](#)

Mathematically, these shape elements are equivalent to a ['path'](#) element that would construct the same shape. The basic shapes may be stroked, and filled. All of the properties available for ['path'](#) elements also apply to the basic shapes.

9.2 The 'rect' element

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

Schema: rect

```
<define name='rect'>
  <element name='rect'>
    <ref name='rect.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='rect.AT' combine='interleave'>
  <ref name='svg.ShapeCommon.attr' />
  <ref name='svg.XYWH.attr' />
  <ref name='svg.RxRyCommon.attr' />
</define>
```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of the side of the rectangle which has the smaller x-axis coordinate value in the current

user coordinate system.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

y = "[<coordinate>](#)"

The y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value in the current user coordinate system.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

width = "[<length>](#)"

The width of the rectangle.

A negative value is unsupported. A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

height = "[<length>](#)"

The height of the rectangle.

A negative value is unsupported. A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

rx = "[<length>](#)"

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

A negative value is unsupported.

See the notes below about what happens if the attribute is not specified.

Animatable: yes.

ry = "[<length>](#)"

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

A negative value is unsupported.

See the notes below about what happens if the attribute is not specified.

Animatable: yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

If a properly specified value is provided for [rx](#) but not for [ry](#), then the user agent must process the **'rect'** element with the effective value for [ry](#) as equal to [rx](#). If a properly specified value is provided for [ry](#) but not for [rx](#), then the user agent must process the **'rect'** element with the effective value for [rx](#) as equal to [ry](#). If neither [rx](#) nor [ry](#) has a properly specified value, then the user agent must process the **'rect'** element as if no rounding had been specified, resulting in square corners. If [rx](#) is greater than half of the width of the rectangle, then the user agent must process the **'rect'** element with the effective value for [rx](#) as half of the width of the rectangle. If [ry](#) is greater than half of the height of the rectangle, then the user agent must process the **'rect'** element with the effective value for [ry](#) as half of the height of the rectangle.

Mathematically, a **'rect'** element, taking its rounded corners into account, must be drawn in a way that produces the same output as the following rules: (Note: all coordinate and length values are first converted into user space coordinates according to [Units](#).)

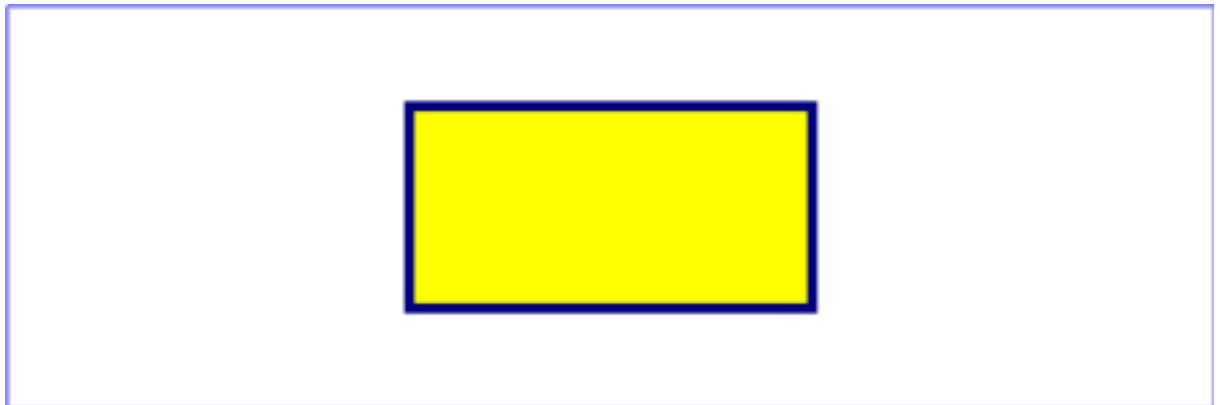
- perform an absolute [moveto](#) operation to location $(x+rx,y)$, where x is the value of the **'rect'** element's [x](#) attribute converted to user space, rx is the effective value of the [rx](#) attribute converted to user space and y is the value of the [y](#) attribute converted to user space
- perform an absolute horizontal [lineto](#) operation to location $(x+width-rx,y)$, where $width$ is the **'rect'** element's [width](#) attribute converted to user space

- perform an absolute elliptical arc operation to coordinate $(x+width,y+ry)$, where the effective values for the [rx](#) and [ry](#) attributes on the **'rect'** element converted to user space are used as the rx and ry attributes on the elliptical arc command, respectively, the x -axis-rotation is set to zero, the *large-arc-flag* is set to zero, and the *sweep-flag* is set to one
- perform a absolute vertical *lineto* to location $(x+width,y+height-ry)$, where *height* is the **'rect'** element's [height](#) attribute converted to user space
- perform an absolute elliptical arc operation to coordinate $(x+width-rx,y+height)$
- perform an absolute horizontal *lineto* to location $(x+rx,y+height)$
- perform an absolute elliptical arc operation to coordinate $(x,y+height-ry)$
- perform an absolute absolute vertical *lineto* to location $(x,y+ry)$
- perform an absolute elliptical arc operation to coordinate $(x+rx,y)$

Example 09_01 shows a rectangle with sharp corners. The **'rect'** element is filled with yellow and stroked with navy.

Example: 09_01.svg

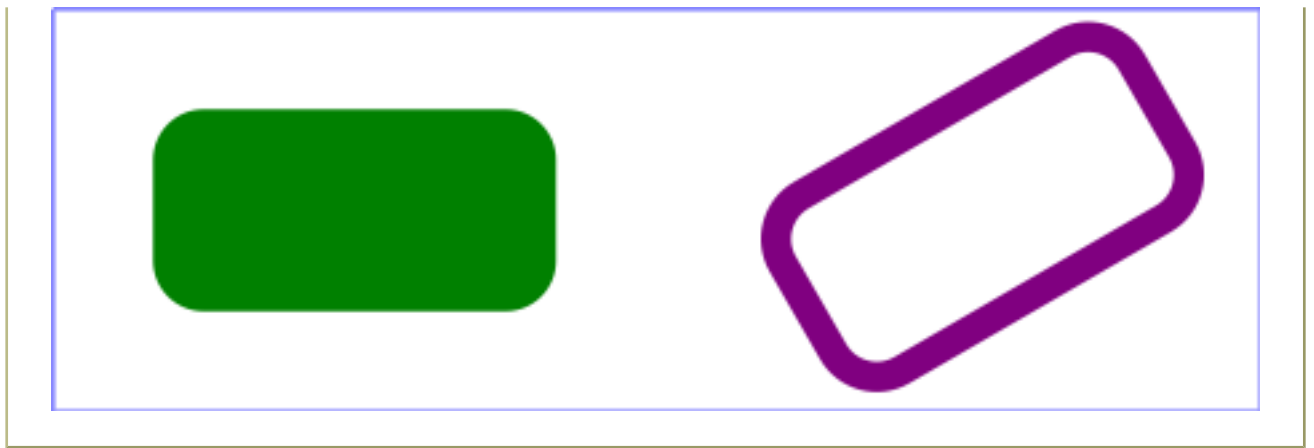
```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example rect01 - rectangle with sharp corners</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
    fill="none" stroke="blue" stroke-width="2"/>
  <rect x="400" y="100" width="400" height="200"
    fill="yellow" stroke="navy" stroke-width="10" />
</svg>
```



Example 09_02 shows two rounded rectangles. The [rx](#) specifies how to round the corners of the rectangles. Note that since no value has been specified for the [ry](#) attribute, it will be assigned the same value as the [rx](#) attribute.

Example: 09_02.svg

```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example rect02 - rounded rectangles</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
    fill="none" stroke="blue" stroke-width="2"/>
  <rect x="100" y="100" width="400" height="200" rx="50"
    fill="green" />
  <g transform="translate(700 210) rotate(-30)">
    <rect x="0" y="0" width="400" height="200" rx="50"
      fill="none" stroke="purple" stroke-width="30" />
  </g>
</svg>
```



9.3 The 'circle' element

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

Schema: circle

```
<define name='circle'>
  <element name='circle'>
    <ref name='circle.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='circle.AT' combine='interleave'>
  <ref name='svg.ShapeCommon.attr' />
  <ref name='svg.CxCy.attr' />
  <ref name='svg.R.attr' />
</define>
```

Attribute definitions:

cx = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

cy = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

r = "[<length>](#)"

The radius of the circle.

A negative value is unsupported. A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

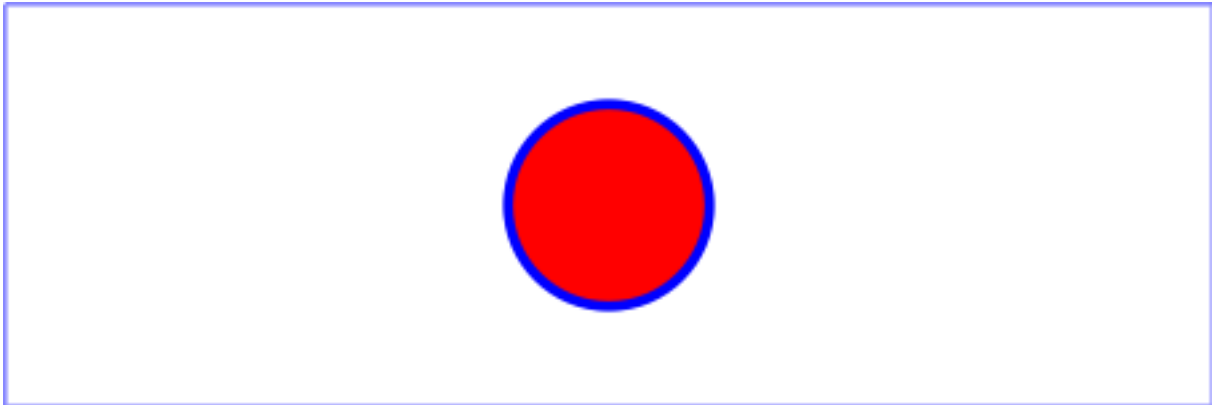
Navigation Attributes

See [definition](#).

Example circle01 consists of a 'circle' element that is filled with red and stroked with blue.

Example: 09_03.svg

```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
      xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example circle01 - circle filled with red and stroked with blue</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>
  <circle cx="600" cy="200" r="100"
          fill="red" stroke="blue" stroke-width="10" />
</svg>
```



9.4 The 'ellipse' element

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

Schema: ellipse

```
<define name='ellipse'>
  <element name='ellipse'>
    <ref name='ellipse.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='ellipse.AT' combine='interleave'>
  <ref name='svg.ShapeCommon.attr' />
  <ref name='svg.RxRyCommon.attr' />
  <ref name='svg.CxCy.attr' />
</define>
```

Attribute definitions:

cx = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

cy = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

rx = "[<length>](#)"

The x-axis radius of the ellipse.

A negative value is unsupported. A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

ry = "[<length>](#)"

The y-axis radius of the ellipse.

A negative value is unsupported. A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

[Animatable](#): Yes

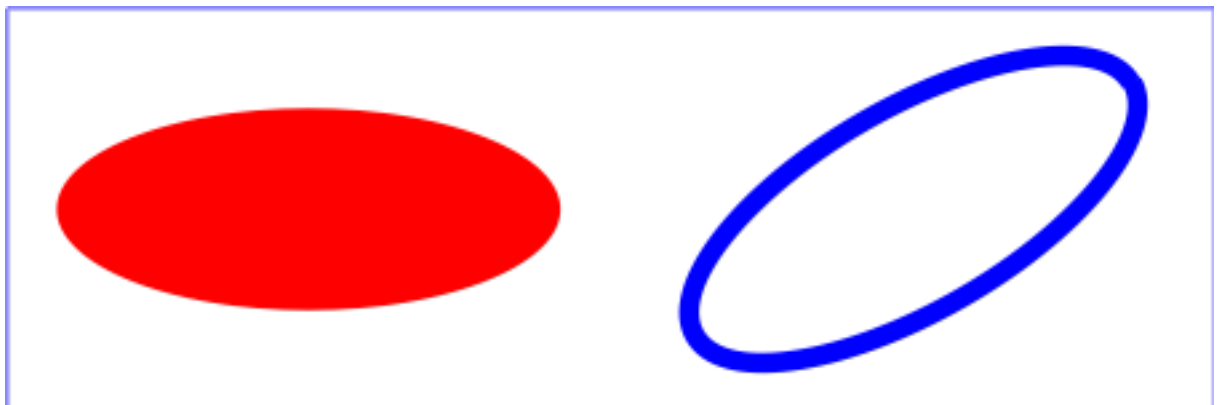
Navigation Attributes

See [definition](#).

Example 09_04 below specifies the coordinates of the two ellipses in the user coordinate system established by the [viewBox](#) attribute on the **'svg'** element and the [transform](#) attribute on the **'g'** and **'ellipse'** elements. Both ellipses use the default values of zero for the [cx](#) and [cy](#) attributes (the center of the ellipse). The second ellipse is rotated.

Example: 09_04.svg

```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example ellipse01 - examples of ellipses</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
    fill="none" stroke="blue" stroke-width="2" />
  <g transform="translate(300 200)">
    <ellipse rx="250" ry="100"
      fill="red" />
  </g>
  <ellipse transform="translate(900 200) rotate(-30)"
    rx="250" ry="100"
    fill="none" stroke="blue" stroke-width="20" />
</svg>
```



9.5 The 'line' element

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

Schema: line

```
<define name='line'>
  <element name='line'>
    <ref name='line.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='line.AT' combine='interleave'>
  <ref name='svg.ShapeCommon.attr' />
  <ref name='svg.X1Y12.attr' />
</define>
```

Attribute definitions:

x1 = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

y1 = "[<coordinate>](#)"

The y-axis coordinate of the start of the line.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

x2 = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

y2 = "[<coordinate>](#)"

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

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

Mathematically, a '[line](#)' element can be mapped to an equivalent '[path](#)' element as follows: (Note: all coordinate and length values are first converted into user space coordinates according to [Units](#).)

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

Because '[line](#)' elements are single lines and thus are geometrically one-dimensional, they have no interior; thus, '[line](#)' elements are never filled (see the '[fill](#)' property).

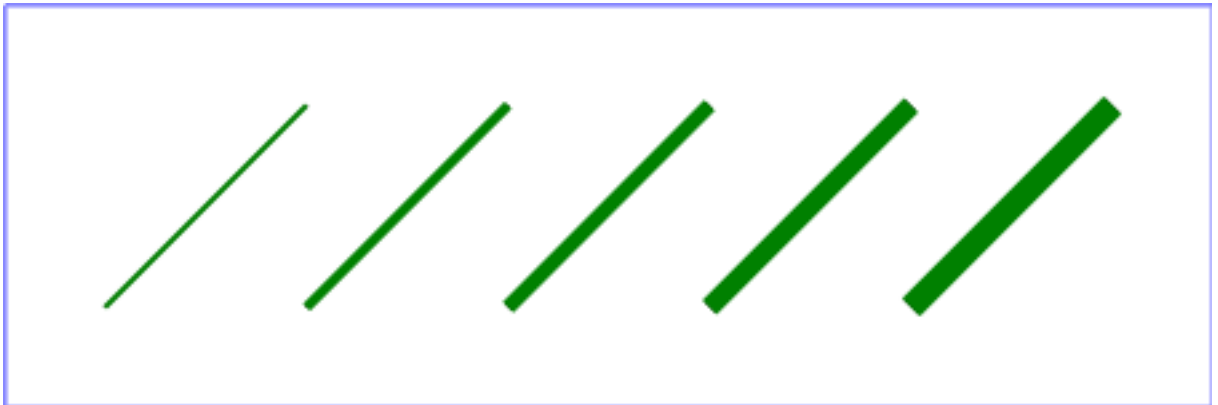
[Example 09_05](#) below specifies the coordinates of the five lines in the user coordinate system established by the [viewBox](#) attribute on the '[svg](#)' element. The lines have different thicknesses.

Example: [09_05.svg](#)

```

<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
      xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example line01 - lines expressed in user coordinates</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />
  <g stroke="green" >
    <line x1="100" y1="300" x2="300" y2="100"
          stroke-width="5" />
    <line x1="300" y1="300" x2="500" y2="100"
          stroke-width="10" />
    <line x1="500" y1="300" x2="700" y2="100"
          stroke-width="15" />
    <line x1="700" y1="300" x2="900" y2="100"
          stroke-width="20" />
    <line x1="900" y1="300" x2="1100" y2="100"
          stroke-width="25" />
  </g>
</svg>

```



9.6 The 'polyline' element

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

Schema: polyline

```

<define name='polyline'>
  <element name='polyline'>
    <ref name='polyCommon.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='polyCommon.AT' combine='interleave'>
  <ref name='svg.ShapeCommon.attr' />
  <attribute name='points' svg:animatable='true' svg:inheritable='false'>
    <ref name='Points.datatype' />
  </attribute>
</define>

```

Attribute definitions:

points = "[<list-of-points>](#)"

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

An empty attribute value (points="") disables rendering of the element. If the attribute is not specified, the effect is as if an empty string (points="") was specified. [Animatable](#): yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

If an odd number of coordinates is provided, then the element is treated as if the attribute had not been specified.

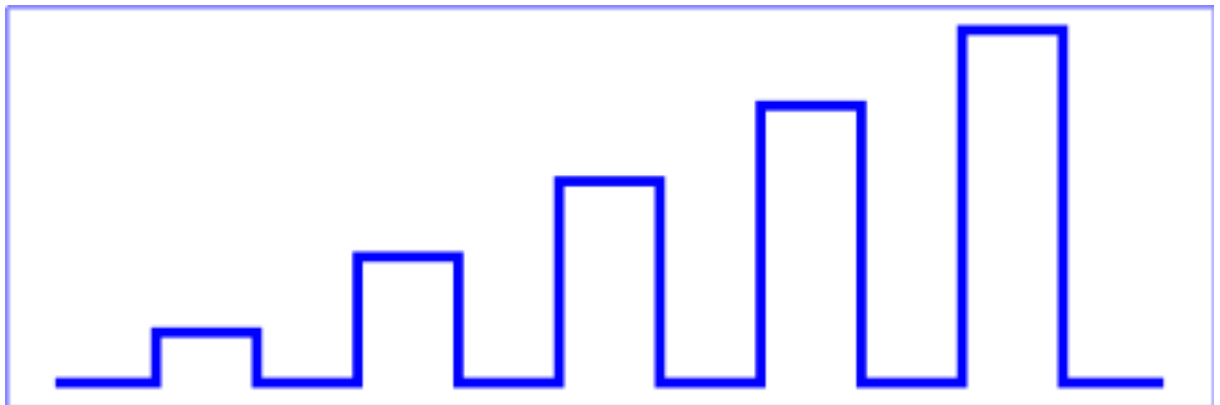
Mathematically, a **'polyline'** element can be mapped to an equivalent **'path'** element as follows:

- perform an absolute [moveto](#) operation to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute [lineto](#) operation to that coordinate pair.

Example 09_06 below specifies a polyline in the user coordinate system established by the [viewBox](#) attribute on the **'svg'** element.

Example: 09_06.svg

```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example polyline01 - increasingly larger bars</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
    fill="none" stroke="blue" stroke-width="2" />
  <polyline fill="none" stroke="blue" stroke-width="10"
    points="50,375
           150,375 150,325 250,325 250,375
           350,375 350,250 450,250 450,375
           550,375 550,175 650,175 650,375
           750,375 750,100 850,100 850,375
           950,375 950,25 1050,25 1050,375
           1150,375" />
</svg>
```



9.7 The **'polygon'** element

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

Schema: polygon

```
<define name='polygon'>
  <element name='polygon'>
    <ref name='polyCommon.AT' />
    <zeroOrMore><ref name='shapeCommon.CM' /></zeroOrMore>
  </element>
```

```
</define>
```

Attribute definitions:

points = "[<list-of-points>](#)"

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

An empty attribute value (points="") disables rendering of the element. If the attribute is not specified, the effect is as if an empty string (points="") was specified. [Animatable](#): yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

[Animatable](#): Yes

Navigation Attributes

See [definition](#).

If an odd number of coordinates is provided, then the element is treated as if the attribute had not been specified.

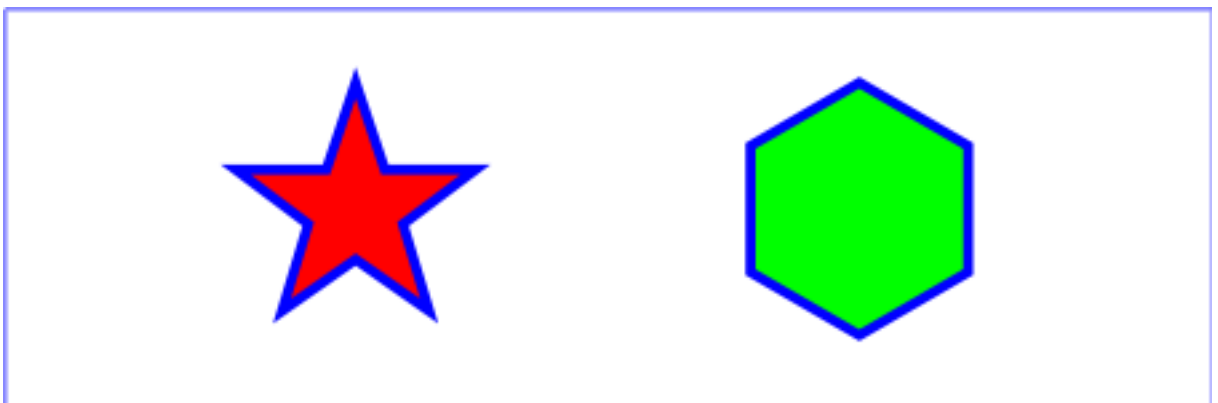
Mathematically, a '[polygon](#)' element can be mapped to an equivalent '[path](#)' element as follows:

- perform an absolute [moveto](#) operation to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute [lineto](#) operation to that coordinate pair
- perform a [closepath](#) command

[Example 09_07](#) below specifies two polygons (a star and a hexagon) in the user coordinate system established by the [viewBox](#) attribute on the '[svg](#)' element.

Example: 09_07.svg

```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example polygon01 - star and hexagon</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
    fill="none" stroke="blue" stroke-width="2" />
  <polygon fill="red" stroke="blue" stroke-width="10"
    points="350,75 379,161 469,161 397,215
           423,301 350,250 277,301 303,215
           231,161 321,161" />
  <polygon fill="lime" stroke="blue" stroke-width="10"
    points="850,75 958,137.5 958,262.5
           850,325 742,262.6 742,137.5" />
</svg>
```



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

The following is the Extended Backus-Naur Form [EBNF] for points specifications in 'polyline' and 'polygon' elements. The following notation is used:

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

```
list-of-points:
    wsp* coordinate-pairs? wsp*
coordinate-pairs:
    coordinate-pair
    | coordinate-pair comma-wsp coordinate-pairs
coordinate-pair:
    coordinate comma-wsp coordinate
coordinate:
    number
number:
    sign? integer-constant
    | sign? floating-point-constant
comma-wsp:
    (wsp+ comma? wsp*) | (comma wsp*)
comma:
    ","
integer-constant:
    digit-sequence
floating-point-constant:
    fractional-constant exponent?
    | digit-sequence exponent
fractional-constant:
    digit-sequence? "." digit-sequence
    | digit-sequence "."
exponent:
    ( "e" | "E" ) sign? digit-sequence
sign:
    "+" | "-"
digit-sequence:
    digit
    | digit digit-sequence
digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
wsp:
    (#x20 | #x9 | #xD | #xA)+
```

9.8 Shape Module

The Shape Module contains the following elements:

- [path](#)
- [rect](#)
- [circle](#)
- [ellipse](#)
- [polygon](#)
- [polyline](#)
- [line](#)

[RNG] [Feature String]

10 Text

Contents

- 10.1 [Introduction](#)
- 10.2 [Characters and their corresponding glyphs](#)
- 10.3 [Fonts, font tables and baselines](#)
- 10.4 [The 'text' element](#)
- 10.5 [The 'tspan' element](#)
- 10.6 [Text layout](#)
 - 10.6.1 [Text layout introduction](#)
 - 10.6.2 [Relationship with bidirectionality](#)
- 10.7 [Text rendering order](#)
- 10.8 [Alignment properties](#)
 - 10.8.1 [Text alignment properties](#)
- 10.9 [Font selection properties](#)
- 10.10 [White space handling](#)
- 10.11 [Text in an area](#)
 - 10.11.1 [Introduction to text in an area](#)
 - 10.11.2 [The 'textArea' element](#)
 - 10.11.3 [The 'tbreak' element](#)
 - 10.11.4 [The 'line-increment' property](#)
 - 10.11.5 [The 'display-align' property](#)
 - 10.11.6 [Text in an area layout rules](#)
- 10.12 [Editable Text Fields](#)
 - 10.12.1 [The editable attribute](#)
- 10.13 [Text selection and clipboard operations](#)
- 10.14 [Text Module](#)

10.1 Introduction

Text that is to be rendered as part of an SVG document fragment is specified using the **'text'** or **'textArea'** elements. The characters to be drawn are expressed as XML character data [[XML11](#)] inside the element.

SVG's **'text'** and **'textArea'** elements are rendered like other graphics elements. Thus, [coordinate system transformations](#) and [painting](#) features apply to text elements in the same way as they apply to [shapes](#) such as [paths](#) and [rectangles](#).

Each **'text'** element causes a single string of text to be rendered. The **'text'** element performs no automatic line breaking or word wrapping. To achieve the effect of multiple lines of text, use one of the following methods:

- The **'textArea'** element
- The author or authoring package needs to pre-compute the line breaks and use **'tbreak'** elements.
- Express the text to be rendered in another XML namespace such as XHTML [[XHTML](#)] embedded inline within a **'foreignObject'** element. (Note: the exact semantics of this approach are not completely defined at this time.)

The text strings within **'text'** elements can be rendered in one straight line (or more, if there are **'tbreak'** elements). SVG supports the following international text processing features for straight line text:

- left-to-right or bidirectional text (i.e., languages which intermix right-to-left and left-to-right text, such as Arabic and Hebrew)
- when [SVG fonts](#) are used, automatic selection of the correct glyph corresponding to the current form for [Arabic](#) and [Han](#) text

The layout rules for straight line text are described in [Text layout](#).

Because SVG text is packaged as XML character data [XML11]:

- Text data in SVG content is readily accessible to the visually impaired (see [Accessibility Support](#))
- In many viewing scenarios, the user will be able to search for and select text strings and copy selected text strings to the system clipboard (see [Text selection and clipboard operations](#))
- XML-compatible Web search engines will find text strings in SVG content with no additional effort over what they need to do to find text strings in other XML documents

Multi-language SVG content is possible by [substituting different text strings based on the user's preferred language](#).

For accessibility reasons, it is recommended that text which is included in a document have appropriate semantic markup to indicate its function. See [SVG accessibility guidelines](#) for more information.

10.2 Characters and their corresponding glyphs

In XML [XML11], textual content is defined in terms of a sequence of XML **characters**, where each character is defined by a particular Unicode code point [UNICODE]. Fonts, on the other hand, consist of a collection of **glyphs** and other associated information, such as font tables. A glyph is a presentable form of one or more characters (or a part of a character in some cases). Each glyph consists of some sort of identifier (in some cases a string, in other cases a number) along with drawing instructions for rendering that particular glyph.

In many cases, there is a one-to-one mapping of Unicode characters (i.e., Unicode code points) to glyphs in a font. For example, it is common for a font designed for Latin languages (where the term *Latin* is used for European languages such as English with alphabets similar to or derivative to the Latin language) to contain a single glyph for each of the standard ASCII characters (i.e., A-to-Z, a-to-z, 0-to-9, plus the various punctuation characters found in ASCII). Thus, in most situations, the string "XML", which consists of three Unicode characters, would be rendered by the three glyphs corresponding to "X", "M" and "L", respectively.

In various other cases, however, there is not a strict one-to-one mapping of Unicode characters to glyphs. Some of the circumstances when the mapping is not one-to-one:

- Ligatures - For best looking typesetting, it is often desirable that particular sequences of characters are rendered as a single glyph. An example is the word "office". Many fonts will define an "ffi" ligature. When the word "office" is rendered, sometimes the user agent will render the glyph for the "ffi" ligature instead of rendering distinct glyphs (i.e., "f", "f" and "i") for each of the three characters. Thus, for ligatures, multiple Unicode characters map to a single glyph. (Note that for proper rendering of some languages, ligatures are required for certain character combinations.)
- Composite characters - In various situations, commonly used adornments such as diacritical marks will be stored once in a font as a particular glyph and then composed with one or more other glyphs to result in the desired character. For example, it is possible that a font engine might render the é character by first rendering the glyph for e and then rendering the glyph for ´ (the accent mark) such that the accent mark will appear over the e. In this situation, a single Unicode character maps to multiple glyphs.
- Glyph substitution - Some typography systems examine the nature of the textual content and utilize different glyphs in different circumstances. For example, in Arabic, the same Unicode character might render as any of four different glyphs, depending on such factors as whether the character appears at the start, the end or the middle of a sequence of cursively joined characters. Different glyphs might be used for a punctuation character depending on inline-progression-direction (e.g., horizontal vs. vertical). In these situations, a single Unicode character might map to one of several alternative glyphs.
- In some languages, particular sequences of characters will be converted into multiple glyphs such that parts of a particular character are in one glyph and the remainder of that character is in another glyph.

In many situations, the algorithms for mapping from characters to glyphs are system-dependent, resulting in the possibility that the rendering of text might be (usually slightly) different when viewed in different user environments. If the author of SVG content requires precise selection of fonts and glyphs, then it is recommended that the necessary fonts (potentially subsetted to include only the glyphs needed for the given document) be available either as [SVG fonts](#) embedded within the SVG content or as [WebFonts](#) posted at the same Web location as the SVG content.

Throughout this chapter, the term **character** shall be equivalent to the definition of a character in XML [XML11].

10.3 Fonts, font tables and baselines

A font consists of a collection of glyphs together with the information (the font tables) necessary to use those glyphs to present characters on some medium. The combination of the collection of glyphs and the font tables is called the *font data*. The font tables include the information necessary to map characters to glyphs, to determine the size of glyph areas and to position the glyph area. Each font table consists of one or more font characteristics, such as the font-weight and font-style.

The geometric font characteristics are expressed in a coordinate system based on the EM box. (The EM is a relative measure of the height of the glyphs in the font; see [CSS2 em square](#).) The box 1 EM high and 1 EM wide is called the *design space*. This space is given a geometric coordinates by sub-dividing the EM into a number of [units-per-em](#).

Note: Units-per-em is a font characteristic. A typical value for units-per-EM is 1000 or 2048.

The coordinate space of the EM box is called the *design space coordinate system*. For scalable fonts, the curves and lines that are used to draw a glyph are represented using this coordinate system.

Note: Most often, the (0,0) point in this coordinate system is positioned on the left edge of the EM box, but not at the bottom left corner. The Y coordinate of the bottom of a roman capital letter is usually zero. And the descenders on lowercase roman letters have negative coordinate values.

SVG assumes that the font tables will provide at least three font characteristics: an ascent, a descent and a set of baseline-tables. The ascent is the distance to the top of the EM box from the (0,0) point of the font; the descent is the distance to the bottom of the EM box from the (0,0) point of the font. The baseline-table is explained below.

Note: Within an OpenType font, for horizontal writing-modes, the ascent and descent are given by the sTypoAscender and sTypoDescender entries in the OS/2 table. For vertical writing-modes, the descent (the distance, in this case from the (0,0) point to the left edge of the glyph) is normally zero because the (0,0) point is on the left edge. The ascent for vertical writing-modes is either 1 em or is specified by the ideographic top baseline value in the OpenType Base table for vertical writing-modes.

In horizontal writing-modes, the glyphs of a given script are positioned so that a particular point on each glyph, the [alignment-point](#), is aligned with the alignment-points of the other glyphs in that script. The glyphs of different scripts, for example, Western, Northern Indic and Far-Eastern scripts, are typically aligned at different points on the glyph. For example, Western glyphs are aligned on the bottoms of the capital letters, northern indic glyphs are aligned at the top of a horizontal stroke near the top of the glyphs and far-eastern glyphs are aligned either at the bottom or center of the glyph. Within a script and within a line of text having a single font-size, the sequence of alignment-points defines, in the inline- progression-direction, a geometric line called a *baseline*. Western and most other alphabetic and syllabic glyphs are aligned to an "alphabetic" baseline, the northern indic glyphs are aligned to a "hanging" baseline and the far-eastern glyphs are aligned to an "ideographic" baseline.

A *baseline-table* specifies the position of one or more baselines in the design space coordinate system. The function of the baseline table is to facilitate the alignment of different scripts with respect to each other when they are mixed on the same text line. Because the desired relative alignments may depend on which script is dominant in a line (or block), there may be a different baseline table for each script. In addition, different alignment positions are needed for horizontal and vertical writing modes. Therefore, the font may have a set of baseline tables: typically, one or more for horizontal writing-modes and zero or more for vertical writing-modes.

Note: Some fonts may not have values for the baseline tables. Heuristics are suggested for approximating the baseline tables when a given font does not supply baseline tables.

SVG further assumes that for each glyph in the font data for a font, there is a width value, an alignment-baseline and an alignment-point for horizontal writing-mode. (Vertical writing-mode is not supported in SVG Tiny 1.2).

In addition to the font characteristics required above, a font may also supply substitution and positioning tables that can be used by a formatter to re-order, combine and position a sequence of glyphs to make one or more composite glyphs. The combination may be as simple as a ligature, or as complex as an indic syllable which combines, usually with some re-ordering, multiple consonants and vowel glyphs.

10.4 The 'text' element

The 'text' element defines a graphics element consisting of text. The XML [\[XML11\]](#) character data within the 'text' element, along with relevant attributes and properties and character-to-glyph mapping tables within the font itself, define the glyphs to be rendered. (See [Characters and their corresponding glyphs](#).) The attributes and properties on the 'text' element indicate such things as the writing direction, font specification and painting attributes which describe how exactly to render the characters. Subsequent sections of this chapter describe the relevant text-specific attributes and properties, particular [text layout](#) and [bidirectionality](#).

Since 'text' elements are rendered using the same rendering methods as other graphics elements, all of the same [coordinate system transformations](#) and [painting](#) features that apply to [shapes](#) such as [paths](#) and [rectangles](#) also apply to 'text' elements.

Text behaves like other graphical objects, and it is therefore possible to apply a gradient to text. When this facility is applied to text then the object bounding box units are computed relative to the entire 'text' element in all cases, even when different effects are applied to different [tspan](#) elements within the same 'text' element.

The **'text'** element renders its first glyph (after [bidirectionality](#) reordering) at the initial [current text position](#), which is established by the **x** and **y** attributes on the **'text'** element (with possible adjustments due to the value of the **'text-anchor'** property). After the glyph (s) corresponding to the given character is (are) rendered, the current text position is updated for the next character. In the simplest case, the new current text position is the previous current text position plus the glyphs' advance value. See [text layout](#) for a description of glyph placement and glyph advance.

Schema: text

```
<define name='text'>
  <element name='text'>
    <ref name='text.AT' />
    <zeroOrMore><ref name='svg.TextCommon.group' /></zeroOrMore>
  </element>
</define>

<define name='text.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Editable.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.Transform.attr' />
  <optional>
    <attribute name='x' svg:animatable='true' svg:inheritable='false'>
      <ref name='Coordinates.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='y' svg:animatable='true' svg:inheritable='false'>
      <ref name='Coordinates.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='rotate' svg:animatable='true' svg:inheritable='false'>
      <ref name='Numbers.datatype' />
    </attribute>
  </optional>
</define>

<define name='svg.TextCommon.group' combine='choice'>
  <choice>
    <ref name='svg.Desc.group' />
    <ref name='svg.Animate.group' />
    <ref name='svg.Handler.group' />
    <ref name='svg.Discard.group' />
    <ref name='tspan' />
    <text />
    <element name='switch'>
      <ref name='switch.AT' />
      <zeroOrMore><ref name='svg.TextCommon.group' /></zeroOrMore>
    </element>
    <element name='a'>
      <ref name='a.AT' />
      <zeroOrMore><ref name='svg.TextCommon-noA.group' /></zeroOrMore>
    </element>
  </choice>
</define>
```

Attribute definitions:

x = "[<coordinate>+](#)"

If a single [<coordinate>](#) is provided, then the value represents the new absolute X coordinate for the [current text position](#) for rendering the glyphs that correspond to the first character within this element or any of its descendants.

If a comma- or space-separated list of <n> [<coordinate>](#)s is provided, then the values represent new absolute X coordinates for the [current text position](#) for rendering the glyphs corresponding to each of the first <n> characters within this element or any of its descendants.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

y = "[<coordinate>+](#)"

The corresponding list of absolute Y coordinates for the glyphs corresponding to the characters within this element. The processing rules for the **'y'** attribute parallel the processing rules for the **'x'** attribute.

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

editable = "none | simple"

This attribute indicates whether the text can be edited. See the definition of the '[editable](#)' attribute.

Animatable: no.

rotate = "<number>+"

This attribute indicates the supplemental rotation about the [alignment-point](#) that must be applied to the glyphs corresponding to characters within this element according to the following rules:

A comma- or space-separated list of <number>s must be provided. The first <number> specifies the supplemental rotation that must be applied to the glyphs corresponding to the first character within this element or any of its descendants, the second <number> specifies the supplemental rotation that must be applied to the glyphs that correspond to the second character, and so on.

If more <number>s are provided than there are characters, then the extra <number>s must be ignored.

If more characters are provided than <number>s, then for each of these extra characters the rotation value specified by the last number must be used.

Where multiple characters map to one glyph, the rotation specified for the first character of the ligature should be used for the glyph, and the subsequent rotations for the other contributing characters should be ignored.

This supplemental rotation must have no impact on the rules by which [current text position](#) as glyphs get rendered.

Animatable: yes (non-additive, 'set' and 'animate' elements only).

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

Example [text01](#) below contains the text string "Hello, out there" which will be rendered onto the canvas using the Verdana font family with the glyphs filled with the color blue.

Example: [10_01.svg](#)

```
<?xml version="1.0"?>
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example text01 - 'Hello, out there' in blue</desc>
  <text x="250" y="150"
    font-family="Verdana" font-size="55" fill="blue" >
    Hello, out there
  </text>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
    fill="none" stroke="blue" stroke-width="2" />
</svg>
```



Hello, out there

10.5 The 'tspan' element

Within a '[text](#)' or '[textArea](#)' element, graphic and font properties can be adjusted by including a '[tspan](#)' element.

Schema: [tspan](#)

```

<define name='tspan'>
  <element name='tspan'>
    <ref name='tspan.AT' />
    <zeroOrMore><ref name='svg.TextCommon.group' /></zeroOrMore>
  </element>
</define>

<define name='tspan.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.Focus.attr' />
</define>

```

The following examples show basic use of the **'tspan'** element.

Example `tspan01` uses a **'tspan'** element to indicate that the word "not" is to use a bold font and have red fill.

Example: `10_03.svg`

```

<?xml version="1.0"?>
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example tspan01 - using tspan to change visual attributes</desc>
  <g font-family="Verdana" font-size="45" >
    <text x="200" y="150" fill="blue" >
      You are
      <tspan font-weight="bold" fill="red" >not</tspan>
      a banana.
    </text>
  </g>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
    fill="none" stroke="blue" stroke-width="2" />
</svg>

```



You are **not** a banana.

Within a **'text'** or **'textArea'** element, graphic and font properties can be adjusted by including a **'tspan'** element. Positional attributes such as `x`, `y`, `dx`, `dy` and `rotate` are not available on **'tspan'** in SVG Tiny 1.2.

Attribute definitions:

focusable = `"true" | "false" | "auto"`
 See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes
 See [definition](#).

10.6 Text layout

10.6.1 Text layout introduction

This section describes the text layout features supported by SVG. Text layout is described in a general and directionally neutral way, to provide a single reference point for layout information which applies to left-to-right (e.g., Latin scripts), bidirectional (e.g., Hebrew or Arabic) and vertical (e.g., Asian scripts). In SVG Tiny 1.2, vertical writing is not supported. The descriptions in this

section assume straight line text (i.e., text that is either strictly horizontal or vertical with respect to the current user coordinate system).

For each **'text'** and **'textArea'** element, the SVG user agent determines the current **reference orientation**. The reference orientation is the vector pointing towards negative infinity in Y within the current user coordinate system. (Note: in the [initial coordinate system](#), the reference orientation is up.)

Based on the reference orientation the SVG user agent determines the current **inline-progression-direction**. For left-to-right text, the inline-progression-direction points 90 degrees clockwise from the reference orientation vector. For right-to-left text, the inline progression points 90 degrees counter-clockwise from the reference orientation vector.

Based on the reference orientation the SVG user agent determines the current **block-progression-direction**. For left-to-right and right-to-left text, the block-progression-direction points 180 degrees from the reference orientation vector because the only available horizontal writing modes are **lr-tb** and **rl-tb**. For top-to-bottom text, the block-progression-direction always points 90 degrees counter-clockwise from the reference orientation vector because the only available top-to-bottom writing mode is **tb-rl**.

In processing a given **'text'** or **'textArea'** element, the SVG user agent keeps track of the **current text position**. The initial current text position is established by the **x** and **y** attributes on the **'text'** or **'textArea'** element.

The current text position is adjusted after each glyph to establish a new current text position at which the next glyph shall be rendered. The adjustment to the current text position is based on the current inline-progression-direction, glyph-specific advance values corresponding to the glyph orientation of the glyph just rendered, kerning tables in the font and the current values of various attributes and properties, such as the spacing properties and any **x** and **y** attributes on **'text'** and **'textArea'** elements. If a glyph does not provide explicit advance values corresponding to the current glyph orientation, then an appropriate approximation should be used. For vertical text, a suggested approximation is the sum of the ascent and descent values for the glyph. Another suggested approximation for an advance value for both horizontal and vertical text is the size of an *em* (see [units-per-em](#)).

For each glyph to be rendered, the SVG user agent determines an appropriate **alignment-point** on the glyph which will be placed exactly at the current text position. The alignment-point is determined based on glyph cell metrics in the glyph itself, the current inline-progression-direction and the glyph orientation relative to the inline-progression-direction. For most uses of Latin text the alignment-point in the glyph will be the intersection of left edge of the glyph cell (or some other glyph-specific x-axis coordinate indicating a left-side origin point) with the Latin baseline of the glyph. For many cases with top-to-bottom vertical text layout, the reference point will be either a glyph-specific origin point based on the set of vertical baselines for the font or the intersection of the center of the glyph with its *top line* (see [\[CSS2-topline\]](#) for a definition of *top line*). If a glyph does not provide explicit origin points corresponding to the current glyph orientation, then an appropriate approximation should be used, such as the intersection of the left edge of the glyph with the appropriate horizontal baseline for the glyph or intersection of the top edge of the glyph with the appropriate vertical baseline. If baseline tables are not available, user agents should establish baseline tables that reflect common practice.

Adjustments to the current text position are either **absolute position adjustments** or **relative position adjustments**. An absolute position adjustment occurs in the following circumstances:

- At the start of a **'text'** element
- At the start of a **'textArea'** element
- For each character within a **'text'** element which has an **x** or **y** attribute value assigned to it explicitly

All other position adjustments to the current text position are relative position adjustments.

Each absolute position adjustment defines a new **text chunk**. Absolute position adjustments impact text layout in the following ways:

- Ligatures only occur when a set of characters which might map to a ligature are all in the same text chunk.
- Each text chunk represents a separate block of text for alignment due to **'text-anchor'** property values.
- Reordering of characters due to [bidirectionality](#) only occurs within a text chunk. Reordering does *not* happen across text chunks.

The following additional rules apply to ligature formation:

- As in [\[CSS2-spacing\]](#), when the resultant space between two characters is not the same as the default letter spacing, user agents should either not use ligatures or not apply letter-spacing depending on their knowledge of which behavior will produce the most coherent results for the script being used (for example, when the letter-spacing is high it is likely that breaking ligatures will produce the best result with Roman scripts, but be less than optimal for Arabic scripts, where ignoring the letter-spacing and maintaining the ligatures will be preferred).
- Ligature formation must only occur between characters that are not separated by element markup, and must still be enabled between characters separated by other XML markup, such as comments, processing instructions, or CDATA

sections. Ligature processing must take place only after entity and character references have been resolved. For example, assuming that there is a ligature defined for the string "dahut" and that the '&hu-ent;' entity reference contains the string "hu", in all the following examples the "dahut" ligature will be enabled:

- `<text>Le dahut vit dans les Alpes grenobloises.</text>`
- `<text>Le da&hu-ent;t vit dans les Alpes grenobloises.</text>`
- `<text>Le da<!-- random comment -->hut vit dans les Alpes grenobloises.</text>`
- `<text>Le dahut vit dans les Alpes grenobloises.</text>`
- `<text>Le da![CDATA[hu]>t vit dans les Alpes grenobloises.</text>`

But it will not be enabled in the following case:

- `<text>Le da<tspan fill='orange'>h</tspan>ut vit dans les Alpes grenobloises.</text>`

- As mentioned above, ligature formation should not be enabled for the glyphs corresponding to characters within different text chunks.

10.6.2 Relationship with bidirectionality

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

The Unicode standard ([UNICODE], specifically UAX#9) defines a complex algorithm for determining the proper directionality of text. The algorithm consists of an implicit part based on character properties, as well as explicit controls for embeddings and overrides. The SVG user agent applies this bidirectional algorithm when determining the layout of characters within a **'text'** or **'textArea'** element.

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

The processing model for bidirectional text is as follows. The user agent processes the characters which are provided in **logical order** (i.e. the order the characters appear in the original document). The user agent determines the set of independent blocks within each of which it should apply the Unicode bidirectional algorithm. Each text chunk represents an independent block of text. While kerning or ligature processing might be font-specific, the preferred model is that kerning and ligature processing occurs between combinations of characters or glyphs after the characters have been re-ordered.

10.7 Text rendering order

The glyphs associated with the characters within **'text'** and **'textArea'** elements are rendered in the logical order of the characters in the original document, independent of any re-ordering necessary to implement bidirectionality. Thus, for text that goes right-to-left visually, the glyphs associated with the rightmost character are rendered before the glyphs associated with the other characters.

Additionally, each distinct glyph is rendered in its entirety (i.e., it is filled and stroked as specified by the **'fill'** and **'stroke'** properties) before the next glyph gets rendered.

10.8 Alignment properties

10.8.1 Text alignment properties

The **'text-anchor'** property is used to align (start-, middle- or end-alignment) a string of text relative to a given point.

The **'text-anchor'** property is applied to each individual text chunk within a given **'text'** element. Each text chunk has an initial current text position, which represents the point in the user coordinate system resulting from (depending on context) application of the **x** and **y** attributes on the **'text'** element assigned explicitly to the first rendered character in a text chunk.

'text-anchor'

Value: start | middle | end | inherit

Initial: start

Applies to: 'text' Element

Inherited: yes

Percentages: N/A

Media: visual

<u>Animatable:</u>	yes
<i>Computed value:</i>	Specified value, except inherit

Values have the following meanings:

start

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

middle

The rendered characters are aligned such that the geometric middle of the text string is at the current text position.

end

The rendered characters are aligned such that the end of the text string is at the initial current text position. For Latin text in its usual orientation, this is comparable to right alignment.

10.9 Font selection properties

SVG uses the following font specification properties. Except for any additional information provided in this specification, the normative definition of the property is in [\[CSS2-fonts\]](#).

'font-family'

<i>Value:</i>	[[<family-name> <generic-family>],]* [<family-name> <generic-family>] <u>inherit</u>
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	text content elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<u>Animatable:</u>	yes
<i>Computed value:</i>	Specified value, except inherit

This property indicates which font family is to be used to render the text, specified as a prioritized list of font family names and/or generic family names. Except for any additional information provided in this specification, the normative definition of the property is in [\[CSS2-font-family\]](#). The rules for expressing the syntax of CSS property values can be found at [\[CSS2-propdef\]](#).

'font-style'

<i>Value:</i>	normal italic oblique <u>inherit</u>
<i>Initial:</i>	normal
<i>Applies to:</i>	text content elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<u>Animatable:</u>	yes
<i>Computed value:</i>	Specified value, except inherit

This property specifies whether the text is to be rendered using a normal, italic or oblique face. Except for any additional information provided in this specification, the normative definition of the property is in [\[CSS2-font-style\]](#).

'font-variant'

<i>Value:</i>	normal small-caps <u>inherit</u>
<i>Initial:</i>	normal
<i>Applies to:</i>	text content elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	no
<i>Computed value:</i>	Specified value, except inherit

This property indicates whether the text is to be rendered using the normal glyphs for lowercase characters or using small-caps glyphs for lowercase characters. Except for any additional information provided in this specification, the normative definition of the property is in [[CSS2-font-variant](#)].

'font-weight'

<i>Value:</i>	normal bold bolder lighter 100 200 300 400 500 600 700 800 900 <u>inherit</u>
<i>Initial:</i>	normal
<i>Applies to:</i>	text content elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	See text.

This property refers to the boldness or lightness of the glyphs used to render the text, relative to other fonts in the same font family. Except for any additional information provided in this specification, the normative definition of the property is in [[CSS2-font-weight](#)].

The computed value of "font-weight" is either:

- one of the legal number values, or
- one of the legal number values combined with one or more of the relative values (bolder or lighter). This type of computed values is necessary to use when the font in question does not have all weight variations that are needed.

SVG does not specify how the computed value of font-weight is represented internally or externally.

'font-size'

<i>Value:</i>	<absolute-size> <relative-size> <length> <u>inherit</u>
<i>Initial:</i>	medium
<i>Applies to:</i>	text content elements
<i>Inherited:</i>	yes, the computed value is inherited
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	Absolute length

This property refers to the size of the font from baseline to baseline when multiple lines of text are set solid in a multiline layout environment. The SVG user agent processes the <length> as a height value in the current user coordinate system.

Except for any additional information provided in this specification, the normative definition of the property is in [[CSS2-font-size](#)].

10.10 White space handling

SVG supports the standard XML attribute **xml:space** to specify the handling of white space characters within a given **'text'** element's character data. The SVG user agent has special processing rules associated with this attribute as described below. These are behaviors that occur subsequent to XML parsing [XML11] and do not affect the contents of the [Document Object Model \(DOM\)](#) [DOM3].

xml:space is an inheritable attribute which can have one of two values:

- **default** (the initial value for `xml:space`) - When `xml:space="default"`, the SVG user agent will do the following using a copy of the original character data content. First, it will remove all newline characters. Then it will convert all tab characters into space characters. Then, it will strip off all leading and trailing space characters. Then, all contiguous space characters will be consolidated.
- **preserve** - When `xml:space="preserve"`, the SVG user agent will do the following using a copy of the original character data content. It will convert all newline and tab characters into space characters. Then, it will draw all space characters, including leading, trailing and multiple contiguous space characters. Thus, when drawn with `xml:space="preserve"`, the string "a b" (three spaces between "a" and "b") will produce a larger separation between "a" and "b" than "a b" (one space between "a" and "b").

The following example illustrates that line indentation can be important when using `xml:space="default"`. The fragment below show two pairs of similar **'text'** elements, with both **'text'** elements using **xml:space='default'**. For these examples, there is no extra white space at the end of any of the lines (i.e., the line break occurs immediately after the last visible character).

```
[01] <text xml:space='default'>
[02]   WS example
[03]   indented lines
[04] </text>
[05] <text xml:space='preserve'>WS example indented lines</text>
[06]
[07] <text xml:space='default'>
[08]WS example
[09]non-indented lines
[10] </text>
[11] <text xml:space='preserve'>WS examplennon-indented lines</text>
```

The first pair of **'text'** elements above show the effect of indented character data. The attribute **xml:space='default'** in the first **'text'** element instructs the user agent to:

- convert all tabs (if any) to space characters,
- strip out all line breaks (i.e., strip out the line breaks at the end of lines [01], [02] and [03]),
- strip out all leading space characters (i.e., strip out space characters before "WS example" on line [02]),
- strip out all trailing space characters (i.e., strip out space characters before "</text>" on line [04]),
- consolidate all intermediate space characters (i.e., the space characters before "indented lines" on line [03]) into a single space character.

The second pair of **'text'** elements above show the effect of non-indented character data. The attribute **xml:space='default'** in the third **'text'** element instructs the user agent to:

- convert all tabs (if any) to space characters,
- strip out all line breaks (i.e., strip out the line breaks at the end of lines [07], [08] and [09]),
- strip out all leading space characters (there are no leading space characters in this example),
- strip out all trailing space characters (i.e., strip out space characters before "</text>" on line [10]),
- consolidate all intermediate space characters into a single space character (in this example, there are no intermediate space characters).

Note that XML parsers are required to convert the standard representations for a newline indicator (e.g., the literal two-character sequence "#xD#xA" or the stand-alone literals #xD or #xA) into the single character #xA before passing character data to the application. (See [XML end-of-line handling](#).)

Any features in the SVG language or the SVG DOM that are based on character position number are based on character position after applying the white space handling rules described here. In particular, if `xml:space="default"`, it is often the case that white space characters are removed as part of processing. Character position numbers index into the text string after the white space characters have been removed per the rules in this section.

The `xml:space` attribute is:

`Animatable: no.`

10.11 Text in an area

10.11.1 Introduction to text in an area

The `'textArea'` element allows simplistic wrapping of text content within a given region. This profile of SVG specifies a single rectangular region. Other profiles may allow a sequence of arbitrary shapes.

Text wrapping via the `'textArea'` element is available as a lightweight and convenient facility for simple text wrapping where a complete box model layout engine is not required.

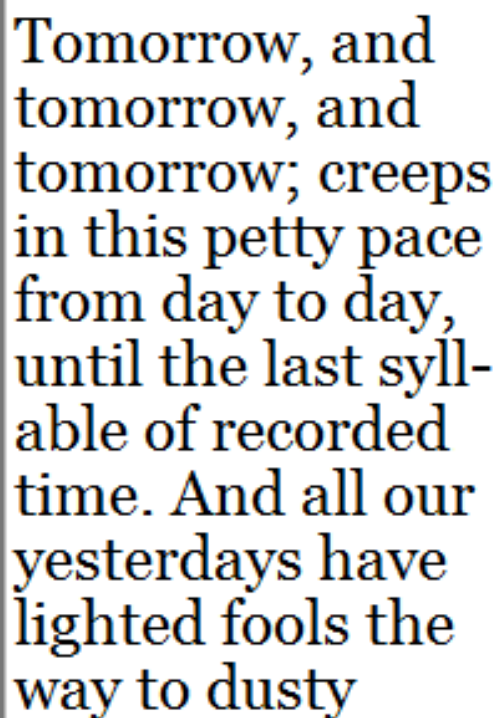
The layout of wrapped text is user agent dependent; thus, content developers need to be aware that there might be different results, particularly with regard to where line breaks occur. The user agent may choose to implement a simplistic text wrapping algorithm as described in this specification or invoke a sophisticated layout engine (e.g., an engine that supports the CSS or XSL-FO box models). The simplistic wrapping algorithm described in this specification is upwardly compatible with the XSL-FO box model.

The minimal layout facilities required for text in an area are described in [text in an area layout rules](#).

Example `textArea01` below contains a text string which will wrap into a rectangular area. Any text which does not fit will not be rendered.

Example: [textArea01.svg](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" viewBox="0 0 220 320"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Basic textflow</title>
  <textArea font-size="25" font-family="Georgia" x="10" y="10" width="200"
    height="300">Tomorrow, and tomorrow, and
    tomorrow; creeps in this petty pace from day to day, until the last syll&#xAD;able of recorded
    time. And all our yesterdays have lighted fools the way to dusty death.</textArea>
  <rect x="5" y="5" width="210" height="310" stroke-width="3" stroke="#777"/>
</svg>
```



Tomorrow, and
tomorrow, and
tomorrow; creeps
in this petty pace
from day to day,
until the last syll-
able of recorded
time. And all our
yesterdays have
lighted fools the
way to dusty

10.11.2 The 'textArea' element

Schema: textArea

```
<define name='textArea'>
  <element name='textArea'>
    <ref name='textArea.AT' />
    <zeroOrMore><ref name='svg.TextCommon.group' /></zeroOrMore>
  </element>
</define>

<define name='textArea.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.XY.attr' />
  <ref name='svg.Editable.attr' />
  <optional>
    <attribute name='width' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <ref name='Length.datatype' />
        <value>auto</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='height' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <ref name='Length.datatype' />
        <value>auto</value>
      </choice>
    </attribute>
  </optional>
</define>
```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of one corner of the rectangular region into which the text content will be placed. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

y = "[<coordinate>](#)"

The y-axis coordinate of one corner of the rectangular region into which the text content will be placed. If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: yes.

width = "[auto](#) | [<coordinate>](#)"

The width of the rectangular region into which the text content will be placed. A value of "auto" indicates that the width of the rectangular region is infinite. If the attribute is not specified, the effect is as if a value of "auto" were specified.

Animatable: yes.

height = "[auto](#) | [<coordinate>](#)"

The height of the rectangular region into which the text content will be placed. A value of "auto" indicates that the height of the rectangular region is infinite. If the attribute is not specified, the effect is as if a value of "auto" were specified.

Animatable: yes.

editable = "[none](#) | [simple](#)"

This attribute indicates whether the text can be edited. See the definition of the ['editable'](#) attribute.

Animatable: no.

focusable = "[true](#)" | "[false](#)" | "[auto](#)"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

10.11.3 The 'tbreak' element

The 'tbreak' element is an empty element that forcibly breaks the current line of text. Multiple consecutive tbreak elements have the same effect on rendering as a single tbreak.

Schema: tbreak

```
<define name='tbreak'>
  <element name='tbreak'>
    <ref name='tbreak.AT' />
    <empty />
  </element>
</define>

<define name='tbreak.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
</define>
```

No Attributes

10.11.4 The 'line-increment' property

The 'line-increment' property provides limited control over the size of each line in the block-progression-direction. This property applies to the ['textArea'](#) element. It is a proper subset of the CSS ['line-height' property](#) so that user agents which have a CSS box model engine can use that engine to provide text wrapping.

'line-increment'

<i>Value:</i>	auto <number> <u>inherit</u>
<i>Initial:</i>	auto
<i>Applies to:</i>	'textArea'
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	Specified value, except inherit

Values for the property have the following meaning:

auto

Subsequent lines are offset from the previous line by the maximum font-size for any glyphs drawn within that line.

<number>

Subsequent lines are offset from the previous line by this amount (in user units).

10.11.5 The 'display-align' property

The 'display-align' property specifies the alignment, in the block-progression-direction, of the text content of the ['textArea'](#) element.

'display-align'

<i>Value:</i>	auto before center after <u>inherit</u>
<i>Initial:</i>	auto
<i>Applies to:</i>	'textArea'
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes

Computed value: Specified value, except inherit

Values for the property have the following meaning:

- auto** For SVG, 'auto' is equivalent to 'before'.
- before** The before-edge of the first line is aligned with the [before-edge](#) of the first region.
- center** The lines are centered in the block-progression-direction.
- after** The after-edge of the last line is aligned with the [after-edge](#) of the last region.

Below is a recommended algorithm for implementing 'display-align'. Implementations relying on a different algorithm must exhibit the same behavior as this algorithm does.

1. A temporary delta variable, D, is initialized to the height of the ['textArea'](#) bounding box in the text progression direction (the distance between the coordinate of the shape closest to the [after-edge](#) and the coordinate of the shape closest to the [before-edge](#).) A starting point is chosen as the coordinate of the shape closest to the [before-edge](#).
2. The flowed text is laid out inside the ['textArea'](#).
3. At the completion of layout, the space remaining in the before direction, B, (the distance from the first line to the coordinate of the shape closest to the [before-edge](#)) is calculated.
4. The space remaining in the after direction, A (the distance between the last line and the coordinate of the shape closest to the [after-edge](#)) is calculated.
5. A measure of the difference from exact centering, C, is calculated as the absolute value of A - B.
6. If C is less than 1px in non-transformed user space for the ['textArea'](#) layout terminates.
7. If C is greater than D then layout terminates. (This will not happen on the first iteration, but may on subsequent iterations if text is flowed into irregular shapes).
8. The temporary delta variable, D, is set to C.
9. A new starting point is chosen, computed as $B + 0.5(A - B)$ and execution returns to step 2.

10.11.6 Text in an area layout rules

Text in an area layout is defined as a post processing step to the standard text layout model of SVG.

A conformant user agent can implement a simplistic layout algorithm which consists simply of inserting line breaks whenever the content explicitly specifies a line break with a ['tbreak'](#) element or when the current line cannot fit all of the remaining glyphs. Any lines of glyphs that do not completely fit within the region(s) are not rendered.

User agents should implement a line-breaking algorithm that supports at a minimum the features described below as a post processing step to SVG's standard text layout model.

1. The text is processed in logical order to determine line breaking opportunities between characters, according to [Unicode Standard Annex No. 14](#)
2. Text layout is performed as normal, on one infinitely long line, soft hyphens are included in the line. The result is a set of positioned Glyphs.
3. The first line is positioned such that its before edge is flush against the region's before edge relative to the block-progression-direction.
4. Glyphs represent a character or characters within a word. Each glyph is associated with the word that contains its respective characters. In cases where characters from multiple words contribute to the same glyph the words are merged and all the glyphs are treated as part of the earliest word in logical order.
5. The glyphs from a word are collapsed into Glyph Groups. A Glyph Group is comprised of all consecutive glyphs from the same word. In most cases each word generates one glyph group however in some cases the interaction between BIDl and special markup may cause glyphs from one word to have glyphs from other words embedded in it.
6. Each Glyph Group has two extents calculated: it's normal extent, and it's last in text area extent. It's normal extent is the sum of the advances of all glyphs in the group except soft hyphens. The normal extent is the extent used when a Glyph Group from a later word is in the same text area. The last in text area extent includes the advance of a trailing soft hyphens but does not include the advance of trailing whitespace or combining marks. The last in text region extent is used when this glyph group is from the last word (in logical order) in this text area. (If the entire line consists of a single word which is not breakable, the User Agent may choose to force a break in the line so that at least some text will appear for the given line.)
7. Words are added to the current line in logical order. All the Glyph Groups from a word must be in the same line and all the glyphs from a Glyph Group must be in the same ['textArea'](#).
8. If line-increment is a number, then each line will be sized in the block-progression-direction to the value of line-increment. If line-increment is "auto", then the maximum font-size for any glyph in the line will determine the size of the line in the

block-progression-direction. When a word is added the line increment may increase, it can never decrease from the first word. An increase in the line increment can only reduce the space available for text placement in the span. The span will have the maximum possible number of words. The position of the dominant baseline for a given line is determined by first computing the line-increment value for that line and then choosing a position for the dominant baseline using the position where the given baseline would appear for the font that will be used to render the first character and an assumed font-size equal to the line-increment value.

9. The Glyphs from the Glyph Groups are then collapsed into the text regions by placing the first selected glyph (in display order) at the start of the text area and each subsequent glyph at the location of the glyph following the preceding selected glyph (in display order).
10. The next word is selected and the next line location is determined. The next line is positioned such that its before edge is flush against the after edge of the previous line relative to the block-progression-direction. Goto Step 7.
11. Any lines which extend outside of the area(s) in the block-progression-direction are not rendered.

10.12 Editable Text Fields

SVG Tiny 1.2 allows text elements to be edited. Although simple text editing can be implemented directly in script, implementing an intuitive and well internationalized text input system which works on a variety of platforms is complex. Therefore, this functionality is provided by the SVG user agent, which has access to system text libraries. Content authors can build higher level widgets, such as form entry fields, on top of the editable text functionality.

10.12.1 The editable attribute

The [text](#) and [textArea](#) elements have an editable attribute which specifies whether the contents of the elements can be edited in place.

Schema: editable

```
<define name='svg.Editable.attr' combine='interleave'>
  <optional>
    <attribute name='editable' a:defaultValue='false' svg:animatable='false' svg:inheritable='false'>
      <ref name='Boolean.datatype' />
    </attribute>
  </optional>
</define>
```

editable = "none" | "simple"

If set to "none" the contents of the [text](#) or [textArea](#) elements must not be editable in place through the user agent. If set to "simple", the user agent must provide a way for the user to edit the content of the [text](#) or [textArea](#) elements and all contained subelements which are not hidden (with visibility="hidden") or disabled (through the switch element or display="none"). The user agent must also, (if a clipboard is supported by the platform), provide a way to cut or copy the selected text from the element to the clipboard, and to paste text from the clipboard into the element. If no value is given for this attribute, the default value is "false". Whenever the [editable](#) attribute is set to `simple`, the [focusable](#) attribute is considered to be set to `simple`, irrespective of what the actual value is.

Animatable: Yes.

SVG Tiny 1.2 user agents should allow for the inline WYSIWYG editing of text. However, editing with a modal editing dialog is an alternate possibility, and may be the only option on some platforms. The current editing position should be indicated, for example with a caret. SVG Tiny 1.2 user agents must also support system functions such as copy/paste and drag/drop if they are available to applications on the platform.

To start editing, the current animated value of the editable attribute must be "simple", the [text](#) or [textArea](#) elements must have focus, and it must then be activated. When editing text in a text field, all [DOM3 Events](#) [[DOM3Events](#)] text and key events are dispatched to the user agent, which processes the events for proper handling of text entry.

If a text or textArea element is editable, then the SVG user agent must not normalize white-space in user input when changing the tree according to the input.

For WYSIWYG editing the following functionality must be made available:

- movement to the next/previous character (in logical order), for example with Left/Right arrows
- in textArea elements, movement to the next/previous line, for example with the Down/Up keys
- movement to the beginning of the line, for example with the Home key
- movement to the end of the line, for example with the End key

- copy/cut/paste, if a clipboard is supported, for example with Copy and Paste keys

The functionality should use the normal key bindings that are used for those tasks on the given platform. For devices without keyboard access, the equivalent system input methods should be used wherever possible to provide the functionality described above.

When doing WYSIWYG editing in place, the content of the DOM nodes that are being edited should be live at all times and reflect the current state of the edited string as it being edited. When using a modal editing dialog, the content of the DOM nodes will only change once the user commits the edit (for example, by using an Enter key or clicking an 'OK' button, or whatever the platform normally does), firing a single DOM 3 text event.

If an Input Method Editor (IME) is used (for example, to input Kanji text, or to input Latin text using number keys on mobile phones), the text events returned by DOM 3 correspond to the actual text entered (eg the Kanji character, or the Latin character) and not to the keyboard or mouse gestures needed to produce it (such as the sequence of kana characters, or the number of sequential presses of a numeric key).

While text is being edited, the user agent SHOULD always make the caret visible to the user as it is moved around the edited text (either due to typing more characters or to moving it within existing text). The precise behavior in which this functionality is supported depends on the user agent.

The behavior of edited text while the caret is placed inside a ligature is implementation dependent. User agents are however encouraged to take into account the notions captured in [Character Model for the World Wide Web 1.0: Fundamentals, Section 6.1: String concepts](#).

10.13 Text selection and clipboard operations

If [SVG viewers](#) support text selection and copy/paste operations then they must support:

- user selection of text strings in SVG content
- the ability to copy selected text strings to the system clipboard

A text selection operation starts when all of the following occur:

- the user positions the pointing device or caret over a glyph that has been rendered as part of a text or textArea element, initiates a *select* operation (e.g., pressing the standard system mouse button for select operations) and then moves the current text position while continuing the *select* operation (e.g., continuing to press the standard system mouse button for select operations);
- no other visible graphics element has been painted above the glyph at the point at which the pointing device was clicked.

As the text selection operation proceeds (e.g., the user continues to press the given mouse button), all associated events with other graphics elements are ignored (i.e., the text selection operation is modal) and the SVG user agent shall dynamically indicate which characters are selected by an appropriate highlighting technique, such as redrawing the selected glyphs with inverse colors. As the current text position is moved during the text selection process, the end glyph for the text selection operation is the glyph within the same **'text'** element whose glyph cell is closest to the pointer. All characters within the **'text'** element whose position within the **'text'** element is between the start of selection and end of selection shall be highlighted, regardless of position on the canvas and regardless of any graphics elements that might be above the end of selection point.

Once the text selection operation ends (e.g., the user releases the given mouse button), the selected text will stay highlighted until an event occurs which cancels text selection, such as a pointer device activation event (e.g., pressing a mouse button).

Detailed rules for determining which characters to highlight during a text selection operation are provided in [Text selection implementation notes](#).

For systems which have system clipboards, the SVG user agent is required to provide a user interface for initiating a copy of the currently selected text to the system clipboard. It is sufficient for the SVG user agent to post the selected text string in the system's appropriate clipboard format for plain text, but it is preferable if the SVG user agent also posts a rich text alternative which captures the various [font properties](#) associated with the given text string.

For bidirectional text, the user agent must support text selection in logical order, which will result in discontinuous highlighting of glyphs due to the bidirectional reordering of characters. User agents can also optionally provide an alternative ability to select bidirectional text in visual rendering order (i.e., after [bidirectional](#) text layout algorithms have been applied), with the result that selected character data might be discontinuous logically. In this case, if the user requests that bidirectional text be copied to the clipboard, then the user agent is required to make appropriate adjustments to copy only the visually selected characters to the clipboard.

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

If the text of an editable element is edited, and the element has child elements, the contents of the edited element must first be *flattened*. Flattening must have the same effect as the following procedure: extract the text content of the edited element and its children, as defined in the [textContent attribute](#) of DOM Level 3 Core [\[DOM3\]](#), and assign it to the edited element's textContent attribute.

In ECMAScript:

```
myTextEl.textContent = myTextEl.textContent;
```

In Java:

```
myTextEl.setTextContent(myTextEl.getTextContent());
```

10.14 Text Module

The Text Module contains the following elements:

- [text](#)
- [tspan](#)
- [textArea](#)
- [tbreak](#)

[RNG] [\[Feature String\]](#)

11 Painting: Filling, Stroking, Colors and Paint Servers

Contents

- 11.1 [Introduction](#)
- 11.2 [Specifying paint](#)
- 11.3 [Fill Properties](#)
- 11.4 [Stroke Properties](#)
- 11.5 [Non-Scaling Stroke](#)
- 11.6 [Simple alpha compositing](#)
 - 11.6.1 [Compositing the currentColor value](#)
- 11.7 [The 'viewport-fill' Property](#)
- 11.8 [The 'viewport-fill-opacity' Property](#)
- 11.9 [Controlling visibility](#)
- 11.10 [Rendering hints](#)
 - 11.10.1 [The 'color-rendering' property](#)
 - 11.10.2 [The 'shape-rendering' property](#)
 - 11.10.3 [The 'text-rendering' property](#)
 - 11.10.4 [The 'image-rendering' property](#)
- 11.11 [Inheritance of painting properties](#)
- 11.12 [Object and group opacity: the 'opacity' property](#)
- 11.13 [Color](#)
 - 11.13.1 [Syntax for color values](#)
 - 11.13.2 [HTML Color keywords](#)
 - 11.13.3 [System Paint](#)
 - 11.13.4 [The solidColor Element](#)
 - 11.13.5 [The SVG 'color' property](#)
- 11.14 [Paint Servers](#)
- 11.15 [Gradients](#)
 - 11.15.1 [Linear gradients](#)
 - 11.15.2 [Radial gradients](#)
 - 11.15.3 [Gradient stops](#)
- 11.16 [Paint Attribute Module](#)
- 11.17 [Opacity Attribute Module](#)
- 11.18 [Graphics Attribute Module](#)
- 11.19 [Gradient Module](#)
- 11.20 [Solid Color Module](#)

11.1 Introduction

'[path](#)' elements, '[text](#)' elements and [basic shapes](#) can be **filled** (which means painting the interior of the object) and **stroked** (which means painting along the outline of the object). Filling and stroking both can be thought of in more general terms as **painting** operations.

With SVG, you can paint (i.e., fill or stroke) with:

- a single solid color, possibly with some level of transparency
- a gradient (linear or radial)

SVG uses the general notion of a **paint server**. Paint servers are specified using a [IRI reference](#) on a '[fill](#)' or '[stroke](#)' property (the [IRI reference](#) must be local as described in the [IRI Reference Section](#)). Gradients and solid colors are just specific types of paint servers.

11.2 Specifying paint

Properties '[fill](#)' and '[stroke](#)' take on a value of type `<paint>`, which is specified as follows:

<paint>:none |

currentColor |

<color> |

<iri> |

inherit

none

Indicates that no paint shall be applied.

currentColor

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

<color>

<color> is the explicit color (in the sRGB [SRGB](#) color space) that must be used to paint the current object. All of the syntax alternatives for **<color>** defined in [\[CSS2-color-types \]](#) including the list of [recognized color keyword names](#), must be supported.

<iri>

<iri> specifies a [paint server](#) such as a gradient. The **<iri>** provides the ID of the paint server (e.g., a [gradient](#) or [solid color](#)) that shall be used to paint the current object. The [IRI reference](#) must be local as described in the [IRI Reference Section](#). If the [IRI reference](#) is not valid (e.g., it points to an object that doesn't exist or the object is not a valid paint server), then it **MUST** be considered unsupported and processed as if it hadn't been specified.

11.3 Fill Properties

'fill'

Value: **<paint> | inherit** (See [Specifying paint](#))

Initial: black

Applies to: [shapes](#) and [text content elements](#)

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

Computed value: Specified value, except inherit

The 'fill' property specifies that the interior of the given graphical element must be painted. The area to be painted shall consist of any areas inside the outline of the shape. To determine the inside of the shape, all subpaths must be considered, and the interior shall be determined according to the rules associated with the current value of the 'fill-rule' property. The zero-width geometric outline of a shape must be included in the area to be painted.

Open subpaths must be filled by performing the fill operation as if an additional "closepath" command were added to the path to connect the last point of the subpath with the first point of the subpath. Thus, fill operations apply to both open subpaths within 'path' elements (i.e., subpaths without a closepath command) and 'polyline' elements.

'fill-rule'

Value: nonzero | evenodd | **inherit**

Initial: nonzero

Applies to: [shapes](#) and [text content elements](#)

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

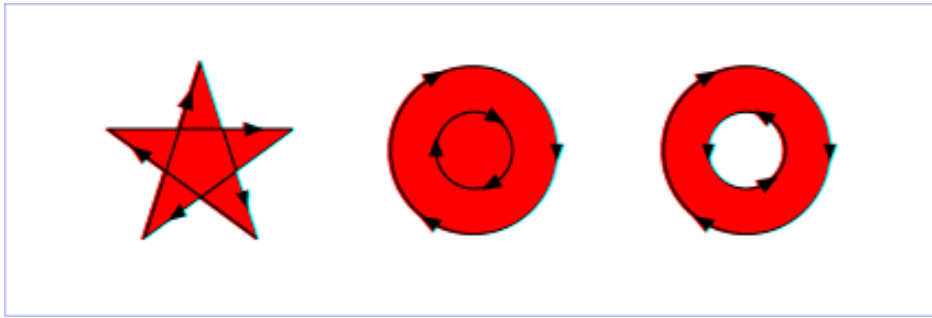
Computed value: Specified value, except inherit

The 'fill-rule' property indicates the algorithm which must be used to determine what parts of the canvas are included inside the shape. For a simple, non-intersecting path, it is intuitively clear what region lies "inside"; however, for a more complex path, such as a path that intersects itself or where one subpath encloses another, the interpretation of "inside" is not so obvious.

The 'fill-rule' property provides two options for how the inside of a shape is determined:

nonzero

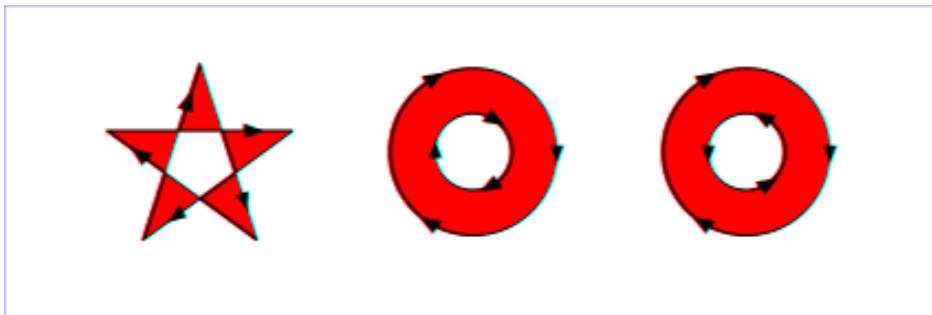
The following algorithm, or any other that gives the same result, must be used to determine the "insideness" of a point on the canvas. Draw a ray from the point to infinity in any direction and then examine the places where a segment of the shape crosses the ray. Starting with a count of zero, add one each time a path segment crosses the ray from left to right and subtract one each time a path segment crosses the ray from right to left. After counting the crossings, if the result is zero then the point is *outside* the path. Otherwise, it is *inside*. The following drawing illustrates the **nonzero** rule:



[View this example as SVG \(SVG-enabled browsers only\)](#)

evenodd

The following algorithm, or any other that gives the same result, must be used to determine the "insideness" of a point on the canvas. Draw a ray from the point to infinity in any direction and counting the number of path segments from the given shape that the ray crosses. If this number is odd, the point is inside; if even, the point is outside. The following drawing illustrates the **evenodd** rule:



[View this example as SVG \(SVG-enabled browsers only\)](#)

(Note: the above explanations do not specify what to do if a path segment coincides with or is tangent to the ray. Since any ray will do, one may simply choose a different ray that does not have such problem intersections.)

'fill-opacity'

Value: <opacity-value> | inherit
Initial: 1
Applies to: [shapes](#) and [text content elements](#)
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes
Computed value: Specified value, except inherit

'**fill-opacity**' specifies the opacity of the painting operation which shall be used to paint the interior the current object. (See [Painting shapes and text.](#))

<opacity-value>

The opacity of the painting operation that must be used to fill the current object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) shall be clamped to this range.

Related property: '[stroke-opacity](#)'.

11.4 Stroke Properties

The following are the properties which affect how an element is stroked.

In all cases, strokes which are affected by directionality, such as those having dash patterns, must be rendered such that the stroke operation starts at the same point at which the graphics element starts. In particular, for **'path'** elements, the start of the path is the first point of the initial "moveto" command.

For strokes, such as dash patterns whose computations are dependent on progress along the outline of the graphics element, distance calculations must use the SVG user agent's standard [Distance along a path](#) algorithms.

When stroking is performed using a complex paint server, such as a gradient, the stroke operation must be identical to the result that would have occurred if the geometric shape defined by the geometry of the current graphics element and its associated stroking properties were converted to an equivalent **'path'** element and then filled using the given paint server.

'stroke'

Value: <paint> | inherit (See [Specifying paint](#))
Initial: none
Applies to: [shapes](#) and [text content elements](#)
Inherited: yes
Percentages: N/A
Media: visual
[Animatable](#): yes
Computed value: Specified value, except inherit

The **'stroke'** property shall paint along the outline of the given graphical element.

A subpath (see [Paths](#)) consisting of a single [moveto](#) shall not be stroked. A subpath consisting of a [moveto](#) and [lineto](#) to the same exact location or a subpath consisting of a [moveto](#) and a [closepath](#) shall be stroked only if the **'stroke-linecap'** property is set to "round", producing a circle centered at the given point.

'stroke-width'

Value: <length> | inherit
Initial: 1
Applies to: [shapes](#) and [text content elements](#)
Inherited: yes
Percentages: N/A
Media: visual
[Animatable](#): yes
Computed value: Specified value, except inherit

<length>

The width of the stroke which shall be used on the current object.

No stroke shall be painted for a zero value. A negative value is unsupported and MUST be treated as if the stroke had not been specified.

'stroke-linecap'

Value: butt | round | square | inherit
Initial: butt
Applies to: [shapes](#) and [text content elements](#)
Inherited: yes
Percentages: N/A
Media: visual
[Animatable](#): yes
Computed value: Specified value, except inherit

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

butt

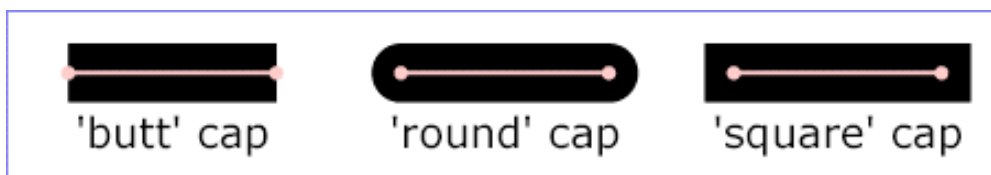
The shape drawn at the end of open subpaths shall be as per the drawing below.

round

The shape drawn at the end of open subpaths shall be as per the drawing below.

square

The shape drawn at the end of open subpaths shall be as per the drawing below.



[View this example as SVG \(SVG-enabled browsers only\)](#)

'stroke-linejoin'

Value: miter | round | bevel | inherit

Initial: miter

Applies to: [shapes](#) and [text content elements](#)

Inherited: yes

Percentages: N/A

Media: visual

[Animatable:](#) yes

Computed value: Specified value, except inherit

'stroke-linejoin' specifies the shape which shall be used at the corners of paths or basic shapes when they are stroked.

miter

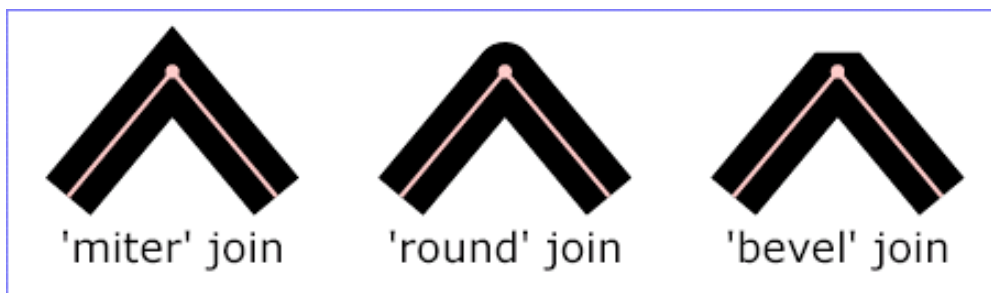
The shape drawn at the corner of paths or basic shapes shall be as per the drawing below.

round

The shape drawn at the corner of paths or basic shapes shall be as per the drawing below.

bevel

The shape drawn at the corner of paths or basic shapes shall be as per the drawing below.



[View this example as SVG \(SVG-enabled browsers only\)](#)

'stroke-miterlimit'

Value: <miterlimit> | inherit

Initial: 4

Applies to: [shapes](#) and [text content elements](#)

Inherited: yes

Percentages: N/A

Media: visual

[Animatable:](#) yes

Computed value: Specified value, except inherit

When two line segments meet at a sharp angle and **miter** joins have been specified for 'stroke-linejoin', it is possible for the miter to extend far beyond

the thickness of the line stroking the path. The **'stroke-miterlimit'** imposes a limit on the ratio of the miter length to the **'stroke-width'**. When the limit is exceeded, the join must be converted from a miter to a bevel.

<miterlimit>

The limit on the ratio of the miter length to the **'stroke-width'**. The value of **<miterlimit>** must be a number greater than or equal to 1. Any other value shall be treated as unsupported and processed as if the property had not been specified.

The ratio of miter length (distance between the outer tip and the inner corner of the miter) to **'stroke-width'** is directly related to the angle (theta) between the segments in user space by the formula:

$$\text{miterLength} / \text{stroke-width} = 1 / \sin (\text{theta} / 2)$$

For example, a miter limit of 1.414 converts miters to bevels for theta less than 90 degrees, a limit of 4.0 converts them for theta less than approximately 29 degrees, and a limit of 10.0 converts them for theta less than approximately 11.5 degrees.

'stroke-dasharray'

Value: none | **<dasharray>** | inherit
Initial: none
Applies to: [shapes](#) and [text content elements](#)
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes ([non-additive](#))
Computed value: Specified value, except inherit

'stroke-dasharray' specifies the pattern of dashes and gaps that shall be used to stroke paths. **<dasharray>** contains a list of comma-separated (with optional white space) **<length>**s that specify the lengths of alternating dashes and gaps that must be used. If an odd number of values is provided, then the list of values shall be 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 shall be used. If stroked, the line must be drawn solid.

<dasharray>

A list of comma-separated **<length>**'s (with optional white space). A negative **<length>** value shall be treated as unsupported and processed as if the property had not been specified. If the sum of the **<length>**'s is zero, then the stroke shall be rendered as if a value of **none** were specified.

'stroke-dashoffset'

Value: [<length>](#) | inherit
Initial: 0
Applies to: [shapes](#) and [text content elements](#)
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes
Computed value: Specified value, except inherit

'stroke-dashoffset' specifies the distance into the dash pattern that must be used to start the dash.

<length>

Values can be negative.

'stroke-opacity'

Value: **<opacity-value>** | inherit
Initial: 1
Applies to: [shapes](#) and [text content elements](#)

Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes
Computed value: Specified value, except inherit

'stroke-opacity' specifies the opacity of the painting operation used to stroke the current object. (See [Painting shapes and text.](#))

<opacity-value>

The opacity of the painting operation which shall be used to stroke the current object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) must be clamped to this range. (See [Clamping values which are restricted to a particular range.](#))

Related property: '[fill-opacity](#)'.

11.5 Non-Scaling Stroke

Sometimes it is of interest to let the outline of an object keep its original width no matter which transforms are applied to it. E.g. in a map with a 2px wide line representing roads it is of interest to keep the roads 2px wide even when the user zooms into the map.

To achieve this SVG1.2 introduces the '[vector-effect](#)' property. Future versions of the SVG language will allow for more powerful vector effects through this property but this version restricts it to the non-scaling stroke.

'vector-effect'

Value: non-scaling-stroke | none | inherit
Initial: none
Applies to: [graphics elements](#)
Inherited: no
Percentages: N/A
Media: visual
Animatable: yes
Computed value: Specified value, except inherit

The value 'none' specifies that no vector effect shall be applied, i.e. the default rendering behaviour from SVG 1.1 is used which is to first fill the geometry of a shape with a specified paint, then stroke the outline with a specified paint.

The value 'non-scaling-stroke' modifies the way an object is stroked. Normally stroking involves calculating stroke outline of the shape's path in current user space and filling that outline with the stroke paint (color or gradient). With non-scaling-stroke vector effect, stroke outline shall be calculated in the "host" coordinate space instead of user coordinate space. More precisely: a user agent establishes a host coordinate space which in SVG 1.2 is always the same as "screen coordinate space". The stroke outline is calculated in the following manner: first, the shape's path is transformed into the host coordinate space. Stroke outline is calculated in the host coordinate space. Resulting outline is transformed back to the user coordinate space. (Stroke outline is always filled with stroke paint in the current user space). The resulting visual effect of this modification is that stroke width is not dependant on the transformations of the element (including non-uniform scaling and shear transformations) and zoom level.

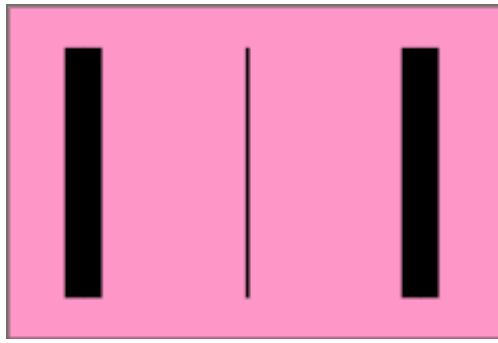
Note: Future versions of SVG may allow ways to control host coordinate system.

Below is an example of a non-scaling-stroke '[vector-effect](#)'.

Example: [non-scaling-stroke.svg](#)

```
<?xml version="1.0"?>
<svg width="6cm" height="4cm" viewBox="0 0 600 400" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg" viewport-fill="rgb(255,150,200)">
  <desc>Example non-scaling stroke</desc>
  <rect x="1" y="1" width="598" height="398" fill="none" stroke="black" />

  <g transform="scale(9,1)">
    <line stroke="black" stroke-width="5" x1="10" y1="50" x2="10" y2="350"/>
    <line vector-effect="non-scaling-stroke" stroke="black" stroke-width="5"
      x1="32" y1="50" x2="32" y2="350"/>
    <line vector-effect="none" stroke="black" stroke-width="5"
      x1="55" y1="50" x2="55" y2="350"/>
  </g>
</svg>
```

11.6 Simple alpha compositing

Graphics elements are blended into the elements already rendered on the canvas using simple alpha compositing, in which the resulting color and opacity at any given pixel on the canvas must be the result of the following formulas (all color values use premultiplied alpha):

```

Er, Eg, Eb    - Element color value
Ea           - Element alpha value
Cr, Cg, Cb   - Canvas color value (before blending)
Ca          - Canvas alpha value (before blending)
Cr', Cg', Cb' - Canvas color value (after blending)
Ca'         - Canvas alpha value (after blending)
Ca' = 1 - (1 - Ea) * (1 - Ca)
Cr' = (1 - Ea) * Cr + Er
Cg' = (1 - Ea) * Cg + Eg
Cb' = (1 - Ea) * Cb + Eb
  
```

The following rendering properties, which provide information about the color space in which to perform the compositing operations, apply to compositing operation:

- 'color-rendering'

11.6.1 Compositing the currentColor value

The **currentColor** property may be assigned a color value that has an opacity component. This opacity value is used in the rendering operation using the alpha compositing method described above. That is, the opacity value in **currentColor** is used when compositing the color into a paint server (which may have its own values for opacity).

11.7 The 'viewport-fill' Property

SVG enables the author to specify a solid color which will be used to fill the **viewport** of any element that creates a **viewport**, such as the **svg** element.

The **'viewport-fill'** property specifies the color which shall be used to fill the **viewport** created by a particular element. It must cause the entire canvas of the element that it applies to to be filled with the specified solid color. That canvas may then be clipped by that element's **viewBox**.

'viewport-fill'

Value: [<color>](#) | [inherit](#)
Initial: none
Applies to: [viewport-creating elements](#)
Inherited: no
Percentages: N/A
Media: visual
Animatable: yes
Computed value: Specified value, except inherit

Below is an example of **'viewport-fill'**.

Example: [11_02.svg](#)

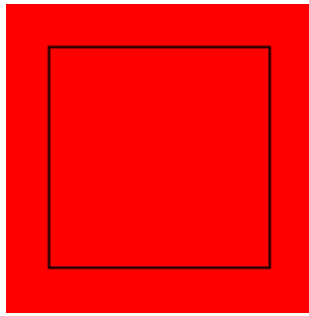
```

<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
  viewport-fill="red" baseProfile="tiny">
  <desc>
  Everything here has a red background.
  The rectangle is not filled, so the red background will show through
  
```

```

</desc>
<rect x="20" y="20" width="100" height="100" fill="none" stroke="black"/>
</svg>

```



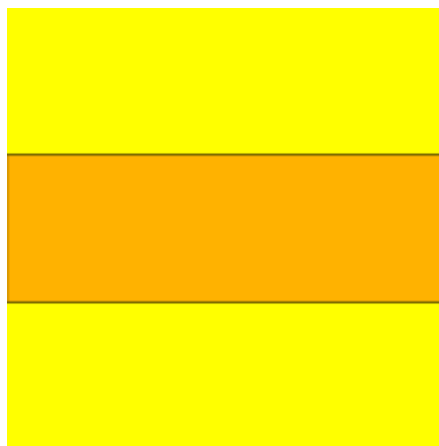
Here is a slightly more complex example. The [viewBox](#) gives a coordinate system 300 units wide and 100 units high. The rendering shows what happens when this is displayed inside a square [viewport](#).

Example: [11_03.svg](#)

```

<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
  viewport-fill="yellow" baseProfile="tiny" viewBox="0 0 300 100">
  <desc>
    The viewport has a yellow background.
    The rectangle is filled and covers the viewport, so the yellow
    background will only show through in the "leftovers" if the
    aspect ratio of the viewport differs from that of the viewBox.
  </desc>
  <rect x="0" y="0" width="300" height="100" fill="red" fill-opacity="0.3" stroke="black"/>
</svg>

```



The filling of the [viewport](#) is the first operation in the rendering chain of an element. Therefore:

- The [viewport](#) fill operation happens before filling and stroking.
- The [viewport](#) fill operation occurs before compositing, and thus is part of the input to the compositing operations.
- The [viewport](#) fill operation renders into the element's conceptual offscreen buffer, and thus **opacity** applies as usual.
- Viewport fill is not affected by the 'fill' or 'fill-opacity' properties.

11.8 The 'viewport-fill-opacity' Property

The 'viewport-fill-opacity' property specifies the opacity of the 'viewport-fill' that shall be used for a particular element.

'viewport-fill-opacity'

<i>Value:</i>	<opacity-value> <u>inherit</u>
<i>Initial:</i>	1.0
<i>Applies to:</i>	viewport-creating elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

Animatable: yes
Computed value: Specified value, except inherit

<opacity-value>

The opacity of the painting operation which shall be used to fill the **viewport**. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) must be clamped to this range. (See [Clamping values which are restricted to a particular range.](#))

11.9 Controlling visibility

SVG uses two properties, **'display'** and **'visibility'**, to control the visibility of graphical elements or (in the case of the **'display'** property) container elements. Neither of these two properties affects the objects existence in the DOM, i.e. no matter what value of these properties the object still remains in the DOM.

The differences between the two properties are as follows:

- When applied to a container element, setting **'display'** to **none** causes the container and all of its children to be invisible; thus, it acts on groups of elements as a group. **'visibility'**, however, only applies to individual graphics elements. Setting **'visibility'** to **hidden** on a **'g'** will make its children invisible as long as the children do not specify their own **'visibility'** properties as **visible**.
- When the **'display'** property is set to **none**, then the given element does not become part of the [rendering tree](#). With **'visibility'** set to **hidden**, however, processing occurs as if the element were part of the [rendering tree](#) and still taking up space, but not actually rendered onto the canvas. This distinction has implications for the **'tspan'** element, [event processing](#), and for [bounding box calculations](#). If **'display'** is set to **none** on a **'tspan'** then the text string is ignored for the purposes of text layout; however, if **'visibility'** is set to **hidden**, the text string is used for text layout (i.e., it takes up space) even though it is not rendered on the canvas. Regarding events, if **'display'** is set to **none**, the element receives no events; however, if **'visibility'** is set to **hidden**, the element might still receive events, depending on the value of property **'pointer-events'**. The geometry of a graphics element with **'display'** set to **none** is not included in [bounding box](#) calculations; however, even if **'visibility'** is to **hidden**, the geometry of the graphics element still contributes to bounding box calculations.

'display'

Value: inline | block | list-item |
run-in | compact | marker |
table | inline-table | table-row-group | table-header-group |
table-footer-group | table-row | table-column-group | table-column |
table-cell | table-caption | none | inherit

Initial: inline

Applies to: **'svg'**, **'g'**, **'switch'**, **'a'**, **'foreignObject'**, [graphics elements](#) (including the **'text'** element) and text sub-elements (i.e., **'tspan'** and **'a'**)

Inherited: no

Percentages: N/A

Media: all

Animatable: yes

Computed value: Specified value, except inherit

A value of **display: none** indicates that the given element and its children shall not be rendered directly (i.e., those elements are not present in the [rendering tree](#)). Any computed value other than **none** indicates that the given element shall be rendered by the SVG user agent.

The **'display'** property only affects the direct rendering of a given element, whereas it does not prevent elements from being referenced by other elements.

Elements with **display: none** do not take up space in text layout operations, do not receive events, and do not contribute to [bounding box](#) calculations.

Except for any additional information provided in this specification, the normative definition is the [CSS2 definition of the 'display' property](#).

'visibility'

Value: visible | hidden | collapse | inherit

Initial: visible

Applies to: [graphics elements](#) (including the **'text'** element) and text sub-elements (i.e., **'tspan'** and **'a'**)

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

Computed value: Specified value, except inherit

visible

The current graphics element shall be visible.

hidden or collapse

The current graphics element shall be invisible (i.e., nothing is painted on the canvas).

Note that if the **'visibility'** property is set to **hidden** on a **'tspan'** element, then the text is invisible but shall still takes up space in text layout calculations.

Depending on the value of property **'pointer-events'**, graphics elements which have their **'visibility'** property set to **hidden** still might receive events.

Except for any additional information provided in this specification, the normative definition is the [CSS2 definition of the 'visibility' property](#).

11.10 Rendering hints

11.10.1 The **'color-rendering'** property

The creator of SVG content might want to provide a hint to the implementation about how to make speed vs. quality tradeoffs as it performs color interpolation and compositing. The **'color-rendering'** property provides a hint to the SVG user agent about how to optimize its color interpolation and compositing operations.

'color-rendering'

Value: auto | optimizeSpeed | optimizeQuality | inherit

Initial: auto

Applies to: [container elements](#), [graphics elements](#) and **'animateColor'**

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

Computed value: Specified value, except inherit

auto

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

optimizeSpeed

Indicates that the user agent shall emphasize rendering speed over quality. For RGB display devices, this option will sometimes cause the user agent to perform color interpolation and compositing in the device RGB color space.

optimizeQuality

Indicates that the user agent shall emphasize quality over rendering speed.

11.10.2 The **'shape-rendering'** property

The creator of SVG content might want to provide a hint to the implementation about what tradeoffs to make as it renders vector graphics elements such as [path](#) elements and [basic shapes](#) such as circles and rectangles. The **'shape-rendering'** property provides these hints.

'shape-rendering'

Value: auto | optimizeSpeed | crispEdges |

geometricPrecision | inherit

Initial: auto

Applies to: [shapes](#)

Inherited: yes

Percentages: N/A

Media: visual
Animatable: yes
Computed value: Specified value, except inherit

auto

Indicates that the user agent shall make appropriate tradeoffs to balance speed, crisp edges and geometric precision, but with geometric precision given more importance than speed and crisp edges.

optimizeSpeed

Indicates that the user agent shall emphasize rendering speed over geometric precision and crisp edges. This option will sometimes cause the user agent to turn off shape anti-aliasing.

crispEdges

Indicates that the user agent shall attempt to emphasize the contrast between clean edges of artwork over rendering speed and geometric precision. To achieve crisp edges, the user agent might turn off anti-aliasing for all lines and curves or possibly just for straight lines which are close to vertical or horizontal. Also, the user agent might adjust line positions and line widths to align edges with device pixels.

geometricPrecision

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

11.10.3 The 'text-rendering' property

The creator of SVG content might want to provide a hint to the implementation about what tradeoffs to make as it renders text. The 'text-rendering' property provides these hints.

'text-rendering'

Value: auto | optimizeSpeed | optimizeLegibility |

geometricPrecision | inherit

Initial: auto

Applies to: **'text'** and **'textArea'** elements

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

Computed value: Specified value, except inherit

auto

Indicates that the user agent shall make appropriate tradeoffs to balance speed, legibility and geometric precision, but with legibility given more importance than speed and geometric precision.

optimizeSpeed

Indicates that the user agent shall emphasize rendering speed over legibility and geometric precision. This option will sometimes cause the user agent to turn off text anti-aliasing.

optimizeLegibility

Indicates that the user agent shall emphasize legibility over rendering speed and geometric precision. The user agent will often choose whether to apply anti-aliasing techniques, built-in font hinting or both to produce the most legible text.

geometricPrecision

Indicates that the user agent shall emphasize geometric precision over legibility and rendering speed. This option will usually cause the user agent to suspend the use of hinting so that glyph outlines are drawn with comparable geometric precision to the rendering of path data.

11.10.4 The 'image-rendering' property

The creator of SVG content might want to provide a hint to the implementation about how to make speed vs. quality tradeoffs as it performs image processing. The 'image-rendering' property provides a hint to the SVG user agent about how to optimize its image rendering.:

'image-rendering'

Value: auto | optimizeSpeed | optimizeQuality |

inherit

Initial: auto

Applies to: images

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

Computed value: Specified value, except inherit

auto

Indicates that the user agent shall make appropriate tradeoffs to balance speed and quality, but quality shall be given more importance than speed. The user agent shall employ a resampling algorithm at least as good as nearest neighbor resampling, but bilinear resampling is strongly preferred. For [Conforming High-Quality SVG Viewers](#), the user agent shall employ a resampling algorithm at least as good as bilinear resampling.

optimizeQuality

Indicates that the user agent shall emphasize quality over rendering speed. The user agent shall employ a resampling algorithm at least as good as bilinear resampling.

optimizeSpeed

Indicates that the user agent shall emphasize rendering speed over quality. The user agent should use a resampling algorithm which achieves the goal of fast rendering, with the requirement that the resampling algorithm shall be at least as good as nearest neighbor resampling. If performance goals can be achieved with higher quality algorithms, then the user agent should use the higher quality algorithms instead of nearest neighbor resampling.

In all cases, resampling must be done in a truecolor (e.g., 24-bit) color space even if the original data and/or the target device is indexed color.

11.11 Inheritance of painting properties

The values of any of the painting properties described in this chapter can be inherited from a given object's parent. Painting, however, is always done on each [graphics element](#) individually, never at the [container element](#) (e.g., a ['g'](#)) level. Thus, for the following SVG, even though the gradient fill is specified on the ['g'](#), the gradient is simply inherited through the ['g'](#) element down into each rectangle, each of which is rendered such that its interior is painted with the gradient.

Example Inheritance

Example: 11_01.svg

```
<?xml version="1.0"?>
<svg width="7cm" height="2cm" viewBox="0 0 700 200"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Gradients apply to leaf nodes
  </desc>
  <g>
    <defs>
      <linearGradient xml:id="MyGradient" gradientUnits="objectBoundingBox">
        <stop offset="0" stop-color="#F60" />
        <stop offset="1" stop-color="#FF6" />
      </linearGradient>
    </defs>
    <rect x="1" y="1" width="698" height="198"
      fill="none" stroke="blue" stroke-width="2" />
    <g fill="url(#MyGradient)" >
      <rect x="100" y="50" width="200" height="100"/>
      <rect x="400" y="50" width="200" height="100"/>
    </g>
  </g>
</svg>
```



Any painting properties defined in terms of the [object's bounding box](#) use the bounding box of the [graphics element](#) to which the operation applies. Note that [text elements](#) are defined such that any painting operations defined in terms of the [object's bounding box](#) use the bounding box of the entire ['text'](#) element. (See the discussion of [object bounding box units and text elements](#).)

11.12 Object and group opacity: the 'opacity' property

There are several opacity properties within SVG:

- [Solid color opacity](#)
- [Fill opacity](#)
- [Stroke opacity](#)
- [Gradient stop opacity](#)
- [Viewport fill 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.

Object/group opacity can, if applied to container elements, be a resource intensive operation. Therefore this version of SVG restricts this property to only apply to the [image](#) element.

'opacity'

<i>Value:</i>	<opacity-value> inherit
<i>Initial:</i>	1
<i>Applies to:</i>	image element
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	Specified value, except inherit

<opacity-value>

The uniform opacity setting must be applied across an entire object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) shall be clamped to this range. (See [Clamping values which are restricted to a particular range.](#))

11.13 Color

In SVG Tiny 1.2, all colors are specified in the sRGB color space (see [SRGB](#)). SVG Tiny 1.2 user agents are not required to, but may, support color management. However, SVG Tiny 1.2 user agents should apply gamma correction if the response curve of the display system differs from that of sRGB

11.13.1 Syntax for color values

Five syntactical forms are specified for SVG Tiny 1.2 All of them must be supported in a conforming SVG Interpreter:

Three digit hex - #rgb

Each hexadecimal digit, in the range 0 to F, represents one sRGB color component in the order red, green and blue. The digits A to F may be in either uppercase or lowercase. The value of the color component is obtained by replicating digits, so 0 become 00, 1 becomes 11, F becomes FF. This compact syntactical form can represent only 4096 colors. Examples: #000 (i.e.black) #fff (i.e.white) #6CF (i.e. #66CCFF, rgb(102, 204, 255)).

Six digit hex - #rrggbb

Each pair of hexadecimal digits, in the range 0 to F, represents one sRGB color component in the order red, green and blue. The digits A to F may be in either uppercase or lowercase. This syntactical form, originally introduced by HTML, can represent 16777216 colors. Examples: #9400D3 (i.e. a dark violet), #FFD700 (ie a golden color).

Integer functional - rgb(rrr, ggg, bbb)

Each integer represents one sRGB color component in the order red, green and blue, separated by a comma and optionally by whitespace. Each integer is in the range 0 to 255. This syntactical form can represent 16777216 colors. Examples: rgb(233, 150, 122) (i.e a salmon pink), rgb(255, 165, 0) (i.e an orange).

Float functional - rgb(R%, G%, B%)

Each percentage value represents one sRGB color component in the order red, green and blue, separated by a comma and optionally by whitespace. For colors inside the sRGB gamut, the range of each component is 0.0% to 100.0% and an arbitrary number of decimal places may be supplied. Scientific notation is not supported. This syntactical form can represent an arbitrary range of colors, completely covering the sRGB gamut. Color values where one or more components are below 0.0% or above 100.0% represent colors outside the sRGB gamut. Examples: rgb(12.375%, 34.286%, 28.97%).

Color keyword

The sixteen color keywords from HTML 4 must be supported.

11.13.2 HTML Color keywords

	black	rgb(0, 0, 0)		green	rgb(0, 128, 0)
	silver	rgb(192, 192, 192)		lime	rgb(0, 255, 0)
	gray	rgb(128, 128, 128)		olive	rgb(128, 128, 0)
	white	rgb(255, 255, 255)		yellow	rgb(255, 255, 0)
	maroon	rgb(128, 0, 0)		navy	rgb(0, 0, 128)
	red	rgb(255, 0, 0)		blue	rgb(0, 0, 255)
	purple	rgb(128, 0, 128)		teal	rgb(0, 128, 128)
	fuchsia	rgb(255, 0, 255)		aqua	rgb(0, 255, 255)

11.13.3 System Paint

The following list of predefined System Paint servers must be supported. The type of paint returned depends on the operating system, user choices, and the implementation. The paint may be a solid color (which may include opacity) or a gradient (which may include multiple stop opacities).

The names are intended to be descriptive. The implementation should attempt to map the predefined System Paint servers to the current appearance of user interface elements on the platform, including user color choices. For each of the predefined System Paint servers below, the text indicates the appropriate user interface element.

ActiveBorder

Active window border.

ActiveCaption

Active window caption.

AppWorkspace

Background color of multiple document interface.

Background

Desktop background.

ButtonFace

Face color for three-dimensional display elements.

ButtonHighlight

Dark shadow for three-dimensional display elements (for edges facing away from the light source).

ButtonShadow

Shadow color for three-dimensional display elements.

ButtonText

Text on push buttons.

CaptionText

Text in caption, size box, and scrollbar arrow box.

GrayText

Disabled ('grayed') text.

Highlight

Item(s) selected in a control.

HighlightText

Text of item(s) selected in a control.

InactiveBorder

Inactive window border.

InactiveCaption

Inactive window caption.

InactiveCaptionText

Color of text in an inactive caption.

InfoBackground

Background color for tooltip controls.

InfoText

Text color for tooltip controls.

Menu

Menu background.

MenuText

Text in menus.

Scrollbar

Scroll bar gray area.

ThreeDDarkShadow

Dark shadow for three-dimensional display elements.

ThreeDFace

Face color for three-dimensional display elements.

ThreeDHighlight

Highlight color for three-dimensional display elements.

ThreeDLightShadow

Light color for three-dimensional display elements (for edges facing the light source).

ThreeDShadow

Dark shadow for three-dimensional display elements.

Window

Window background.

WindowFrame

Window frame.

WindowText

Text in windows.

11.13.4 The solidColor Element

The **solidColor** element is a paint server that provides a single color with opacity. It can be referenced like the other paint servers (i.e. gradients).

Schema: solidColor

```
<define name='solidColor'>
  <element name='solidColor'>
    <ref name='solidColor.AT' />
    <ref name='solidColor.CM' />
  </element>
</define>

<define name='solidColor.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
      <ref name='svg.Handler.group' />
      <ref name='svg.Discard.group' />
    </choice>
  </zeroOrMore>
</define>
```



```

<define name='solidColor.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
</define>

<define name='svg.Properties.attr' combine='interleave'>
  <optional>
    <attribute name='solid-color' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <value>inherit</value>
        <ref name='SVGColor.datatype' />
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='solid-opacity' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <value>inherit</value>
        <ref name='OpacityValue.datatype' />
      </choice>
    </attribute>
  </optional>
</define>

```

The **solid-color** property specifies the color that shall be used for this **solidColor** element. The keyword **currentColor** can be specified in the same manner as within a `<paint>` specification for the **fill** and **stroke** properties.

'solid-color'

Value: `currentColor` | `<color>` | `inherit`

Initial: `black`

Applies to: **solidColor** elements

Inherited: `no`

Percentages: `N/A`

Media: `visual`

Animatable: `yes`

Computed value: Specified value, except `inherit`

The **solid-opacity** property defines the opacity of a given solid color.

'solid-opacity'

Value: `<opacity-value>` | `inherit`

Initial: `1`

Applies to: **solidColor** elements

Inherited: `no`

Percentages: `N/A`

Media: `visual`

Animatable: `yes`

Computed value: Specified value, except `inherit`

<opacity-value>

The opacity of the **solidColor**. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) must be clamped to this range. (See [Clamping values which are restricted to a particular range.](#))

The **solidColor** paint server applies paint of the specified color using the opacity defined in **solid-opacity**. The value of **solid-opacity** is independent of the opacity used to render the paint via fill or stroke (see [alpha compositing](#)).

Below is an example of the **solidColor** element:

Example: [solidcolor.svg](#)

```

<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
xml:id="svg-root" width="480" height="360" viewBox="0 0 480 360">
  <title>&lt;solidColor&gt; Example</title>

  <defs>
    <solidColor xml:id="solidMaroon" solid-color="maroon" solid-opacity="0.7"/>
  </defs>

```

```

<g>
  <circle transform="translate(100, 150)" fill="url(#solidMaroon)" r="30" />
  <rect fill="url(#solidMaroon)" transform="translate(190, 150)" x="-30" y="-30" width="60" height="60" />
  <path fill="url(#solidMaroon)" transform="translate(270, 150)" d="M 0 -30 L 30 30 L -30 30 Z" />
  <text fill="url(#solidMaroon)" transform="translate(340, 150)" y="21" font-weight="bold" font-size="60">A</text>
</g>
</svg>

```



11.13.5 The SVG 'color' property

The **'color'** property, which defined in CSS2 as the color of text, does not directly apply to SVG elements. The value of the SVG color property may however be used to provide an indirect value for those properties which allow the **currentColor** keyword: the **'fill'**, **'stroke'**, **'solid-color'** and **'stop-color'** properties.

'color'

<i>Value:</i>	<color> inherit
<i>Initial:</i>	depends on user agent
<i>Applies to:</i>	None. Indirectly affects other properties via currentColor
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	Specified value, except inherit

Except for any additional information provided in this specification, the normative definition of the property is in [\[CSS2\]](#).

11.14 Paint Servers

With SVG, you can fill (i.e., paint the interior) or stroke (i.e., paint the outline) of shapes and text using one of the following:

- [color](#) (using `<color>` or the **'solidColor'** element)
- [gradients](#) (linear or radial)

SVG uses the general notion of a **paint server**. Gradients and patterns are just specific types of built-in paint servers. The **'solidColor'** element is another built-in paint server, described in [Color](#).

Paint servers are referenced using an [IRI reference](#) on a **'fill'** or **'stroke'** property.

11.15 Gradients

Gradients consist of continuously smooth color transitions along a vector from one color to another, possibly followed by additional transitions along the same vector to other colors. SVG provides for two types of gradients, [linear gradients](#) and [radial gradients](#).

Once defined, gradients are then referenced using **'fill'** or **'stroke'** properties on a given [graphics element](#) to indicate that the given element shall be filled or stroked with the referenced gradient.

11.15.1 Linear gradients

Linear gradients are defined by a **'linearGradient'** element.

Schema: linearGradient

```
<define name='linearGradient'>
  <element name='linearGradient'>
    <ref name='linearGradient.AT' />
    <ref name='GradientCommon.CM' />
  </element>
</define>

<define name='linearGradient.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.X12Y12.attr' />
</define>

<define name='svg.GradientCommon.attr' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
  <optional>
    <attribute name='gradientUnits' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <value>userSpaceOnUse</value>
        <value>objectBoundingBox</value>
      </choice>
    </attribute>
  </optional>
</define>

<define name='GradientCommon.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
      <ref name='svg.Discard.group' />
      <ref name='stop' />
    </choice>
  </zeroOrMore>
</define>
```

Attribute definitions:

gradientUnits = "userSpaceOnUse | objectBoundingBox"

Defines the coordinate system for attributes [x1, y1, x2, y2](#) that shall be used when rendering the gradient.

If **gradientUnits="userSpaceOnUse"**, [x1, y1, x2, y2](#) shall represent values in the coordinate system that results from taking the current user coordinate system in place at the time when the gradient element is referenced (i.e., the user coordinate system for the element referencing the gradient element via a **'fill'** or **'stroke'** property).

If **gradientUnits="objectBoundingBox"**, the user coordinate system for attributes [x1, y1, x2, y2](#) shall be established using the bounding box of the element to which the gradient is applied (see [Object bounding box units](#)).

When **gradientUnits="objectBoundingBox"** the stripes of the linear gradient shall be perpendicular to the gradient vector in object bounding box space (i.e., the abstract coordinate system where (0,0) is at the top/left of the object bounding box and (1,0) is at the top/right of the object bounding box). When the object's bounding box is not square, the stripes that are conceptually perpendicular to the gradient vector within object bounding box space shall render non-perpendicular relative to the gradient vector in user space due to application of the non-uniform scaling transformation from bounding box space to user space.

If attribute **gradientUnits** is not specified, then the effect shall be as if a value of **objectBoundingBox** were specified.

Animatable: yes.

x1 = "<coordinate>"

[x1, y1, x2, y2](#) define a *gradient vector* for the linear gradient. This *gradient vector* provides starting and ending points onto which the [gradient stops](#) shall be mapped. The values of [x1, y1, x2, y2](#) must be numbers.

If the attribute is not specified, the effect shall be as if a value of "0" were specified.

Animatable: yes.

y1 = "<coordinate>"

See [x1](#).

If the attribute is not specified, the effect shall be as if a value of "0" were specified.

Animatable: yes.

`x2 = "<coordinate>"`

See [x1](#).

If the attribute is not specified, the effect shall be as if a value of "1" were specified.

Animatable: yes.

`y2 = "<coordinate>"`

See [x1](#).

If the attribute is not specified, the effect shall be as if a value of "1" were specified.

Animatable: yes.

If `x1 = x2` and `y1 = y2`, then the area to be painted shall be painted as a single color using the color and opacity of the last [gradient stop](#).

If the gradient starts or ends inside the bounds of the *target rectangle* the terminal colors of the gradient shall be used to fill the remainder of the target region.

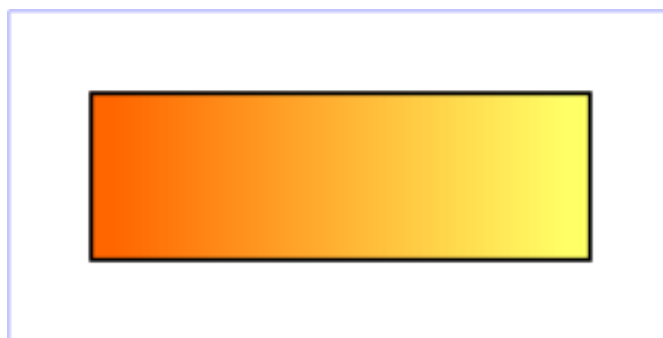
[Properties](#) shall inherit into the **'linearGradient'** element from its ancestors; properties shall *not* inherit from the element referencing the **'linearGradient'** element.

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

Example [lingrad01](#) shows how to fill a rectangle by referencing a linear gradient paint server.

Example: [13_01.svg](#)

```
<?xml version="1.0"?>
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example lingrad01 - fill a rectangle using a
    linear gradient paint server</desc>
  <g>
    <defs>
      <linearGradient xml:id="MyGradient">
        <stop offset="0.05" stop-color="#F60" />
        <stop offset="0.95" stop-color="#FF6" />
      </linearGradient>
    </defs>
    <!-- Outline the drawing area in blue -->
    <rect fill="none" stroke="blue"
      x="1" y="1" width="798" height="398"/>
    <!-- The rectangle is filled using a linear gradient paint server -->
    <rect fill="url(#MyGradient)" stroke="black" stroke-width="5"
      x="100" y="100" width="600" height="200"/>
  </g>
</svg>
```



11.15.2 Radial gradients

Radial gradients are defined by a **'radialGradient'** element.

```

<define name='radialGradient'>
  <element name='radialGradient'>
    <ref name='radialGradient.AT' />
    <ref name='GradientCommon.CM' />
  </element>
</define>

<define name='radialGradient.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
  <ref name='svg.CxCy.attr' />
  <ref name='svg.R.attr' />
</define>

```

Attribute definitions:

gradientUnits = "userSpaceOnUse | objectBoundingBox"

Defines the coordinate system for attributes [cx](#), [cy](#), [r](#) that shall be used when rendering the gradient.

If **gradientUnits="userSpaceOnUse"**, [cx](#), [cy](#), [r](#) shall represent values in the coordinate system that results from taking the current user coordinate system in place at the time when the gradient element is referenced (i.e., the user coordinate system for the element referencing the gradient element via a **'fill'** or **'stroke'** property).

If **gradientUnits="objectBoundingBox"**, the user coordinate system for attributes [cx](#), [cy](#), [r](#) shall be established using the bounding box of the element to which the gradient is applied (see [Object bounding box units](#)).

When **gradientUnits="objectBoundingBox"** the rings of the radial gradient shall be circular with respect to the object bounding box space (i.e., the abstract coordinate system where (0,0) is at the top/left of the object bounding box and (1,1) is at the bottom/right of the object bounding box). When the object's bounding box is not square, the rings that are conceptually circular within object bounding box space shall render as elliptical due to application of the non-uniform scaling transformation from bounding box space to user space.

If attribute **gradientUnits** is not specified, then the effect shall be as if a value of **objectBoundingBox** were specified.

[Animatable](#): yes.

cx = "<coordinate>"

[cx](#), [cy](#), [r](#) define the largest (i.e., outermost) circle for the radial gradient. The gradient shall be drawn such that the "1" [gradient stop](#) is mapped to the perimeter of this largest (i.e., outermost) circle.

If the attribute is not specified, the effect shall be as if a value of "0.5" were specified.

The gradient shall be drawn such that the "0" [gradient stop](#) is mapped to (cx, cy).

[Animatable](#): yes.

cy = "<coordinate>"

See [cx](#).

If the attribute is not specified, the effect shall be as if a value of "0.5" were specified.

[Animatable](#): yes.

r = "<length>"

See [cx](#).

A negative value shall be treated as unsupported. A value of zero shall cause the area to be painted as a single color using the color and opacity of the last [gradient stop](#).

If the attribute is not specified, the effect shall be as if a value of "0.5" were specified.

[Animatable](#): yes.

If the gradient starts or ends inside the bounds of the object(s) being painted by the gradient the terminal colors of the gradient shall be used to fill the remainder of the target region.

[Properties](#) shall inherit into the **'radialGradient'** element from its ancestors; properties shall *not* inherit from the element referencing the **'radialGradient'** element.

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

Example [radgrad01](#) shows how to fill a rectangle by referencing a radial gradient paint server.

Example: 13_02.svg

```

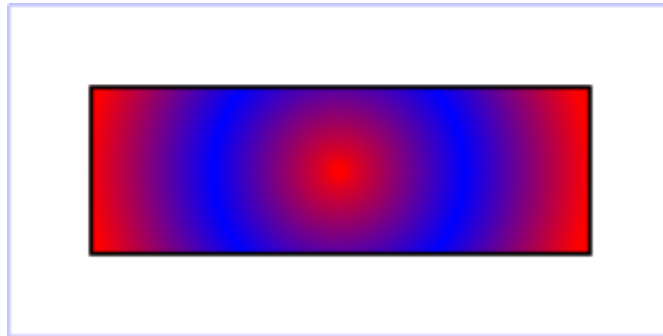
<?xml version="1.0"?>
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example radgrad01 - fill a rectangle by referencing a
    radial gradient paint server</desc>

```

```

<g>
  <defs>
    <radialGradient xml:id="MyGradient" gradientUnits="userSpaceOnUse"
      cx="400" cy="200" r="300">
      <stop offset="0" stop-color="red" />
      <stop offset="0.5" stop-color="blue" />
      <stop offset="1" stop-color="red" />
    </radialGradient>
  </defs>
  <!-- Outline the drawing area in blue -->
  <rect fill="none" stroke="blue"
    x="1" y="1" width="798" height="398"/>
  <!-- The rectangle is filled using a radial gradient paint server -->
  <rect fill="url(#MyGradient)" stroke="black" stroke-width="5"
    x="100" y="100" width="600" height="200"/>
</g>
</svg>

```



11.15.3 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.

Schema: stop

```

<define name='stop'>
  <element name='stop'>
    <ref name='stop.AT' />
    <ref name='stop.CM' />
  </element>
</define>

<define name='stop.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
    </choice>
  </zeroOrMore>
</define>

<define name='stop.AT' combine='interleave'>
  <ref name='svg.Properties.attr' />
  <ref name='svg.Core.attr' />
  <attribute name='offset' svg:animatable='true' svg:inheritable='false'>
    <ref name='GradientOffset.datatype' />
  </attribute>
</define>

```

Attribute definitions:

offset = "[<number>](#)"

The **offset** attribute is a [<number>](#) (usually ranging from 0 to 1) which indicates where the gradient stop shall be placed. For linear gradients, the **offset** attribute represents a location along the *gradient vector*. For radial gradients, it represents a relative distance from (cx,cy) to the edge of the outermost/largest circle.

If the attribute is not specified, the effect is as if a value of "0" was specified. [Animatable](#): yes.

The **'stop-color'** property specifies the color that shall be used at the gradient stop. The keyword **currentColor** can be specified in the same manner as within a [<paint>](#) specification for the **'fill'** and **'stroke'** properties.

'stop-color'

Value: **currentColor** | [<color>](#) | **inherit**

<i>Initial:</i>	black
<i>Applies to:</i>	' stop ' elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	Specified value, except inherit

The '**stop-opacity**' property specifies the opacity that shall be used for the gradient stop.

'stop-opacity'

<i>Value:</i>	<opacity-value> <u>inherit</u>
<i>Initial:</i>	1
<i>Applies to:</i>	' stop ' elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual
<i>Animatable:</i>	yes
<i>Computed value:</i>	Specified value, except inherit

The gradient paint server applies paint of the specified gradient using the opacities defined by **stop-opacity** values. The values of **stop-opacity** are independent of the opacity used to render the paint via fill or stroke (see [alpha compositing](#))

Some notes on gradients:

- Any gradient offset values outside the range 0.0 to 1.0 must be clamped to this range. (See [Clamping values which are restricted to a particular range.](#))
- It is necessary that at least two stops defined to have a gradient effect. If no stops are defined, then painting shall occur as if 'none' were specified as the paint style. If one stop is defined, then painting shall occur with the solid color fill using the color defined for that gradient stop.
- Each gradient offset value is required to be equal to or greater than the previous gradient stop's offset value. If a given gradient stop's offset value is not equal to or greater than all previous offset values, then the offset value must be 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 shall control the color value at the overlap point. In particular:

```
<stop offset="0" stop-color="white"/>
<stop offset=".2" stop-color="red"/>
<stop offset=".2" stop-color="blue"/>
<stop offset="1" stop-color="black"/>
```

will have approximately the same effect as:

```
<stop offset="0" stop-color="white"/>
<stop offset=".199999999" stop-color="red"/>
<stop offset=".2" stop-color="blue"/>
<stop offset="1" stop-color="black"/>
```

which is a gradient that goes smoothly from white to red, then abruptly shifts from red to blue, and then goes smoothly from blue to black.

11.16 Paint Attribute Module

The Paint Attribute Module contains the following attributes:

- [color](#)
- [color-rendering](#)
- [fill](#)
- [fill-rule](#)
- [stroke](#)
- [stroke-dasharray](#)
- [stroke-dashoffset](#)
- [stroke-linecap](#)
- [stroke-linejoin](#)
- [stroke-miterlimit](#)
- [stroke-width](#)

11.17 Opacity Attribute Module

The Opacity Attribute Module contains the following attributes:

- [fill-opacity](#)
- [stroke-opacity](#)

[RNG] [[Feature String](#)]

11.18 Graphics Attribute Module

The Graphics Attribute Module contains the following attributes:

- [display](#)
- [image-rendering](#)
- [pointer-events](#)
- [shape-rendering](#)
- [text-rendering](#)
- [visibility](#)

[RNG] [[Feature String](#)]

11.19 Gradient Module

The Gradient Module contains the following elements:

- [linearGradient](#)
- [radialGradient](#)
- [stop](#)

[RNG] [[Feature String](#)]

11.20 Solid Color Module

The Solid Color Module contains the following element:

- [solidColor](#)

[RNG] [[Feature String](#)]

12 Multimedia

Contents

- 12.1 [Media elements](#)
 - 12.1.1 [Media timeline and document timeline](#)
 - 12.1.2 [Media availability](#)
 - 12.1.3 [Platform limits](#)
- 12.2 [The 'audio' element](#)
- 12.3 [The 'video' element](#)
 - 12.3.1 [Restricting the transformation of the 'video' element](#)
 - 12.3.2 [Restricting compositing of the 'video' element](#)
 - 12.3.3 [Examples](#)
- 12.4 [The 'animation' element](#)
- 12.5 [The audio-level property](#)
- 12.6 [Attributes for run-time synchronization](#)
- 12.7 [The 'initialVisibility' attribute](#)
- 12.8 [Audio Module](#)
- 12.9 [Video Module](#)
- 12.10 [Animation Module](#)

12.1 Media elements

SVG supports media elements similar to the [SMIL 2.0 Media Elements](#). Media elements define their own timelines within their time container. All SVG Media elements support the [SVG Timing attributes](#) and [run time synchronization](#).

The following elements are media elements:

- [audio](#)
- [video](#)
- [animation](#)

12.1.1 Media timeline and document timeline

Media elements start playing, when they become active, i.e. at a time specified in the document timeline which depends on their [begin](#) attribute (see [SVG Timing attributes](#)). However, depending on the value of the [timelineBegin](#) attribute on the [rootmost svg element](#), the actual beginning of the document timeline may be delayed until the whole document is loaded. This is the case when [timelineBegin](#) is set to **'onLoad'**. In that case, the beginning of the actual playback of the media will be delayed but the media begin time in the document timeline will remain as specified.

Note: **'image'** elements are not considered as media elements because they are not timed. They start playing at time 0 in the document timeline.

The following examples illustrate this behavior:

Example: video-timelineBegin-01.svg

```
<?xml version="1.0"?>
<svg id="A" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2" baseProfile="tiny"
    timelineBegin="onLoad">
  <!-- process time = t0 -->
  <!-- ...[many elements]... -->
  <!-- additional process time = t1 = 5s -->
  <video xlink:href="myvideo.mp4" begin="0s"/>
  <!-- additional process time = t2 = 1s -->
</svg>
```

In this example, the document timeline will start after the document is fully processed, i.e. at time $t_0+t_1+t_2 \geq 6s$. The video will start when the document is loaded. But, at that time, the document time will be 0. So, the video will start with the first frame of the video.

Example: video-timelineBegin-02.svg

```
<?xml version="1.0"?>
<svg id="B" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2" baseProfile="tiny"
    timelineBegin="onStart">
  <!-- process time = t0 -->
  <!-- ...[many elements]... -->
  <!-- additional process time = t1 = 5s -->
  <video xlink:href="myvideo.mp4" begin="0s"/>
  <!-- additional process time = t2 = 1s -->
</svg>
```

In this example, the document timeline will start when the svg tag is fully parsed and processed, i.e. at time t_0 . The video will also start at document time 0, but since the video will only be processed when document time is $t_0+t_1+t_2$, the video will start displaying the frame at time $t_0+t_1+t_2$ in the video timeline.

Furthermore, the time in the media timeline which is played, e.g. the exact frame of video or the exact sample of audio that is played, can be altered by the [syncBehavior](#) attribute. The following examples illustrate this behavior. These examples are the same as the previous ones, but the value of the [syncBehavior](#) attribute are replaced from the default one (**'default'**, interpreted as **'locked'** in that case since the [syncBehaviorDefault](#) is not specified) to **'independent'**.

Example: video-timelineBegin-03.svg

```
<?xml version="1.0"?>
<svg id="A" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2" baseProfile="tiny"
    timelineBegin="onLoad" <!-- process time = t0 -->
<!-- ...[many elements]... --> <!-- additional process time = t1 = 5s -->
<video xlink:href="myvideo.mp4" begin="0s" syncBehavior="independent"/> <!-- additional process time = t2 = 1s -->
</svg>
```

Example: video-timelineBegin-04.svg

```
<?xml version="1.0"?>
<svg id="B" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.2" baseProfile="tiny"
    timelineBegin="onStart" <!-- process time = t0 -->
<!-- ...[many elements]... --> <!-- additional process time = t1 = 5s -->
<video xlink:href="myvideo.mp4" begin="0s" syncBehavior="independent"/> <!-- additional process time = t2 = 1s -->
</svg>
```

In example video-timelineBegin-03.svg, the video does not start until the document's load event, whereas in video-timelineBegin-04.svg, the video begins as soon as the video element is parsed and the video is ready for rendering. In both cases, since the timeline of the document and of the video are independent, when the video will start, it will start from the first frame, i.e. time 0 in the media timeline.

12.1.2 Media availability

The value of the [externalResourcesRequired](#) attribute may also delay the actual time at which a media (even images) will start playing, but it does not affect the time in the document timeline. Indeed, media elements and the image element may require external resources referenced by the [xlink:href](#) attribute. If the [externalResourcesRequired](#) attribute is set to 'true', on the resource or on a parent in the scene tree, e.g. a 'g' element, then this external resource has to become available before the media can start. If the [externalResourcesRequired](#) attribute is set to 'false', the media element or the image element will start playing as soon as it becomes active.

The meaning of 'available' depends on the Media Type, on the protocol used to access the resource as well as on the implementation. For example, if a protocol like HTTP is used, available may mean that the whole resource is downloaded. It may also mean that a coarse version of the resource is available, for example in case of progressive PNG (see [PNG Pass extraction](#)). In that case, it is an implementation choice to decide or not to display the coarse version before the whole resource is downloaded. Another example is when streaming protocols like RTSP/RTP are used. In that case, 'available' usually means that enough stream has been buffered before the playback may start. To reduce the amount of time required for a media to become available, authors are encouraged to use the prefetch element to signal that external resources have to be prefetched.

12.1.3 Platform limits

Particular platforms may have restrictions on the number of audio voices or channels that can be mixed, or the number of video streams that may be presented concurrently. Since these vary, the SVG language itself does not impose any such limits on audio or video.

12.2 The 'audio' element

The **audio** element specifies an audio file which is to be rendered to provide synchronized audio. The usual SMIL timing features are used to start and stop the audio at the appropriate times. An [xlink:href](#) must be used to link to the audio content. No visual representation shall be produced. However, content authors can if desired create graphical representations of control panels to start, stop, pause, rewind, or adjust the volume of audio content.

If 2 or more audio streams are active at the same time, their rendering should be mixed in proportions equal to the computed value of the audio-level property of each audio stream. An audio stream may be active if it is referred to by an active audio element or if it is part of video content that is referred to by an active video element.

Schema: audio

```
<define name='audio'>
  <element name='audio'>
    <ref name='audio.AT' />
    <ref name='audio.CM' />
  </element>
</define>

<define name='audio.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.XLinkEmbed.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateSync.attr' />
  <ref name='svg.Media.attr' />
  <ref name='svg.ContentType.attr' />
</define>

<define name='audio.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
      <ref name='svg.Handler.group' />
      <ref name='svg.Discard.group' />
    </choice>
  </zeroOrMore>
</define>
```

Attribute definitions:

xlink:href = "[<iri>](#)"

An [IRI reference](#). An empty attribute value (xlink:href="") disables playback of the element. If the attribute is not specified, the effect is as if an empty value ("") was specified.

[Animatable](#): yes.

`type = "<media/type>"`

The audio format. Implementations may choose not to fetch audios of formats that they do not support.

[Animatable](#): no.

Run-time synchronization attributes

See [definition](#).

SVG Timing attributes

If the ['fill'](#) attribute is specified, it has no effect. See [definition](#).

The following example illustrates the use of the audio element. When the button is pushed, the audio file is played three times.

Example: media01.svg

```
<svg width="100%" height="100%" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>SVG audio example</desc>
  <audio xlink:href="ouch.ogg" audio-level="0.7" type="audio/vorbis"
    begin="mybutton.click" repeatCount="3"/>
  <g xml:id="mybutton">
    <rect width="150" height="50" x="20" y="20" rx="10"
      fill="#ffd" stroke="#933" stroke-width="5"/>
    <text x="95" y="55" text-anchor="middle" font-size="30"
      fill="#933">Press Me</text>
  </g>
  <rect x="0" y="0" width="190" height="90" fill="none" stroke="#777"/>
</svg>
```

When rendered, this looks as follows:



This specification does not mandate support for any particular audio format. Content can check for a particular audio codec with the [requiredFormats](#) test attribute.

12.3 The 'video' element

The **video** element specifies a video file which is to be rendered to provide synchronized video. It should not point to other types of media. The usual SMIL timing features are used to start and stop the video at the appropriate times. An [xlink:href](#) must be used to link to the video content. It is assumed that the video content may also include an audio stream, since this is the usual way that video content is produced, and thus the audio shall be controlled by the **video** element's media attributes.

If 2 or more audio streams are active at the same time, their rendering should be mixed in proportions equal to the computed value of the audio-level property of each audio stream. An audio stream may be active if it is referred to by an active audio element or if it is part of video content that is referred to by an active video element.

The **video** element produces a rendered result, and thus has [width](#), [height](#), [x](#) and [y](#) attributes.

A **'video'** element must establish a new viewport for the referenced file as described in [Establishing a new viewport](#). Therefore the **video** element supports the ['viewport-fill'](#) and ['viewport-fill-opacity'](#) properties. The bounds for the new viewport shall be defined by attributes [x](#), [y](#), [width](#) and [height](#). The placement and scaling of the referenced video shall be controlled by the [preserveAspectRatio](#) attribute on the **'video'** element.

The value of the ['viewBox'](#) attribute to use when evaluating the [preserveAspectRatio](#) attribute shall be defined by the referenced content. For content that clearly identifies a [viewBox](#) that value shall be used. For most video content the bounds of the video should be used (i.e. the **'video'** element has an implicit ['viewBox'](#) of "0 0 video-width video-height"). Where no value is readily available the [preserveAspectRatio](#) attribute shall be ignored.

Schema: video

```
<define name='video'>
  <element name='video'>
    <ref name='video.AT' />
    <ref name='video.CM' />
  </element>
</define>

<define name='video.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.Media.attr' />
  <ref name='svg.XLinkEmbed.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateSync.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.XYWH.attr' />
  <ref name='svg.PAR.attr' />
```

```

<ref name='svg.ContentType.attr' />
<ref name='svg.InitialVisibility.attr' />
<optional>
  <attribute name='transformBehavior' a:defaultValue='geometric' svg:animatable='no' svg:inheritable='false'>
    <choice>
      <value>geometric</value>
      <value>pinned</value>
      <value>pinned90</value>
      <value>pinned180</value>
      <value>pinned270</value>
    </choice>
  </attribute>
</optional>
<optional>
  <attribute name='overlay' a:defaultValue='none' svg:animatable='no' svg:inheritable='false'>
    <choice>
      <value>none</value>
      <value>top</value>
    </choice>
  </attribute>
</optional>
</define>

<define name='video.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='svg.Animate.group' />
      <ref name='svg.Handler.group' />
      <ref name='svg.Discard.group' />
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of the rectangular region into which the video is placed.

If the transform behaviour of the video is geometric, this coordinate is one corner of the rectangular region. If it is pinned, this coordinate is the pin point of the rectangular region. See the [transformBehavior attribute](#) for the interpretation of this attribute.

If the attribute is not specified, the effect must be the same as when the coordinate is set to "0".

[Animatable](#): Yes.

y = "[<coordinate>](#)"

The y-axis coordinate of the rectangular region into which the video is placed.

If the transform behaviour of the video is geometric, this coordinate is one corner of the rectangular region. If it is pinned, this coordinate is the pin point of the rectangular region. See the [transformBehavior attribute](#) for the interpretation of this attribute.

If the attribute is not specified, the effect must be the same as when the coordinate is set to "0".

[Animatable](#): Yes.

width = "[<length>](#)"

The width of the rectangular region into which the video is placed.

A negative value shall be treated as unsupported. A value of zero shall disable rendering of the element. If the attribute is not specified, the effect must be the same as when a value of "0" is specified.

[Animatable](#): Yes.

height = "[<length>](#)"

The height of the rectangular region into which the video is placed.

A negative value shall be treated as unsupported. A value of zero shall disable rendering of the element. If the attribute is not specified, the effect must be the same as when a value of "0" is specified.

[Animatable](#): Yes.

xlink:href = "[<iri>](#)"

An [IRI reference](#). An empty attribute value (xlink:href="") disables rendering of the element. If the attribute is not specified, the effect is as if an empty value ("") was specified.

[Animatable](#): Yes.

preserveAspectRatio = [[defer](#)] [<align>](#) [[<meet>](#)]

Indicates whether or not to force uniform scaling. (See [The preserveAspectRatio attribute](#) for the syntax of [<align>](#) and the interpretation of this attribute.)

[Animatable](#): Yes.

type = "[<media/type>](#)"

The video format. Implementations may choose not to fetch videos of formats that they do not support.

[Animatable](#): No.

transformBehavior = "geometric" | "pinned"

See [attribute definition](#) for description.

[Animatable](#): No

overlay = "top" | "none"

See [attribute definition](#) for description.

[Animatable](#): No

initialVisibility = "whenStarted" | "always"

See [attribute definition](#) for description.

[Animatable](#): No

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

[Animatable](#): Yes

Navigation Attributes

See [definition](#).

Run-time synchronization attributes

See [definition](#).

SVG Timing attributes

See [definition](#).

The following example illustrates the use of the **'video'** element. The video content is partially obscured by other graphics elements. This example shows the **video** element being rendered into an offscreen buffer and then transformed and composited in the normal way, so that it behaves like any other graphical primitive such as an image or a rectangle. In this manner, the video element may be scaled, rotated, skewed, displayed at various sizes, and animated.

Example: media02.svg

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="420" height="340" viewBox="0 0 420 340">
  <desc>SVG 1.2 video example</desc>
  <g>
    <circle cx="0" cy="0" r="170" fill="#da4" fill-opacity="0.3"/>
    <video xlink:href="noonoo.avi" audio-level=".8" type="video/x-msvideo"
      width="320" height="240" x="50" y="50" repeatCount="indefinite"/>
    <circle cx="420" cy="340" r="170" fill="#927" fill-opacity="0.3"/>
    <rect x="1" y="1" width="418" height="338" fill="none"
      stroke="#777" stroke-width="1"/>
  </g>
</svg>
```

[Show this example](#) of the **video** element (requires an SVG 1.2 viewer and support for a Windows AVI using Motion JPEG. This is a 3.7M video file).

When rendered, this looks as follows:



This specification does not mandate support for any particular video format. Content can check for a particular video codec with the [requiredFormats](#) test attribute.

The content creator should be aware that video is a feature that may not be available on all target devices. In order to create interoperable content the content creator should provide a fall-back alternative by using the **'switch'** element. The following feature string is defined for checking for video support: <http://www.w3.org/Graphics/SVG/feature/1.2/#Video>. Video may not be completely supported on a resource limited device. SVGT 1.2 introduces more granular video rendering control to provide reproducible results in all environments. This control is documented in the two following sections.

12.3.1 Restricting the transformation of the 'video' element

Transforming video is an expensive operation that should be used with caution, especially on content targeted for mobile devices. Using transformed video may also lead to non-interoperable content since not all devices will support this feature. To give the content creator control over video transformation SVGT 1.2 introduces the **transformBehavior** attribute and a corresponding feature string: <http://www.w3.org/Graphics/SVG/feature/1.2/#TransformedVideo>. A viewer supporting video transformation must treat the **'video'** element like any other element regarding transformations. A viewer not supporting video transformation must treat the video as a point (given by x and y attributes). The width and height attributes shall be ignored if present and instead the width and height (in device pixels) shall be taken from the media itself. The video must be displayed with its center aligned with the origin of the local coordinate system.

A content creator can use the **transformBehavior** attribute to explicitly choose the transform behavior on a viewer supporting transformed video. This might be of interest to increase the performance of content targeting restricted devices.

Attribute definition:

transformBehavior = "geometric" | "pinned" | "pinned90" | "pinned180" | "pinned270"

Define whether a video is transformed/resampled (in essence treated as a geometric rectangle) or pinned/unresampled (ie, treated as a pin point for a non-geometric blit region).

The attribute value can be either of the following:

"geometric"

the media shall be treated as a geometric rectangle in the local coordinate system, defined by x y width and height attributes. the media must be resampled to fill the rectangle and subject to transformation. This is the default [transformBehavior](#)

For the four next values, the following applies:

The media shall be treated as a point, defined by x and y attributes. This point must be transformed to the nearest actual device pixel. Video at the native resolution given by the media shall then painted on a region whose center is the pin point and whose width and height are defined by the media. The pixels must be aligned to the device pixel grid and there shall be no resampling.

"pinned"

The video is displayed without rotation.

"pinned90"

The video is displayed with a fixed rotation equivalent to the effect of transform="rotate(90)".

"pinned180"

The video is displayed with a fixed rotation equivalent to the effect of transform="rotate(180)".

"pinned270"

The video is displayed with a fixed rotation equivalent to the effect of transform="rotate(270)".

[Animatable](#): No.

12.3.2 Restricting compositing of the 'video' element

For the same reasons as restricting transformations the content creator might need to restrict the compositing of video with other elements. Not all devices support compositing of the video element with other content. In that case it is necessary to render the video on top of all other svg content. SVG1.2 therefore introduces the [overlay](#) attribute and a corresponding feature string: <http://www.w3.org/Graphics/SVG/feature/1.2/#CompositedVideo>. A viewer supporting compositing of video must render the video element according to the svg painters model, graphical elements might be rendered on top of the video. A viewer not supporting video compositing must always render the video on top of all other svg elements.

A content creator can use the [overlay](#) attribute to explicitly choose the compositing behavior on a viewer supporting composited video. This may increase the performance of content that is targeted at restricted devices.

Attribute definition:

overlay = "top" | "none"

Define whether a 'video' is rendered according to the svg painters model or if it must be positioned on top of all other svg elements.

The attribute value can be either of the following:

"top"

When a 'video' element has the [overlay](#) set to 'top' it must not be composited to the background as usual. Instead a temporary video canvas must be set aside and drawn last in the whole document's compositing process.

"none"

The 'video' must be rendered according to the svg painters model. This is the default [overlay](#) behavior.

[Animatable](#): No.

If multiple 'video' elements have [overlay='top'](#), the drawing order of those 'video' elements follows the typical SVG rendering order.

12.3.3 Examples

The following example illustrates the use of the <http://www.w3.org/Graphics/SVG/feature/1.2/#TransformedVideo> feature string. A switch element is wrapped around two groups, the first group will render a scaled and rotated video sequence on a viewer supporting video transformations while the second group will render the untransformed video on viewers that doesn't support video transformations.

Example: [media04.svg](#)

```
<svg version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="100%" height="100%" viewBox="0 0 400 300">
  <desc>Example of switching on the http://www.w3.org/TR/SVG12/feature#TransformedVideo feature string</desc>
  <switch>

    <!-- Transformed video group -->
    <g requiredFeatures="http://www.w3.org/TR/SVG12/feature#TransformedVideo"
      transform="translate(-21,-34) scale(1.24) rotate(-30)">
      <rect x="6" y="166" width="184" height="140" fill="none" stroke="blue"
        stroke-width="4" />
      <video xlink:href="ski.avi" audio-level=".8" type="video/x-msvideo"
        x="10" y="170" width="176" height="132"/>
    </g>

    <!-- Untransformed video group -->
    <g>
      <rect x="6" y="166" width="184" height="140" fill="none" stroke="blue"
        stroke-width="4"/>
      <video xlink:href="ski.avi" audio-level=".8" type="video/x-msvideo"
        x="10" y="170"/>
    </g>
  </switch>
</svg>
```

Screenshot of the example above rendered on two viewers, the first one supporting transformed video, the second one not:



SVGT 1.2 viewer supporting transformed video.



SVGT 1.2 viewer not supporting transformed video.

The following example illustrates the use of the <http://www.w3.org/Graphics/SVG/feature/1.2/#ComposedVideo> feature string. A switch element is wrapped around two groups, the first group must render a video with text composited on top on viewers supporting composed video while the second group must render a video with text placed above the video on viewers that do not support composed video.

Example: [media05.svg](#)

```
<?xml version="1.1"?>
<svg version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="100%" height="100%" viewBox="0 0 400 300">
  <desc>Example of switching on the http://www.w3.org/TR/SVG12/feature#ComposedVideo feature string</desc>
  <rect x="2" y="2" width="396" height="296" fill="none" stroke="black"
    stroke-width="2" />
  <rect x="106" y="66" width="184" height="140" fill="none" stroke="blue"
    stroke-width="4" />

  <switch>

    <!-- Composited video group -->
    <g transform="translate(100 0)" requiredFeatures="http://www.w3.org/TR/SVG12/feature#ComposedVideo">
      <video xlink:href="ski.avi" audio-level=".8" type="video/x-msvideo"
        x="10" y="70" width="176" height="132"/>
      <text x="20" y="100" fill-opacity="0.5" fill="blue" font-size="20">Composited title.</text>
    </g>

    <!-- Overlaid video group -->
    <g transform="translate(100 0)" font-size="18">
      <video xlink:href="ski.avi" audio-level=".8" type="video/x-msvideo"
        x="10" y="70" overlay="top" width="176" height="132"/>
      <text x="15" y="60" fill="blue" fill-opacity="0.5">Non-composited title.</text>
    </g>
  </switch>
</svg>
```

Screenshots of the example above rendered on two viewers, the first one supporting composited video, the second one not:



12.4 The 'animation' element

The **'animation'** element specifies an SVG document providing synchronized animated vector graphics. Like **'video'**, the **'animation'** element is a graphical object with size determined by its **'x'**, **'y'**, **'width'** and **'height'** attributes. Furthermore the **'animation'** element supports timing and synchronisation attributes which allows multiple animations to run with independent timelines in the same svg document. Just like **'video'** and **'image'**, the **'animation'** element must not point to document fragments within svg files.

An **'animation'** element establishes a new viewport for the referenced file as described in [Establishing a new viewport](#). The bounds for the new viewport are defined by attributes **'x'**, **'y'**, **'width'** and **'height'**. The **'preserveAspectRatio'** attribute on the [rootmost svg element](#) in the referenced SVG Document shall be ignored (as are its x, y, width and height attributes). Instead, the **'preserveAspectRatio'** attribute on the referencing **'animation'** element shall define how the SVG content is fitted into the viewport. The same rule applies for the **'viewport-fill'** and **'viewport-fill-opacity'** properties.

The value of the **'viewBox'** attribute to use when evaluating the **'preserveAspectRatio'** attribute is defined by the referenced document's **'viewBox'** value. When no value is available the **'preserveAspectRatio'** attribute must be ignored, and only the translation due to the **'x'** and **'y'** attributes of the viewport must be used to display the content.

The referenced SVG document represents a separate document which generates its own parse tree and document object model. Thus, there is no inheritance of properties into the referenced animation. For details, see [Externally referenced documents](#).

The SVG specification does not specify when an animation that is not being displayed should be loaded. A user agent is not required to load animation data for an animation that is not displayed (e.g. `display='none'`). However, it should be noted that this may cause a delay when an animation becomes visible for the first time. In the case where an author wants to suggest that the user agent load animation data before it is displayed, they should use the **'prefetch'** element.

Schema: animation

```
<define name='animation'>
  <element name='animation'>
    <ref name='animation.AT' />
    <ref name='animation.CM' />
  </element>
</define>

<define name='animation.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
```



```

<ref name='svg.FocusHighlight.attr'/>
<ref name='svg.Media.attr'/>
<ref name='svg.Conditional.attr'/>
<ref name='svg.External.attr'/>
<ref name='svg.XLinkEmbed.attr'/>
<ref name='svg.Focus.attr'/>
<ref name='svg.AnimateTiming.attr'/>
<ref name='svg.AnimateSync.attr'/>
<ref name='svg.XYWH.attr'/>
<ref name='svg.PAR.attr'/>
<ref name='svg.Transform.attr'/>
<ref name='svg.InitialVisibility.attr'/>
</define>

<define name='animation.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group'/>
      <ref name='svg.Animate.group'/>
      <ref name='svg.Discard.group'/>
      <ref name='svg.Handler.group'/>
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of one corner of the rectangular region into which the animation is placed.

If the attribute is not specified, the effect shall be as if a value of "0" were specified.

Animatable: yes.

y = "[<coordinate>](#)"

The y-axis coordinate of one corner of the rectangular region into which the animation is placed.

If the attribute is not specified, the effect shall be as if a value of "0" were specified.

Animatable: yes.

width = "[<length>](#)"

The width of the rectangular region into which the animation is placed.

A negative value is unsupported. A value of zero must disable rendering of the element. If the attribute is not specified, the effect shall be as if a value of "0" were specified.

Animatable: yes.

height = "[<length>](#)"

The height of the rectangular region into which the animation is placed.

A negative value is unsupported. A value of zero must disable rendering of the element. If the attribute is not specified, the effect shall be as if a value of "0" were specified.

Animatable: yes.

xlink:href = "[<iri>](#)"

An [IRI reference](#). An empty attribute value (xlink:href="") disables rendering of the element. If the attribute is not specified, the effect is as if an empty value ("") was specified.

Animatable: yes.

preserveAspectRatio = [[defer](#)] [<align>](#) [[<meet>](#)]

Indicates whether or not to force uniform scaling. (See [The preserveAspectRatio attribute](#) for the syntax of <align> and the interpretation of this attribute.)

Animatable: yes.

initialVisibility = "[whenStarted](#)" | "[always](#)"

See [attribute definition](#) for description.

Animatable: No

focusable = "[true](#)" | "[false](#)" | "[auto](#)"

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

Run-time synchronization attributes

See [definition](#).

SVG Timing attributes

See [definition](#).

The example below shows basic usage of the [animation](#) element, for another example see [use and animation example](#).

Example: media03.svg

```

<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  version="1.2" baseProfile="tiny">
  <desc>Example of two animation elements pointing to the same content.</desc>

  <animation begin="1" dur="3" repeatCount="1.5" fill="freeze"
    x="100" y="100" width="50" height="50"
    xlink:href="bouncingBall.svg"/>

  <animation begin="2" x="300" y="100" width="50" height="50"
    xlink:href="bouncingBall.svg"/>
</svg>

```

12.5 The audio-level property

The **audio-level** property can be applied to the **'audio'**, **'video'** and **'animation'** elements described above, plus container elements such as the **'g'** element.

'audio-level'

Value: <float> | inherit

Initial: 1

Applies to: **audio**, **video**, **animation** and container elements

Inherited: no

Percentages: N/A

Media: visual, audible

Animatable: yes

Computed value: At the [rootmost svg element](#), the value of the **audio-level**. On other elements, the product of **audio-level** property and element volume of parent.

The **audio-level** property specifies a value between 0 and 1 that is used to calculate the volume of a particular element. Values above 1 and below 0 must be clipped.

An element's volume is the product of its **audio-level** property and the element volume of its parent. The exception to this rule is the [rootmost svg element](#), where the element volume is only the value of its **audio-level** property. Therefore, element volume must be calculated in a similar way to opacity.

If the audio-level of an element to a value less than 1.0, all children elements will be quieter. It is not possible to increase the volume on an element.

The output signal level is calculated using the logarithmic scale described below (where vol is the value of the element volume):

```
dB change in signal level = 20 * log10(vol)
```

If the element has an element volume of 0, then the output signal must be inaudible. If the element has an element volume of 1, then the output signal must be at the system volume level. Neither the **audio-level** property nor the element volume override the system volume setting.

12.6 Attributes for run-time synchronization

SVG supports the five attributes listed below from [SMIL 2.0](#) to control run-time synchronization of timed elements. In SVG1.2 the **syncBehavior**, **syncTolerance** and **syncMaster** attributes can be specified on the **'audio'**, **'video'** and **'animation'** elements. The **syncBehaviorDefault** and **syncToleranceDefault** attributes can be specified on the **svg** element.

'syncBehavior' attribute

Attribute definition:

syncBehavior = "canSlip" | "locked" | "independent" | "default"
See the [SMIL 2.0 Specification](#) for attribute definition.

Animatable: No

'syncBehaviorDefault' attribute

Attribute definition:

syncBehaviorDefault = "canSlip" | "locked" | "independent" | "inherit"
See the [SMIL 2.0 Specification](#) for attribute definition.

Animatable: No

'syncTolerance' attribute

Attribute definition:

syncTolerance = [Clock-value](#) | "default"
See the [SMIL 2.0 Specification](#) for attribute definition.

Animatable: No

'syncToleranceDefault' attribute

Attribute definition:

syncToleranceDefault = [Clock-value](#) | "inherit"
See the [SMIL 2.0 Specification](#) for attribute definition.

Animatable: No

'syncMaster' attribute

Attribute definition:

syncMaster = [Boolean](#)

See the [SMIL 2.0 Specification](#) for attribute definition.

Animatable: No

Example: video content synchronized with some text

The two files below illustrate how it is possible to make sure some video content can be synchronized with some text using the `sync*` attributes.

Example: [sync-attr-main.svg](#)

```
<?xml version="1.0"?>

<svg viewBox="0 0 400 100" height="100%" width="100%"
    xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.2" baseProfile="tiny"
    syncBehaviorDefault="locked">

  <desc>An example which illustrates the use of sync* attributes</desc>

  <video x="10" y="10" xml:id="myclip" xlink:href="rtsp://www.example.org/mysong.m4v" syncMaster="true"/>
  <animation x="10" y="50" xml:id="mylyrics" xlink:href="timed-lyrics.svg"/>
</svg>
```

Example: [timed-lyrics.svg](#)

```
<?xml version="1.0"?>

<svg viewBox="0 0 400 100" height="100%" width="100%"
    xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.2" baseProfile="tiny">

  <desc>This document contains the textual lyrics to synchronize with some video content in the referencing document</desc>

  <g fill="blue" font-family="Arial" font-size="10" transform="translate(20, 20)">
    <text id="Text0" display="none">This is some text</text>
    <set xlink:href="#Text0" attributeName="display" to="inline" begin="0" end="1"/>
    <text id="Text10" display="none">simulating some lyrics</text>
    <set xlink:href="#Text10" attributeName="display" to="inline" begin="1.1" end="2"/>
    <text id="Text20" display="none">displayed synchronously</text>
    <set xlink:href="#Text20" attributeName="display" to="inline" begin="2.1" end="3"/>
    <text id="Text30" display="none">with some video</text>
    <set xlink:href="#Text30" attributeName="display" to="inline" begin="3.1" end="4"/>
    <text id="Text40" display="none">in a different document</text>
    <set xlink:href="#Text40" attributeName="display" to="inline" begin="4.1" end="5"/>
  </g>
</svg>
```

Since the timed elements (**video** and **animation**) do not specify their runtime synchronization behavior using the `syncBehavior` attribute, the behavior is deduced from the `syncBehaviorDefault` attribute on the nearest ancestor, in this case on the **svg** element.

This latter indicates **'locked'** which means that all the timed elements in the subtree share the same timeline. In this case, the main scene timeline, the **video** and **animation** timelines are then locked to each other.

Then, the master is given to the video, which means that if the video is stalled in the streaming session, the timeline of the video will be paused and, as a consequence, the timeline of the lyrics and of the main scene will be paused as well.

12.7 The 'initialVisibility' attribute

The **'initialVisibility'** attribute applies to visual media elements (**video** and **animation**) and is used to control the visibility of the media object before its first active duration period has started. A visible media element that is visible before activation shall have its first frame displayed. For an **animation** element this means the referenced file rendered at time 0. For a **video** element it means the first frame of the video sequence.

Attribute definition:

initialVisibility = "whenStarted" | "always"

Controls the visibility of the media object before its first active duration period has started.

The attribute value can be either of the following:

"whenStarted"

The default value. Means that the media object becomes visible when its first active duration period starts.

"always"

The media object is visible from the initialization of the parent time container (i.e. time 0 of the parent **svg** document).

Animatable: No

12.8 Audio Module

The Audio Module contains the following element:

- [audio](#)

[RNG] [[Feature String](#)]

12.9 Video Module

The Video Module contains the following element:

- [video](#)

[RNG] [[Feature String](#)]

12.10 Animation Module

The Animation Module contains the following element:

- [animation](#)

[RNG] [[Feature String](#)]

13 Interactivity

Contents

- 13.1 [Introduction](#)
- 13.2 [Complete list of supported events](#)
- 13.3 [User interface events](#)
- 13.4 [Pointer events](#)
- 13.5 [Text events](#)
- 13.6 [Key events](#)
- 13.7 [Event Dispatching](#)
- 13.8 [Processing order for user interface events](#)
- 13.9 [The 'pointer-events' property](#)
- 13.10 [Magnification and panning](#)
- 13.11 [Element focus](#)
 - 13.11.1 [The focusable attribute](#)
- 13.12 [Navigation](#)
 - 13.12.1 [Navigation behavior](#)
 - 13.12.2 [Specifying navigation](#)
 - 13.12.3 [Specifying Focus Highlighting](#)
 - 13.12.4 [Obtaining and listening to focus programmatically](#)

13.1 Introduction

SVG content can be interactive (i.e., responsive to user-initiated events) by utilizing the following features in the SVG language:

- User-initiated actions such as a key-press can cause [animations](#) to start or stop, [scripts](#) to execute or listeners elements to trigger handler elements.
- The user can initiate hyperlinks to new Web pages (see [Links out of SVG content: the 'a' element](#)) by actions such as a stylus click on a particular graphics element.
- In many cases, depending on the value of the [zoomAndPan](#) attribute on the ['svg'](#) element and on the characteristics of the user agent, users are able to zoom into and pan around SVG content.

This chapter describes:

- information about [events](#), including under which circumstances events are triggered
- how to indicate whether a given document can be [zoomed and panned](#)
- Element [focus](#) and [navigation](#).

Related information can be found in other chapters:

- hyperlinks are discussed in [Links](#)
- scripting and handler elements are discussed in [Scripting](#)
- animation is discussed in [Animation](#)

13.2 Complete list of supported events

The following aspects of SVG are affected by events:

- The [SVG uDOM](#) enables a script to [register event listeners](#) so that the script can be invoked when a given event occurs.
- The [ev:event](#) attribute on the [handler](#) element specifies for which event the handler should trigger.
- [Timed Elements](#) can be defined to begin or end based on events.

The following table lists all of the events which must be recognized and supported in SVG. The 'description' column describes the required conditions for the event to occur.

Event Identifier {event-namespace, event-localname}	Description	DOM3 event category	Animation event name	Cancel-able	uDOM interface
{ "http://www.w3.org/2001/xml-events" , "DOMFocusIn" } SVG 1.2 alias: { "http://www.w3.org/2001/xml-events" , "focusin" } (see Notes below).	Occurs when an element receives focus.	UIEvent	focusin	No	UIEvent

<p>{"http://www.w3.org/2001/xml-events", "DOMFocusOut"}</p> <p>SVG 1.2 alias: {"http://www.w3.org/2001/xml-events", "focusout"} (see Notes below).</p>	<p>Occurs when an element loses focus.</p>	<p>UIEvent</p>	<p>focusout</p>	<p>No</p>	<p>UIEvent</p>
<p>{"http://www.w3.org/2001/xml-events", "DOMActivate"}</p> <p>SVG 1.2 alias: {"http://www.w3.org/2001/xml-events", "activate"} (see Notes below).</p>	<p>Occurs when an element is activated, for instance, thru a mouse click or a keypress. A numerical argument is provided to give an indication of the type of activation that occurs: 1 for a simple activation (e.g. a simple click or Enter), 2 for hyperactivation (for instance a double click or Shift Enter).</p>	<p>UIEvent</p>	<p>activate</p>	<p>Yes</p>	<p>UIEvent</p>
<p>{"http://www.w3.org/2001/xml-events", "click"}</p>	<p>Occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: mousedown, mouseup, click. If multiple clicks occur at the same screen location, the sequence repeats with the detail attribute incrementing with each repetition.</p>	<p>MouseEvent</p>	<p>click</p>	<p>Yes</p>	<p>MouseEvent</p>
<p>{"http://www.w3.org/2001/xml-events", "mousedown"}</p>	<p>Occurs when the pointing device button is pressed over an element.</p>	<p>MouseEvent</p>	<p>mousedown</p>	<p>Yes</p>	<p>MouseEvent</p>
<p>{"http://www.w3.org/2001/xml-events", "mouseup"}</p>	<p>Occurs when the pointing device button is released over an element.</p>	<p>MouseEvent</p>	<p>mouseup</p>	<p>Yes</p>	<p>MouseEvent</p>
<p>{"http://www.w3.org/2001/xml-events", "mouseover"}</p>	<p>Occurs when the pointing device is moved onto an element.</p>	<p>MouseEvent</p>	<p>mouseover</p>	<p>Yes</p>	<p>MouseEvent</p>
<p>{"http://www.w3.org/2001/xml-events", "mousemove"}</p>	<p>Occurs when the pointing device is moved while it is over an element.</p>	<p>MouseEvent</p>	<p>mousemove</p>	<p>Yes</p>	<p>MouseEvent</p>

{"http://www.w3.org/2001/xml-events", "mouseout"}	Occurs when the pointing device is moved away from an element.	MouseEvent	mouseout	Yes	MouseEvent
{"http://www.w3.org/2001/xml-events", "textInput"}	One or more characters have been entered.	TextEvent	none	Yes	TextEvent
{"http://www.w3.org/2001/xml-events", "keydown"}	A key is pressed down. (The normative definition of this event is the description in the DOM3 Events specification .)	KeyboardEvent	none	Yes	KeyboardEvent
{"http://www.w3.org/2001/xml-events", "keyup"}	A key is released. (The normative definition of this event is the description in the DOM3 Events specification .)	KeyboardEvent	none	Yes	KeyboardEvent
<p data-bbox="162 1086 533 1149">{"http://www.w3.org/2001/xml-events", "load"}</p> <p data-bbox="162 1234 533 1391">Deprecated backwards-compatibility alias: {"http://www.w3.org/2001/xml-events", "SVGLoad"} (see Notes below).</p>	<p data-bbox="552 801 772 1245">The event is triggered at the point at which the user agent has fully parsed the element and its descendants and is ready to act appropriately upon that element, such as being ready to render the element to the target device.</p> <p data-bbox="552 1254 772 1682">Referenced external resources that are required must be loaded, parsed and ready to render before the event is triggered. Optional external resources are not required to be ready for the event to be triggered.</p>	HTMLEvent	load	No	Event
<p data-bbox="162 1749 533 1812">{"http://www.w3.org/2001/xml-events", "resize"}</p> <p data-bbox="162 1897 533 2054">Deprecated backwards-compatibility alias: {"http://www.w3.org/2001/xml-events", "SVGResize"} (see Notes below).</p>	Occurs when a document view is being resized. This event is only applicable to 'svg' elements and is dispatched after the resize operation has taken place. The target of the event is the 'svg' element.	HTMLEvent	resize	No	Event

<p>{http://www.w3.org/2001/xml-events", "scroll"}</p> <p>Deprecated backwards-compatibility alias: {http://www.w3.org/2001/xml-events", "SVGScroll"} (see Notes below).</p>	<p>Occurs when a document view is being shifted along the X or Y or both axis, either through a direct user interaction or any change on the 'currentTranslate' property available on SVGSVGElement interface. This event is only applicable to 'svg' elements and is dispatched after the shift modification has taken place. The target of the event is the 'svg' element.</p>	<p>HTMLEvent</p>	<p>scroll</p>	<p>No</p>	<p>Event</p>
<p>{http://www.w3.org/2001/xml-events", "zoom"}</p> <p>Deprecated backwards-compatibility alias: {http://www.w3.org/2001/xml-events", "SVGZoom"} (see Notes below).</p>	<p>Occurs when the zoom level of a document view is being changed, either through a direct user interaction or any change to the 'currentScale' property available on SVGSVGElement interface. This event is only applicable to 'svg' elements and is dispatched after the zoom level modification has taken place. The target of the event is the 'svg' element.</p>	<p>DOM3's SVG Events</p>	<p>zoom</p>	<p>No</p>	<p>Event</p>
<p>{http://www.w3.org/2001/xml-events", "beginEvent"}</p>	<p>Occurs when an animation element begins. For details, see the description of the Events and event model in SMIL 2.0.</p>	<p>DOM3's Timing Events</p>	<p>beginEvent</p>	<p>No</p>	<p>TimeEvent</p>
<p>{http://www.w3.org/2001/xml-events", "endEvent"}</p>	<p>Occurs when an animation element ends. For details, see the description of the Events and event model in SMIL 2.0.</p>	<p>DOM3's Timing Events</p>	<p>endEvent</p>	<p>No</p>	<p>TimeEvent</p>

{ http://www.w3.org/2001/xml-events ", " repeatEvent "}	Occurs when an animation element repeats. It is raised each time the element repeats, after the first iteration. For details, see the description of the Events and event model in SMIL 2.0 .	DOM3's Timing Events	repeat	No	TimeEvent
{ http://www.w3.org/2001/xml-events ", " wheel "}	Occurs when a rotational input device has been activated.	UIEvent	none	Yes	WheelEvent
{ http://www.w3.org/2000/svg ", " preload "}	A load operation has begun.	none	none	No	ProgressEvent
{ http://www.w3.org/2000/svg ", " loadProgress "}	Progress has occurred in loading a given resource.	none	none	No	ProgressEvent
{ http://www.w3.org/2000/svg ", " postload "}	A load operation has completed.	none	none	No	ProgressEvent
{ http://www.w3.org/2001/xml-events ", " timer "}	Occurs when the specified timer interval has elapsed for a timer. This event is triggered only by 'enabled' timers in the current global execution context of the SVG document (i.e. for timers which have been instantiated via the SVGGlobal interface and started via the start () method of the SVGTimer interface).	none	none	No	Event
{ http://www.w3.org/2000/svg ", " connectionConnected "}	Occurs when a connection has been established. No context information is available.	none	none	No	ConnectionEvent
{ http://www.w3.org/2000/svg ", " connectionClosed "}	Occurs when a connection has been closed. No context information is available	none	none	No	ConnectionEvent
{ http://www.w3.org/2000/svg ", " connectionError "}	Occurs when an error happens during the lifetime of a connection. Additional context information is available in the errorCode field.	none	none	No	ConnectionEvent

{"http://www.w3.org/2000/svg", "connectionDataSent"}	Occurs when data has been successfully transmitted. No context information is available.	none	none	No	ConnectionEvent
{"http://www.w3.org/2000/svg", "connectionDataReceived"}	Occurs when data has been received on the connection. Additional context information is available on the receivedData field.	none	none	No	ConnectionEvent

Notes:

- Unqualified event names, including SVG 1.1 event names, are assumed to be in the "http://www.w3.org/2001/xml-events" namespace. This allows SVG 1.1 content (which did not have a notion of namespaced events) to be upwardly compatible with SVG 1.2 (which adds a notion of namespaced events). Therefore, the SVG 1.1 "SVGZoom" event becomes the {"http://www.w3.org/2001/xml-events", "SVGZoom"} event in SVG 1.2.
- In order to unify event names with other W3C languages, SVG 1.2 deprecates some of the SVG 1.1 event names. (The term "deprecate" in this case means that user agents which are compatible with both SVG 1.1 and SVG 1.2 must support both the old deprecated event names and the new event names, but an SVG 1.2-only user agent should support only the new event names. Content creators who are making content that targets SVG 1.2 should use the new event names, not the deprecated event names.) Here are the specifics:
 - "SVGLoad" event is deprecated in favor of {"http://www.w3.org/2001/xml-events", "load"}
 - "SVGResize" event is deprecated in favor of {"http://www.w3.org/2001/xml-events", "resize"}
 - "SVGScroll" event is deprecated in favor of {"http://www.w3.org/2001/xml-events", "scroll"}
 - "SVGZoom" event is deprecated in favor of {"http://www.w3.org/2001/xml-events", "zoom"}

In cases where the event name from SVG 1.1 differs from the DOM3 event name, but the SVG event name is more user-friendly (e.g., "focusin" is more user friendly than "DOMFocusIn"), the SVG 1.1 event name is not deprecated but instead retained as an alias for the DOM3 event name. Therefore the following aliases shall be available:

- {"http://www.w3.org/2001/xml-events", "focusin"} is equivalent to {"http://www.w3.org/2001/xml-events", "DOMFocusIn"}
 - {"http://www.w3.org/2001/xml-events", "focusout"} is equivalent to {"http://www.w3.org/2001/xml-events", "DOMFocusOut"}
 - {"http://www.w3.org/2001/xml-events", "activate"} is equivalent to {"http://www.w3.org/2001/xml-events", "DOMActivate"}
 - In almost all cases, the event identifiers for SMIL timing attributes (e.g., the 'begin' and 'end' attributes on animation elements) are required to be the localname for the events, where the assumed namespace for the event is "http://www.w3.org/2001/xml-events". For example, to indicate that an animation should begin with a "click" event, then the begin attribute would be specified as begin="click". Some exceptions to this rule are necessary for SVG 1.1 backwards compatibility and to make it easier to integrate SVG with other W3C languages. The following events shall use the following strings rather than their localname as their event identifier for SMIL timing attributes:
 - The DOMFocusIn event uses the string "focusin"
 - The DOMFocusOut event uses the string "focusout"
 - The DOMActivate event uses the string "activate"
- For backwards-compatibility and to make it easier to integrate SVG with other W3C languages, with SVG 1.2 SMIL timing attributes must accept the following aliases for event identifiers:
- "load" and "SVGLoad" are aliases for each other
 - "resize" and "SVGResize" are aliases for each other
 - "scroll" and "SVGScroll" are aliases for each other
 - "zoom" and "SVGZoom" are aliases for each other

A **SVGLoad** event must be dispatched only to the element to which the event applies; it must not be dispatched to its ancestors. For example, if an **'image'** element and its parent **'g'** element both have event listeners for **SVGLoad** events, when the **'image'** element has been loaded, only its event listener will be invoked. (The **'g'** element's event listener will indeed get invoked, but the invocation will happen when the **'g'** itself has been loaded.)

Details on the parameters passed to event listeners for the event types from DOM2 can be found in the DOM2 specification. For other event types, the parameters passed to event listeners are described elsewhere in this specification.

13.3 User interface events

On user agents which support interactivity, it is common for authors to define SVG documents such that they are responsive to user interface events. Among the set of possible user events are [pointer events](#), keyboard events, and document events.

In response to user interface (UI) events, the author might start an animation, perform a hyperlink to another Web page, highlight part of the document (e.g. change the color of the graphics elements which are under the pointer), initiate a "roll-over" (e.g., cause some previously hidden graphics elements to appear near the pointer) or launch a script which communicates with a remote database.

The following example shows the use of an activate event to trigger an ECMAScript event handler:

```

Example: activate.svg

<?xml version="1.0" encoding="UTF-8"?>
<svg width="6cm" height="5cm" viewBox="0 0 600 500" xmlns="http://www.w3.org/2000/svg" version="1.2"
  baseProfile="tiny" xmlns:ev="http://www.w3.org/2001/xml-events">
  <desc>Example: invoke an ECMAScript function from an activate event </desc>
  <!-- ECMAScript to change the radius -->
  <script type="application/ecmascript">
    <![CDATA[
      function change(evt) {
        var circle = evt.target;
        var currentRadius = circle.getFloatTrait("r");
        if (currentRadius == 100)
          circle.setFloatTrait("r", currentRadius*2);
        else
          circle.setFloatTrait("r", currentRadius*0.5);
      }
    ]]>
  </script>

```

```

    }
  ]]>
  </script>
  <!-- Act on each activate event -->
  <circle cx="300" cy="225" r="100" fill="red">
    <handler type="application/ecmascript" ev:event="activate"> change(evt); </handler>
  </circle>
  <text x="300" y="480" font-family="Verdana" font-size="35" text-anchor="middle"> Activate the
    circle to change its size </text>
</svg>

```

For all UI event-related features defined as part of the SVG language via [XML Events](#) or [animation](#), the event model corresponds to the *event bubbling* model described in DOM3 [\[DOM3-EVBUBBLE\]](#). The *event capture* model from DOM3 is not supported.

13.4 Pointer events

Note: The W3C's Web Content Accessibility Guidelines ([WCAG](#)) advise content creators to create [device-independent content](#); in particular, content should not require that the user has access to a pointer device.

User interface events that occur because of user actions performed on a pointer device are called pointer events.

Many systems support pointer devices such as a mouse, trackball, stylus or joystick. On systems which use a mouse, pointer events consist of actions such as mouse movements and mouse clicks. On systems with a different pointer device, the pointing device often emulates the behavior of the mouse by providing a mechanism for equivalent user actions, such as a button to press which is equivalent to a mouse click.

One difference between stylus-based pointers and mouse-based pointers is that for a mouse, the cursor always has a position; for a stylus which may be lifted, the cursor may only have a position when the stylus is tapped on the screen. Thus, content which assumes that all pointer devices will generate `mouseenter` and `mouseleave` events will not work on all devices.

13.5 Text events

User interface events that occur because of user actions that generate text are called text events. They are usually generated by a keyboard, but can also be generated by a different input method such as an IME (for Japanese text, for example), by speech input, etc. DOM 3 Text Events only requires that a string of Unicode characters is returned, making it independent of the input device used.

13.6 Key events

Note: The W3C's Web Content Accessibility Guidelines ([WCAG](#)) advise content creators to create [device-independent content](#); in particular, content should not require that the user has access to a (full-size) keyboard.

User interface events that occur because of user actions that generate key presses (as opposed to text - for example, function keys, key presses for a game, etc) are called key events.

13.7 Event Dispatching

For each pointer event, text event or key event, the SVG user agent determines the *target object* of a given event. The *target object* must be the topmost graphics element or [SVGElementInstance](#) object whose relevant graphical content is under the pointer (for pointer events) or has focus (for text and key events) at the time of the event. (See property ['pointer-events'](#) for a description of how to determine whether an element's relevant graphical content is under the pointer, and thus in which circumstances that graphic element can be the target object for a pointer event.) When an element is not displayed (i.e., when the ['display'](#) property on that element or one of its ancestors has a value of `none`), that element must not be the target of pointer events. If the user agent cannot find a graphics element or [SVGElementInstance](#) object to be the target object (e.g., the pointer event occurs over the viewport background), then the user agent must set the target object to be the ['svg'](#) element that is the root of the current SVG document fragment.

The decision on whether to dispatch the event to the *target object* or to one of the target elements ancestors shall depend on the following:

- If there is no target object, the event is not dispatched.
- Otherwise, if the target object has an appropriate event handler for the given event, the event is dispatched to the target object.
- Otherwise, each ancestor of the target object (starting with its immediate parent) is checked to see if it has an appropriate event handler. If an ancestor is found with an appropriate event handler, the event is dispatched to that ancestor element.
- Otherwise, the event is discarded.

When event bubbling [\[DOM2-EVBUBBLE\]](#) is active, bubbling occurs up to all direct ancestors of the target object. Descendant elements receive events before their ancestors. Thus, if a ['path'](#) element is a child of a ['g'](#) element and they both have event listeners for `click` events, then the event will be dispatched to the ['path'](#) element before the ['g'](#) element.

After an event is initially dispatched to a particular element, unless an appropriate action has been taken to prevent further processing, the event must be passed to the appropriate event handlers (if any) for that element's ancestors (in the case of event bubbling) for further processing.

13.8 Processing order for user interface events

The processing order for user interface events shall be as follows:

- Event handlers assigned to the topmost graphics element under the pointer (and the various ancestors of that graphics element via potential event bubbling) receive the event first. If none of the activation event handlers take an explicit action to prevent further processing of the given event, then the event is passed on for:
- (For those user interface events which invoke hyperlinks, such as mouse clicks in some user agents) Link processing. If a hyperlink is invoked in response to a user interface event, the hyperlink typically will disable further activation event processing (e.g., often, the link will define a hyperlink to another Web page). If link processing does not disable further processing of the given event, then the event is passed on for:
- (For those user interface events which can select text, such as mouse clicks and drags on `'text'` elements) Text selection processing. When a text selection operation occurs, typically it will disable further processing of the given event; otherwise, the event is passed on for:
- Document-wide event processing, such as user agent facilities to allow zooming and panning of an SVG document fragment.

Various elements in SVG create shadow content which can be the target of user interface events. The following is a list of elements that can create shadow content which can be the target of user interface events:

- The **use** element
- The **animation** element

For these situations, user interface events within the shadow content shall participate in the processing of user interface events in the same manner as if the shadow content were part of the main document. In other words, if shadow content contains a graphics element that renders above other content at the current pointer location, then it represents the topmost graphics element and will receive the pointer events before other elements. In this case, the user interface events bubble up through the target's ancestors, and then across the document border into the referencing element, and then through the ancestors of the referencing element. This process continues as necessary if there are multiple levels of nested shadow trees.

13.9 The 'pointer-events' property

In different circumstances, authors may want to control under what circumstances particular graphic elements can become the target of pointer events. For example, the author might want a given element to receive pointer events only when the pointer is over the stroked perimeter of a given shape. In other cases, the author might want a given element to ignore pointer events under all circumstances so that graphical elements underneath the given element will become the target of pointer events.

For example, suppose a circle with a **'stroke'** of **red** (i.e., the outline is solid red) and a **'fill'** of **none** (i.e., the interior is not painted) is rendered directly on top of a rectangle with a **'fill'** of **blue**. The author might want the circle to be the target of pointer events only when the pointer is over the perimeter of the circle. When the pointer is over the interior of the circle, the author might want the underlying rectangle to be the target element of pointer events.

The **'pointer-events'** property specifies under what circumstances a given graphics element can be the target element for a pointer event. It affects the circumstances under which the following are processed:

- user interface events such as key press.
- hyperlinks (see [Links out of SVG content: the 'a' element](#))

'pointer-events'

Value: visiblePainted | visibleFill | visibleStroke | visible |

painted | fill | stroke | all | none | inherit

Initial: visiblePainted

Applies to: [graphics elements](#)

Inherited: yes

Percentages: N/A

Media: visual

Animatable: yes

Computed value: Specified value, except inherit

visiblePainted

The given element must only be a target element for pointer events when the **'visibility'** property is set to **visible** and when the pointer is over a "painted" area. The pointer is over a painted area if it is over the interior (i.e., fill) of the element and the **'fill'** property is set to a value other than 'none' or it is over the perimeter (i.e., stroke) of the element and the **'stroke'** property is set to a value other than 'none'.

visibleFill

The given element must only be a target element for pointer events when the **'visibility'** property is set to **visible** and when the pointer is over the interior (i.e., fill) of the element. The value of the **'fill'** property does not effect event processing.

visibleStroke

The given element must only be a target element for pointer events when the **'visibility'** property is set to **visible** and when the pointer is over the perimeter (i.e., stroke) of the element. The value of the **'stroke'** property does not effect event processing.

visible

The given element must only be a target element for pointer events when the **'visibility'** property is set to **visible** and the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element. The values of the **'fill'** and **'stroke'** do not effect event processing.

painted

The given element must only be a target element for pointer events when the pointer is over a "painted" area. The pointer is over a painted area if it is over the interior (i.e., fill) of the element and the **'fill'** property is set to a value other than 'none' or it is over the perimeter (i.e., stroke) of the element and the **'stroke'** property is set to a value other than 'none'. The value of the **'visibility'** property does not effect event processing.

fill

The given element must only be a target element for pointer events when the pointer is over the interior (i.e., fill) of the element. The values of the **'fill'** and **'visibility'** properties do not effect event processing.

stroke

The given element must only be a target element for pointer events when the pointer is over the perimeter (i.e., stroke) of the element. The values of the **'stroke'** and **'visibility'** properties do not effect event processing.

all

The given element must be a target element for pointer events whenever the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element. The values of the **'fill'**, **'stroke'** and **'visibility'** properties do not effect event processing.

none

The given element must not receive pointer events.

For text elements, hit detection shall be performed on a character cell basis:

- The value **visiblePainted** means that the text string can receive events anywhere within the character cell if either the **'fill'** property is set to a value other than **none** or the **'stroke'** property is set to a value other than **none**, with the additional requirement that the **'visibility'** property is set to **visible**.
- The values **visibleFill**, **visibleStroke** and **visible** are equivalent and indicate that the text string can receive events anywhere within the character cell if the **'visibility'** property is set to **visible**. The values of the **'fill'** and **'stroke'** properties do not effect event processing.
- The value **painted** means that the text string can receive events anywhere within the character cell if either the **'fill'** property is set to a value other than **none** or the **'stroke'** property is set to a value other than **none**. The value of the **'visibility'** property does not effect event processing.
- The values **fill**, **stroke** and **all** are equivalent and indicate that the text string can receive events anywhere within the character cell. The values of the **'fill'**, **'stroke'** and **'visibility'** properties do not effect event processing.
- The value **none** indicates that the given text does not receive pointer events.

For raster images, hit detection shall either be performed on a whole-image basis (i.e., the rectangular area for the image is one of the determinants for whether the image receives the event) or on a per-pixel basis (i.e., the alpha values for pixels under the pointer help determine whether the image receives the event). The following rules must be adhered to:

- The value **visiblePainted** means that the raster image can receive events anywhere within the bounds of the image if any pixel from the raster image which is under the pointer is not fully transparent, with the additional requirement that the **'visibility'** property is set to **visible**.
- The values **visibleFill**, **visibleStroke** and **visible** are equivalent and indicate that the image can receive events anywhere within the rectangular area for the image if the **'visibility'** property is set to **visible**.
- The value **painted** means that the raster image can receive events anywhere within the bounds of the image if any pixel from the raster image which is under the pointer is not fully transparent. The value of the **'visibility'** property does not effect event processing.
- The values **fill**, **stroke** and **all** are equivalent and indicate that the image can receive events anywhere within the rectangular area for the image. The value of the **'visibility'** property does not effect event processing.
- The value **none** indicates that the image does not receive pointer events.

Note that for raster images, the values of properties **'fill-opacity'**, **'stroke-opacity'**, **'fill'** and **'stroke'** do not effect event processing.

13.10 Magnification and panning

Magnification represents a complete, uniform transformation on an SVG document fragment, where the magnify operation scales all graphical elements by the same amount. A magnify operation has the effect of a supplemental scale and translate transformation placed at the rootmost level on the SVG document fragment (i.e., outside the [rootmost svg element](#)).

Panning represents a translation (i.e., a shift) transformation on an SVG document fragment in response to a user interface action.

SVG user agents that operate in interaction-capable user environments are required to support the ability to magnify and pan.

The **'svg'** element in an SVG document fragment has attribute **zoomAndPan**, which takes the possible values of *disable* and *magnify*, with the default being *magnify*. This is intended for applications where SVG is used for both the content and for the user interface, eg a mapping application. The default zoom might move critical user interface components from view, confusing the user; disabling the default zoom and pan while providing zoom and pan controls for a smaller content area would give a better user experience.

If *disable*, the user agent shall in its default interaction mode disable any magnification and panning controls and not allow the user to magnify or pan on the given document fragment. The user agent may provide another mode which continues to allow zoom and pan at user option.

If *magnify*, in environments that support user interactivity, the user agent shall provide controls to allow the user to perform a "magnify" operation on the document fragment.

Animatable: no.

13.11 Element focus

13.11.1 The focusable attribute

In many cases, such as text editing, the user is required to place focus on a particular element, ensuring that input events, such as keyboard input, are sent to that element.

All renderable elements are required to be able to accept focus if specified by the author, including **container elements** (except **'defs'**), **graphics elements**, **'tspan'** and **'foreignObject'**. A focusable **container element** may contain focusable descendants.

Attribute definition:

focusable = "true" | "false" | "auto"

Defines if an element can get keyboard focus (i.e. receive keyboard events) and be a target for field-to-field navigation actions. (Note: in some environments, field-to-field navigation can be accomplished with the tab key.)

The attribute value can be either of the following:

"true"

The element is keyboard-aware and must be treated as any other UI component that can get focus.

"false"

The element is not focusable.

"auto"

The default value. Equivalent to **false**, except that it must be treated like **true** for the following cases:

- the **a** element
- the **text** and **textArea** elements with **editable** set to **true**
- elements that are the target of an **animation** whose **begin or end criteria** is triggered by the following **user interface events**: focusin, focusout, activate
- elements that are the **target** or **observer** of a **listener** element that is registered on one of the following **user interface events**: focusin, focusout, activate

Animatable: Yes

13.12 Navigation

13.12.1 Navigation behavior

System-dependent input facilities (e.g., the tab key on most desktop computers) should be supported to allow navigation between elements that can obtain focus (i.e. elements for which the value of the **focusable** attribute evaluates to "true").

The document has the concept of a focus ring, which is the order in which elements obtain focus. By default the focus ring shall be obtained using document order. All focusable elements must be part of the default focus ring. A document's focus ring includes any focusable objects within shadow trees for **'use'** elements. The **focus attributes** may be used to modify the default focus ring.

The SVG language supports a flattened notion of field navigation between focusable elements where an author may define field navigation between any two focusable elements defined within a given SVG document without regard to document hierarchy. For example:

```
<rect id="r1" focusable="true".../>
<g id="g1" focusable="true">
  <circle id="c1" focusable="true".../>
</g>
```

In the above example, the author may specify field-to-field navigation such the user can navigate directly from any of the three elements. Thus, assuming a desktop computer which uses the tab key for field navigation, the author may specify focus navigation order such that the tab key takes the user from r1 to c1 to g1.

When navigating to an element that is not visible on the canvas the following rules shall apply:

- The UA must not navigate to an element which has `display='none'`. (An element which has `display='none'` is not focusable.)
- The UA must allow navigation to elements which are not visible (i.e. which has a 100% transparency or which is hidden by another element).
- The UA must allow navigation to elements which are located outside of the current viewport. In this case it is recommended that the UA SHOULD change the current viewport so that the focused element becomes visible.

SVG's flattened notion of field navigation shall extend to referenced content and shadow trees as follows:

- Focusable elements within the content referenced by a use element participate in field navigation operations using the flattened focus model. (Note: If a referenced group contains a focusable element, and that group is referenced by two use elements, then the document will have two separate focusable fields, not just one.)
- If an animation element references an SVG document, then all of the focusable fields defined within the referenced SVG document participate in field navigation operations using the flattened focus model.

Focus navigation shall behave as specified:

1. When the document is loaded the focus is first offered to the User Agent.
2. Once the User Agent releases focus, then focus passes to the entity that first matches the following criteria:
 1. the [rootmost svg element](#) if it is focusable,
 2. the element referenced by the 'nav-next' attribute on the [rootmost svg element](#), if the attribute is present,
 3. the first focusable element in the document starting from the [rootmost svg element](#),
 4. the User Agent
3. If the focus is held by an element in the document, then the next element in navigation order shall be the entity that first matches the following criteria:
 1. the element referenced by the 'nav-next' attribute on the focused element,
 2. the next focusable element in document order,
 3. the User Agent
4. If the focus is held by an element in the document, then the previous element in navigation order shall be the entity that first matches the following criteria:
 1. the element referenced by the 'nav-prev' attribute on the focused element,
 2. the previous focusable element in document order,
 3. the User Agent

For stand-alone SVG documents, the User Agent must always have a currently focused object. If focus is not held by any object in the document tree, the User Agent must give focus to the [SVGDocument](#) object.

For SVG documents which are referenced by a non-SVG host document (e.g., XHTML), the SVG document may participate within the host document's focus ring, which would allow direct navigation from an SVG focusable element onto a focusable element within the host document. Other compound document specifications may define supplemental SVG focus navigation rules for situations when SVG content is used as a component within a compound document.

13.12.2 Specifying navigation

Navigation order can be specified using the attributes defined below.

nav-next

The 'nav-next' attribute specifies the next element in the focus ring.

Attribute definition:

nav-next = url (<Local IRI Reference>) | "auto" | "self"

Specifies the next element in the focus ring.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the next direction is triggered.

"auto"

The default value. Means that the behavior shall be as if the attribute was not specified (navigation must follow the rules specified in [Navigation behavior](#)).

"self"

The focus must stay on the element itself.

Animatable: Yes

nav-prev

The 'nav-prev' attribute specifies the previous element in the focus ring.

Attribute definition:

nav-prev = url (<Local IRI Reference>) | "auto" | "self"

Specifies the previous element in the focus ring.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the previous direction is triggered.

- "auto" The default value. Means that the behavior shall be as if the attribute was not specified (navigation must follow the rules specified in [Navigation behavior](#)).
- "self" The focus must stay on the element itself.

[Animatable](#): Yes

nav-up

The 'nav-up' attribute specifies the element to be focused if a navigation in the upward direction is triggered.

Attribute definition:

nav-up = url (<Local IRI Reference>) | "auto" | "self"
Specifies the element to receive focus in the upward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the upward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

[Animatable](#): Yes

nav-up-right

The 'nav-up-right' attribute specifies the element to be focused if a navigation in the upward-rightward direction is triggered.

Attribute definition:

nav-up-right = url (<Local IRI Reference>) | "auto" | "self"
Specifies the element to receive focus in the upward-rightward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the upward-rightward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

[Animatable](#): Yes

nav-right

The 'nav-right' attribute specifies the element to be focused if a navigation in the rightward direction is triggered.

Attribute definition:

nav-right = url (<Local IRI Reference>) | "auto" | "self"
Specifies the element to receive focus in the rightward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the rightward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

[Animatable](#): Yes

nav-down-right

The 'nav-down-right' attribute specifies the element to be focused if a navigation in the downward-rightward direction is triggered.

Attribute definition:

nav-down-right = url (<Local IRI Reference>) | "auto" | "self"
Specifies the element to receive focus in the downward-rightward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the downward-rightward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

[Animatable](#): Yes

nav-down

The 'nav-down' attribute specifies the element to be focused if a navigation in the downward direction is triggered.

Attribute definition:

nav-down = url (<Local IRI Reference>) | "auto" | "self"

Specifies the element to receive focus in the downward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the downward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

Animatable: Yes

nav-down-left

The 'nav-down-left' attribute specifies the element to be focused if a navigation in the downward-leftward direction is triggered.

Attribute definition:

nav-down-left = url (<Local IRI Reference>) | "auto" | "self"

Specifies the element to receive focus in the downward-leftward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the downward-leftward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

Animatable: Yes

nav-left

The 'nav-left' attribute specifies the element to be focused if a navigation in the leftward direction is triggered.

Attribute definition:

nav-left = url (<Local IRI Reference>) | "auto" | "self"

Specifies the element to receive focus in the leftward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the leftward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

Animatable: Yes

nav-up-left

The 'nav-up-left' attribute specifies the element to be focused if a navigation in the upward-leftward direction is triggered.

Attribute definition:

nav-up-left = url (<Local IRI Reference>) | "auto" | "self"

Specifies the element to receive focus in the upward-leftward direction.

The attribute value can be either of the following:

url (<Local IRI Reference>)

Specifies the element that must receive focus when navigation in the upward-leftward direction is triggered.

"auto"

The default value. Means that the behavior is left up to the User Agent.

"self"

The focus must stay on the element itself.

Animatable: Yes

Example: [navigation.svg](#)

```
<?xml version="1.0"?>
<svg viewBox="0 0 220 40"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
```



```

version="1.2" baseProfile="tiny">
<desc>An example which illustrates the use of nav-* attributes</desc>
<!-- List of available channels -->
<rect x="0" y="0" width="100" height="20" fill="#fb0" stroke="#000" stroke-width="2" />
<text x="50" y="13" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >Channel 1</text>
<rect x="0" y="20" width="100" height="20" fill="#fb0" stroke="#000" stroke-width="2" />
<text x="50" y="33" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >Channel 2</text>
<rect x="0" y="40" width="100" height="20" fill="#fb0" stroke="#000" stroke-width="2" />
<text x="50" y="53" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >Channel 3</text>
<!-- List of programs for channel nb 1 -->
<g xml:id="Chan1Prog1" focusable="true" nav-left="self" nav-right="url(#Chan1Prog2)"
nav-up="self" nav-down="url(#Chan2Prog1)">
<rect x="100" y="0" width="100" height="20" fill="#fd0" stroke="#000" stroke-width="2">
<set attributeName="fill" begin="Chan1Prog1.focusin" end="Chan1Prog1.focusout" to="#fa0"/>
</rect>
<text x="150" y="13" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >Weather</text>
</g>
<g xml:id="Chan1Prog2" focusable="true" nav-left="url(#Chan1Prog1)" nav-right="url(#Chan1Prog3)"
nav-up="self" nav-down="auto">
<rect x="200" y="0" width="120" height="20" fill="#fd0" stroke="#000" stroke-width="2">
<set attributeName="fill" begin="Chan1Prog2.focusin" end="Chan1Prog2.focusout" to="#fa0"/>
</rect>
<text x="260" y="13" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >The news</text>
</g>
<g xml:id="Chan1Prog3" focusable="true" nav-left="url(#Chan1Prog2)" nav-right="self" nav-up="self"
nav-down="auto" nav-next="self">
<rect x="320" y="0" width="120" height="20" fill="#fd0" stroke="#000" stroke-width="2">
<set attributeName="fill" begin="Chan1Prog3.focusin" end="Chan1Prog3.focusout" to="#fa0"/>
</rect>
<text x="380" y="13" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >Football</text>
</g>
<!-- List of programs for channel nb 2 -->
<g xml:id="Chan2Prog1" focusable="true" nav-left="self" nav-right="auto" nav-up="url(#Chan1Prog1)"
nav-down="auto" nav-prev="url(#Chan1Prog1)" nav-next="auto">
<rect x="100" y="20" width="150" height="20" fill="#fd0" stroke="#000" stroke-width="2">
<set attributeName="fill" begin="Chan2Prog1.focusin" end="Chan2Prog1.focusout" to="#fa0"/>
</rect>
<text x="175" y="33" font-size="8" font-family="Arial Black"
fill="#fff" text-anchor="middle" >Long Movie</text>
</g>
</svg>

```

This example illustrates how it is possible for an author to control the focus order between several focusable elements displayed on the canvas.

On a device which provides a 2-way navigation system (a TAB mechanism for instance), here are the interesting behaviors:

- Whenever the focus is located on a program which is at the beginning of the timeline of a given channel, there are 3 options when the user wants to go to the previous focusable item (i.e., the user presses the "Reverse-Tab" key on most desktop computers):
 - option 1: the focus goes up to the first program of the previous channel
 - option 2: the focus goes up to the last program of the previous channel
 - option 3: the focus remains at the same place

Here, in this example, for channel 2, because there is `nav-prev="url(#Chan1Prog1)"` attribute in element 'g' with `id="Chan2Prog1"`, option 1 will be applied.

In order to apply option 2, we could have set `nav-prev="url(#Chan1Prog3)"` instead.

In order to apply option 3, we could have set `nav-prev="self"` instead.

- Whenever the focus is located on a program which is at the end of the timeline of a given channel, there are 2 options when the user wants to go to the next focusable item (i.e., the user presses the "Tab" key on most desktop computers):
 - option 1: the focus goes down to the first program of the next channel
 - option 2: the focus remains at the same place

Here, in this example, for channel 1, because there is `nav-next="self"` attribute in element 'g' with `id="Chan1Prog3"`, option 2 will be applied.

In order to apply option 1, we could have set `nav-next="url(#Chan2Prog1)"` instead.

- Whenever the focus is located on "Chan2Prog1" container, if the user wants to go to the next focusable element, the concept of a focus ring will apply because of value `nav-next="auto"`. Here, according to the focus ring navigation rules, focus will be offered to the User Agent because there is no more focusable element in the document order.

On a device which provides a 4-way navigation system (i.e. a joystick for instance), here are the interesting behaviors:

- Whenever the focus is located at the beginning of the timeline of a given channel, when the user wants to go "Left", focus remains on the same element because both element 'g' with `id="Chan1Prog1"` and element 'g' with `id="Chan2Prog1"` have `nav-left="self"`.
- Whenever the focus is located on "Chan1Prog1" container, if the user wants to go "Right", the focus will be put on container element "Chan1Prog2" because of the `nav-right="url(#Chan1Prog2)"` value. But, because some part of "Chan1Prog2" bounding box is outside of the current viewport, the UA SHOULD change the current viewport so that the new focused element becomes visible.



Before element #Chan1Prog2 receives focus

After element #Chan1Prog2 receives focus (UA scrolls automatically)

- On element 'g' with id="Chan2Prog1", there is a value `nav-right="auto"`. This value is the default one for `nav-*` attributes and therefore the behavior is the same as if no `'nav-right'` attribute was defined. This value "auto" means that it's up to the User Agent to choose which focusable element should receive focus when the user wants to go 'right'.

13.12.3 Specifying Focus Highlighting

Automated highlighting upon focus can be specified using the `focusHighlight` attribute. This hint indicates whether the User Agent should highlight an element on focus. The highlighting method is implementation dependent and the User Agent should pick a method that works well for varying content. This attribute is available on all graphical and container elements.

`focusHighlight = "auto" | "none"`

Specifies whether a User Agent should highlight an element on focus.

The attribute value can be either of the following:

"auto"

This indicates that the element should be highlighted on focus. The highlighting method is left up to the User Agent. If the value is not specified the effect must be as if "auto" was specified.

"none"

The User Agent should not highlight this element on focus.

Animatable: No

Example: [focusHighlight.svg](#)

```
<?xml version="1.0"?>
<svg viewBox="0 0 210 80"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  version="1.2" baseProfile="tiny">

  <desc>An example which illustrates the use of focusHighlight attribute</desc>

  <text x="5" y="10">Do you want to validate transaction ?</text>
  <text x="5" y="25">You may read <a xlink:href="http://www.example.org/pay"
    >this</a> and <a xlink:href="http://www.example.org/infos">that</a>
  </text>

  <a xml:id="ValidateButton" transform="translate(5,40)" focusHighlight="none" xlink:href="validate.htm">
    <rect x="0" y="0" width="90" height="20" fill="#0f0" stroke="#000" stroke-width="2">
      <set attributeName="fill" begin="ValidateButton.focusin" end="ValidateButton.focusout" to="#0a0"/>
    </rect>
    <text x="45" y="13" font-size="8" font-family="Arial Black" letter-spacing="0.44em"
      fill="#000" text-anchor="middle">Validate</text>
  </a>
  <a xml:id="AbortButton" transform="translate(100,40)" focusHighlight="none" xlink:href="abort.htm">
    <rect x="0" y="0" width="90" height="20" fill="#f00" stroke="#000" stroke-width="2">
      <set attributeName="fill" begin="AbortButton.focusin" end="AbortButton.focusout" to="#a00"/>
    </rect>
    <text x="45" y="13" font-size="8" font-family="Arial Black"
      fill="#000" text-anchor="middle">Abort</text>
  </a>
</svg>
```

Do you want to validate transaction ?
You may read this and that



In the above SVG example:

- highlight of the focus on the first two textual links is left up to the User Agent (underline the text, highlight of the bounding box, change color of the text, ...) since the default value is `focusHighlight="auto"`. This text may have been retrieved from a database where there may be no notion of graphical styling or no way to know in advance the kind of focusable elements it contains, therefore the author doesn't handle focus highlight on that part of the document.
- highlight of the focus on the 2 graphical buttons is designed by the author and therefore the User Agent doesn't need to highlight it as well. Therefore, `focusHighlight="none"` is used to disable the default focus highlight behavior.

13.12.4 Obtaining and listening to focus programmatically

When the user agent gives an element focus it receives a (`"http://www.w3.org/2001/xml-events", "DOMFocusIn"`) event which has the the new focused object as the event target and a (`"http://www.w3.org/2001/xml-events", "DOMFocusOut"`) event which has the previously focused object as the event target.

The [SVGSVGElement](#) interface has a [setFocus\(DOMObject object\)](#) method that puts the focus on the requested object. Calling setFocus with an element that is not focusable causes focus to stay on the currently focused object.

The [SVGSVGElement](#) interface has a [moveFocus\(short motionType\)](#) which moves current focus to a different object based on the value of motionType.

User agents which support pointer devices such as a mouse MUST allow users to put focus onto focusable elements. For example, it should be possible to click on a focusable element in order to give focus.

Empty text fields in SVG theoretically take up no space, but they have a point or zero-width line segment that represents the location of the empty text field. User agents should allow users with pointer devices to put focus into empty text fields by initiating a select action (e.g., a mouse click) at the location of the empty text field.

An author may change the field navigation order from a script by using the [setTrait](#) method to change the current value of focus* attributes on a given element (see Example below).

Example: changeNavigationOrder.svg

```
<?xml version="1.0"?>

<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:ev="http://www.w3.org/2001/xml-events">

  <desc>An example of how to change the navigation order from script by
  changing nav-* attribute values. In this example, "myRect2" disappears between 10 and 20 sec
  and is replaced by the "myRectNew" rectangle during this period. Consequently, the navigation order
  must be changed accordingly during this period and we must re-established initial order after 20s.</desc>

  <rect xml:id="myRect1" x="10" y="20" width="100" height="50" fill="red" focusable="true" nav-right="url(#myRect2)">
    <set begin="focusin" end="focusout" attributeName="fill" to="purple"/>
  </rect>

  <rect xml:id="myRect2" x="120" y="20" width="100" height="50" fill="red" focusable="true"
  nav-right="url(#myRect3)" nav-left="url(#myRect1)">
    <set begin="focusin" end="focusout" attributeName="fill" to="purple"/>
    <set begin="0" end="10" attributeName="display" to="inline"/>
    <set begin="10" end="20" attributeName="display" to="none"/>
    <set begin="20" attributeName="display" to="inline"/>
  </rect>

  <rect xml:id="myRect3" x="230" y="20" width="100" height="50" fill="red" focusable="true" nav-left="url(#myRect2)">
    <set begin="focusin" end="focusout" attributeName="fill" to="purple"/>
  </rect>

  <rect xml:id="myRectNew" x="120" y="20" width="100" height="50" fill="blue" focusable="true" nav-right="url(#myRect3)"
  nav-left="url(#myRect1)" display="none" >
    <set xml:id="myRectNewFillAnim" begin="focusin" end="focusout" attributeName="fill" to="black"/>
    <set xml:id="myRectNewDisplayAnim" begin="10" end="20" attributeName="display" to="inline"/>
  </rect>

  <!-- register a listener for myRectNew.beginEvent event -->
  <ev:listener event="beginEvent" observer="myRectNew" handler="#myAnimationHandler" />
  <ev:listener event="endEvent" observer="myRectNew" handler="#myAnimationHandler" />

  <!-- handler which is called when myRect2 is replaced by myRectNew -->
  <handler xml:id="myNavigationHandler" type="application/ecascript"><![CDATA[
    var myRect1 = document.getElementById("myRect1");
    var myRect3 = document.getElementById("myRect3");

    if (evt.type == "beginEvent" && evt.target.id == "myRectNewDisplayAnim")
    {
      myRect1.setTrait("nav-right", "url(#myRectNew)");
      myRect3.setTrait("nav-left", "url(#myRectNew)");
    }
    else if (evt.type == "endEvent" && evt.target.id == "myRectNewDisplayAnim")
    {
      myRect1.setTrait("nav-right", "url(#myRect2)");
      myRect3.setTrait("nav-left", "url(#myRect2)");
    }
  ]]></handler>

</svg>
```

14 Linking

Contents

- 14.1 [References](#)
 - 14.1.1 [Overview](#)
 - 14.1.2 [IRIs and URIs](#)
 - 14.1.3 [Reference Restrictions](#)
 - 14.1.4 [IRI reference attributes](#)
 - 14.1.5 [Externally referenced documents](#)
- 14.2 [Links out of SVG content: the 'a' element](#)
- 14.3 [Linking into SVG content: IRI fragments and SVG views](#)
 - 14.3.1 [Introduction: IRI fragments and SVG views](#)
 - 14.3.2 [SVG fragment identifiers](#)
- 14.4 [Hyperlinking Module](#)
- 14.5 [XLink Attribute Module](#)

14.1 References

14.1.1 Overview

On the Internet, resources are identified using [IRIs](#) (International Resource Identifiers). For example, an SVG file called MyDrawing.svg located at <http://example.com> might have the following [IRI](#):

```
http://example.com/MyDrawing.svg
```

An [IRI](#) can also address a particular element within an XML document by including an [IRI](#) fragment identifier as part of the [IRI](#). An [IRI](#) which includes a [IRI](#) fragment identifier consists of an optional base [IRI](#), followed by a "#" character, followed by the [IRI](#) fragment identifier. For example, the following [IRI](#) can be used to specify the element whose ID is "Lamppost" within file MyDrawing.svg:

```
http://example.com/MyDrawing.svg#Lamppost
```

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

Example: 05_07.xml

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

You then reference the linear gradient as the value of the ['fill'](#) property for the rectangle, as in:

Example: 05_08.xml

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

14.1.2 IRIs and URIs

Internationalized Resource Identifiers ([IRIs](#)) are a more generalized complement to Uniform Resource Identifiers (URIs). An [IRI](#) is a sequence of characters from the Universal Character Set [Unicode40]. A URI is constructed from a much more restricted set of characters. All URIs are already conformant [IRIs](#). A mapping from [IRIs](#) to URIs is defined by the [IRI](#) specification, which means that IRIs can be used instead of URIs in XML documents, to identify resources. [IRIs](#) can be converted to URIs for resolution on a network, if the protocol does not support [IRIs](#) directly.

Previous versions of SVG, following XLink, defined a [\[IRI Reference\]](#) type as a URI or as a sequence of characters which must result in a URI reference after a particular escaping procedure was applied. The escaping procedure was repeated in the XLink specification [\[XLink\]](#), and in the W3C XML Schema Part 2: Datatypes specification [\[Schema2\]](#). This copying introduced the possibility of error and divergence, but was done because the [IRI](#) specification was not yet standardized.

In this specification, the correct term [IRI](#) is used for this URI or sequence of characters plus an algorithm' and the escaping method is defined by reference to the [IRI specification](#), which has since become an IETF Proposed Standard. Other W3C specifications are expected to be revised over time to remove these duplicate descriptions of the escaping procedure and to refer to [IRI](#) directly.

14.1.3 Reference Restrictions

Some of the elements using [IRI](#) references have restrictions on them. Which parts of [IRI](#) references that are allowed on each element is listed in the table below.

The following possibilities exist in SVG. A referencing element supports one or more of these possibilities.

The referencing attribute can:

- **A:** Reference a fragment within the current **SVG document fragment** (e.g. "#myelement").
- **B:** Reference a fragment of an external SVG document (e.g. "afile.svg#anelement").
- **C:** Reference an SVG document (e.g. "afile.svg").
- **D:** Reference a media resource other than SVG, with or without the use fragments (e.g. "myimage.jpg" or "mycontainer#fragment"). Where applicable, the table shows the supported Media Types.
- **E:** Use `data:` **IRIs** (e.g. "data:image/jpeg;base64,/9j..."). Note that `data:` (if XML) produces a different tree, but the data is already loaded as it is part of the **IRI** itself

An **IRI** reference not complying to the restrictions in the table below is an invalid **IRI** reference.

Elements	Referencing attribute	A	B	C	D	E
'set'	xlink:href	Yes, see Identifying the target element for an animation for reference rules.	No	No	No	No
'animate'	xlink:href	Yes, see Identifying the target element for an animation for reference rules.	No	No	No	No
'animateColor'	xlink:href	Yes, see Identifying the target element for an animation for reference rules.	No	No	No	No
'animateMotion'	xlink:href	Yes, see Identifying the target element for an animation for reference rules.	No	No	No	No
'animateTransform'	xlink:href	Yes, see Identifying the target element for an animation for reference rules.	No	No	No	No
'discard'	xlink:href	Yes, see Identifying the target element for an animation for reference rules.	No	No	No	No

<u>'a'</u>	xlink:href xlink:role xlink:arcrole	Yes, linking to animation elements .	Yes, referencing any element or a view on an SVG document (i.e. #svgView()).	Yes	Yes	Yes
<u>'use'</u>	xlink:href	Yes, but a <u>'use'</u> element must not reference an <u>'svg'</u> element.	Yes, but the referenced fragment must not contain scripting, hyperlinking to animations or any externally referenced <u>'use'</u> or <u>'animation'</u> elements.	No	No	No
<u>'image'</u>	xlink:href	No	No	No	Yes , but the <u>'image'</u> element must reference content that is a raster image format.	Yes, but the content within the 'data:' reference must be a raster image.
<u>'animation'</u>	xlink:href	No	No	Yes	No	Yes
<u>'prefetch'</u>	xlink:href	Yes	Yes	Yes	Yes	No
<u>'audio'</u>	xlink:href	No	No	No	Yes, depending on supported audio formats, indicated by the type attribute.	Yes
<u>'video'</u>	xlink:href	No	No	No	Yes, depending on supported video formats, indicated by the type attribute.	Yes
<u>'foreignObject'</u>	xlink:href	Yes	Yes	Yes	Yes	Yes
<u>'script'</u>	xlink:href	No	No	No	Yes , but it must reference an external resource that provides the script content.	Yes

'handler'	xlink:href	No	No	No	Yes , but it must reference an external resource that provides the script content.	Yes
'listener'	observer target handler	Yes	No	No	No	No
Any element using the 'fill' property	'fill'	Yes, only referencing a paint server, see Specifying paint .	No	No	No	No
Any element using the 'stroke' property	'stroke'	Yes, only referencing a paint server, see Specifying paint .	No	No	No	No
Any element using the focus attributes	'nav-next' 'nav-prev' 'nav-up' 'nav-up-right' 'nav-up-left' 'nav-left' 'nav-right' 'nav-down' 'nav-down-right' 'nav-down-left'	Yes, see Specifying Navigation .	No	No	No	No
'font-face-uri'	xlink:href	Yes, the reference must be to an SVG 'font' element.	Yes, the reference must be to an SVG 'font' element.	No	No	Yes
'mpath'	xlink:href	Yes, only referencing a 'path' element.	No	No	No	No

The following rules apply to the processing of invalid [IRI](#) references:

- A circular [IRI](#) reference shall represent an error (see [Error processing](#)).
- When attribute [externalResourcesRequired](#) has been set to **true** on the referencing element or one of its ancestors, then an unresolved external [IRI](#) reference (i.e., a resource that cannot be located) shall represent an error (see [Error processing](#)).

It is recommended that, wherever possible, referenced elements be defined inside of a **'defs'** element. Among the elements that are always referenced are **'linearGradient'** and **'radialGradient'**. Defining these elements inside of a **'defs'** element promotes understandability of the SVG content and thus promotes accessibility.

14.1.4 IRI reference attributes

An [IRI](#) reference must be specified within an [href](#) attribute in the XLink [[XLINK](#)] namespace. For example, if the prefix of 'xlink' is used for attributes in the XLink namespace, then the attribute shall be specified as [xlink:href](#). The value of this attribute must be an [IRI](#) reference for the desired resource (or secondary resource, if there is a fragment identifier).

The value of the [href](#) attribute must be an [IRI](#) reference as defined in the [IRI specification](#).

If the protocol, such as HTTP, does not support [IRIs](#) directly, the [IRI](#) is converted to a URI by the SVG implementation, as described in section 3.1 of the [IRI specification](#).

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

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

Additional XLink attributes can be specified that provide supplemental information regarding the referenced resource.

Schema: xlinkatt

```
<define name='svg.XLinkBase.attr' combine='interleave'>
  <optional>
    <attribute name='xlink:type' svg:animatable='true' svg:inheritable='false'>
      <value>simple</value>
    </attribute>
  </optional>
  <optional>
    <attribute name='xlink:role' svg:animatable='false' svg:inheritable='false'>
      <ref name='IRI.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='xlink:arcrole' svg:animatable='false' svg:inheritable='false'>
      <ref name='IRI.datatype' />
    </attribute>
  </optional>
  <optional>
    <attribute name='xlink:title' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>
</define>

<define name='svg.XLinkHrefRequired.attr' combine='interleave'>
  <attribute name='xlink:href' svg:animatable='true' svg:inheritable='false'>
    <ref name='IRI.datatype' />
  </attribute>
</define>

<define name='svg.XLinkBaseRequired.attr' combine='interleave'>
  <ref name='svg.XLinkBase.attr' />
  <ref name='svg.XLinkHrefRequired.attr' />
</define>

<define name='svg.XLinkActuateOnLoad.attr' combine='interleave'>
  <optional>
    <attribute name='xlink:actuate' svg:animatable='false' svg:inheritable='false'>
      <value>onLoad</value>
    </attribute>
  </optional>
</define>

<define name='svg.XLinkShowOther.attr' combine='interleave'>
  <optional>
    <attribute name='xlink:show' svg:animatable='false' svg:inheritable='false'>
      <value>other</value>
    </attribute>
  </optional>
</define>

<define name='svg.XLinkEmbed.attr' combine='interleave'>
  <optional>
    <attribute name='xlink:show' svg:animatable='false' svg:inheritable='false'>
      <value>embed</value>
    </attribute>
  </optional>
  <ref name='svg.XLinkActuateOnLoad.attr' />
  <ref name='svg.XLinkBaseRequired.attr' />
</define>

<define name='svg.XLinkRequired.attr' combine='interleave'>
  <ref name='svg.XLinkShowOther.attr' />
  <ref name='svg.XLinkActuateOnLoad.attr' />
  <ref name='svg.XLinkBaseRequired.attr' />
</define>

<define name='svg.XLinkReplace.attr' combine='interleave'>
  <optional>
    <attribute name='xlink:show' a:defaultValue='replace' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>new</value>
        <value>replace</value>
      </choice>
    </attribute>
  </optional>
</define>
```



```

</optional>
<optional>
  <attribute name='xlink:actuate' svg:animatable='false' svg:inheritable='false'>
    <value>onRequest</value>
  </attribute>
</optional>
<ref name='svg.XLinkBaseRequired.attr' />
</define>

<define name='svg.XLink.attr' combine='interleave'>
<optional>
  <ref name='svg.XLinkHrefRequired.attr' />
</optional>
<ref name='svg.XLinkShowOther.attr' />
<ref name='svg.XLinkActuateOnLoad.attr' />
<ref name='svg.XLinkBase.attr' />
</define>

```

xlink:type = 'simple'

Identifies the type of XLink being used. In SVG Tiny 1.2, only simple links are available. All links are simple links by default, so the attribute **xlink:type="simple"** is optional and need not be explicitly stated. Refer to the "XML Linking Language (XLink)" [\[XLink\]](#).

Animatable: no.

xlink:role = '<iri>'

An optional **IRI** reference that identifies some resource that describes the intended property. The value must be an **IRI** reference as defined in [\[RFC3987\]](#), except that if the **IRI** scheme used is allowed to have absolute and relative forms, the **IRI** portion must be absolute. When no value is supplied, no particular role value shall be inferred. Refer to the "XML Linking Language (XLink)" [\[XLink\]](#).

Animatable: no.

xlink:arcrole = '<iri>'

An optional **IRI** reference that identifies some resource that describes the intended property. The value must be an **IRI** reference as defined in [\[RFC3987\]](#), except that if the **IRI** scheme used is allowed to have absolute and relative forms, the **IRI** portion must be absolute. When no value is supplied, no particular role value shall be inferred. The arcrole attribute corresponds to the [\[RDF\]](#) notion of a property, where the role can be interpreted as stating that "starting-resource HAS arc-role ending-resource." This contextual role can differ from the meaning of an ending resource when taken outside the context of this particular arc. For example, a resource might generically represent a "person," but in the context of a particular arc it might have the role of "mother" and in the context of a different arc it might have the role of "daughter." Refer to the "XML Linking Language (XLink)" [\[XLink\]](#).

Animatable: no.

xlink:title = '<string>'

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

Animatable: no.

xlink:show = 'new' | 'replace' | 'embed' | 'other' | 'none'

This attribute is provided for backwards compatibility with SVG 1.1. It provides documentation to XLink-aware processors. In case of a conflict, the target attribute has priority, since it can express a wider range of values. Refer to the "XML Linking Language (XLink)" [\[XLink\]](#).

Animatable: no.

xlink:actuate = 'onLoad'

This attribute is provided for backwards compatibility with SVG 1.1. It provides documentation to XLink-aware processors. Refer to the "XML Linking Language (XLink)" [\[XLink\]](#).

Animatable: no.

In all cases, for compliance with the "Namespaces in XML 1.1" Recommendation [\[XML-NS\]](#), an explicit XLink namespace declaration must be provided whenever one of the above XLink attributes is used within SVG content. One simple way to provide such an XLink namespace declaration is to include an **xmlns** attribute for the XLink namespace on the **'svg'** element for content that uses XLink attributes.

Example: XLink namespace declaration

Example: 05_09.svg

```

<?xml version="1.0"?>
<svg xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Declaring the XLink namespace, as well as the SVG one</desc>
  <image xlink:href="foo.png"/>
</svg>

```

Example: use and animation

The two files below are the referenced files in the [use](#) and [animation](#) examples further down.

Example: referencedRect.svg

```

<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  version="1.2" baseProfile="tiny"
  id="animationRef" width="150" height="50" viewBox="0 0 150 50" fill="inherit">

  <rect id="aMovingRect" width="50" height="50" rx="5" ry="5" fill="inherit" stroke-width="3" stroke="black">
    <animateTransform attributeName="transform" type="translate" values="0,0;0,100" begin="0" dur="2" fill="freeze"/>

```

```
</rect>
</svg>
```

Example: [referencedRect2.svg](#)

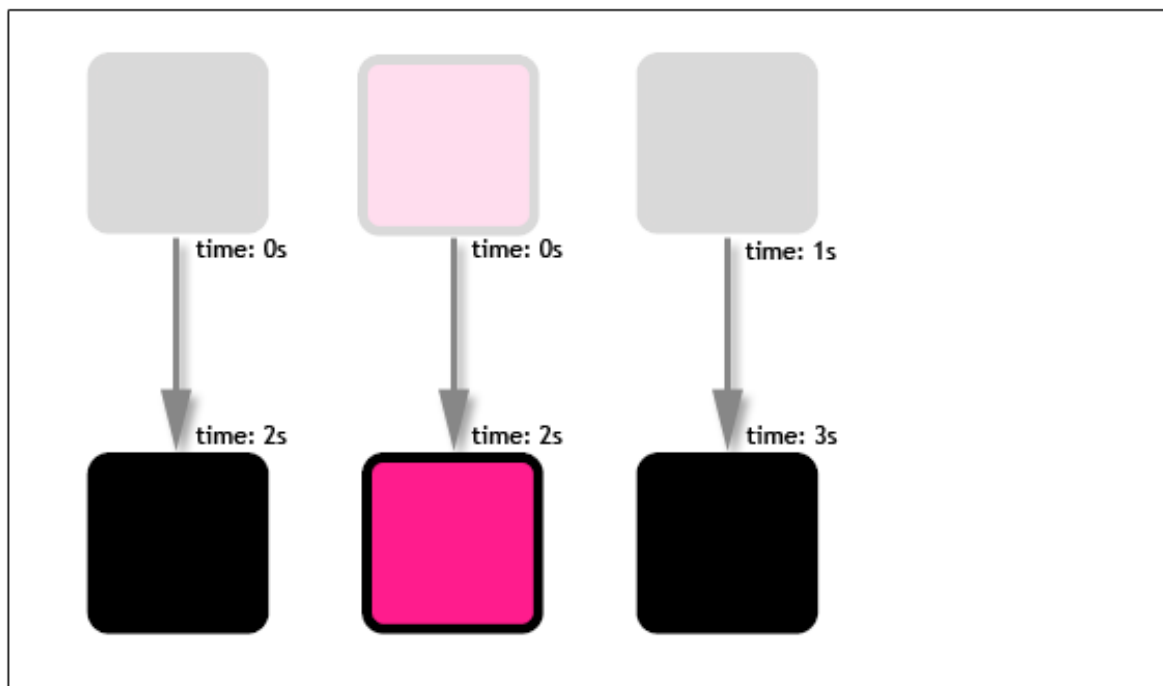
```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.2" baseProfile="tiny"
    id="animationRef" width="150" height="50" viewBox="0 0 150 50" fill="inherit">
    <rect id="aMovingRect" width="50" height="50" rx="5" ry="5" fill="rgb(255,28,141)" stroke-width="3" stroke="black">
        <animateTransform attributeName="transform" type="translate" values="0,0;0,100" begin="0" dur="2" fill="freeze"/>
    </rect>
</svg>
```

The following example illustrates how to reference svg content from the [animation](#) element. Different **'fill'** values are used to show the way properties are inherited on content referenced from the [animation](#) element.

Example: [animation.svg](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.2" baseProfile="tiny" width="100%" height="100%" viewBox="0 0 580 400">
    <g fill="rgb(157,0,79)">
        <animation x="20" xlink:href="referencedRect.svg"/>
        <animation x="100" xlink:href="referencedRect2.svg"/>
        <animation begin="1" x="180" viewport-fill="rgb(255,28,141)" xlink:href="referencedRect.svg"/>
    </g>
</svg>
```

The image below shows the correct rendering of the animation example above. The arrows indicates the animation. The grayed rectangles shows the initial state (i. e. time=0), the colored rectangles shows the final state (animations are completed).



The following example illustrates the different ways svg content can be referenced from a [use](#) element. Different **'fill'** values are used to show the way properties are inherited on content referenced from the [use](#) element.

Example: [use.svg](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.2" baseProfile="tiny" width="100%" height="100%" viewBox="0 0 580 400">
    <defs>
        <g fill="green">
            <rect id="aMovingRect" width="50" height="50" rx="5" ry="5" fill="inherit" stroke-width="3" stroke="black">
                <animateTransform attributeName="transform" type="translate" values="0,0;0,100" begin="0" dur="2" fill="freeze"/>
            </rect>
        </g>
    </defs>
    <g fill="rgb(157,0,79)">
```

```

<use x="20" xlink:href="#aMovingRect" />

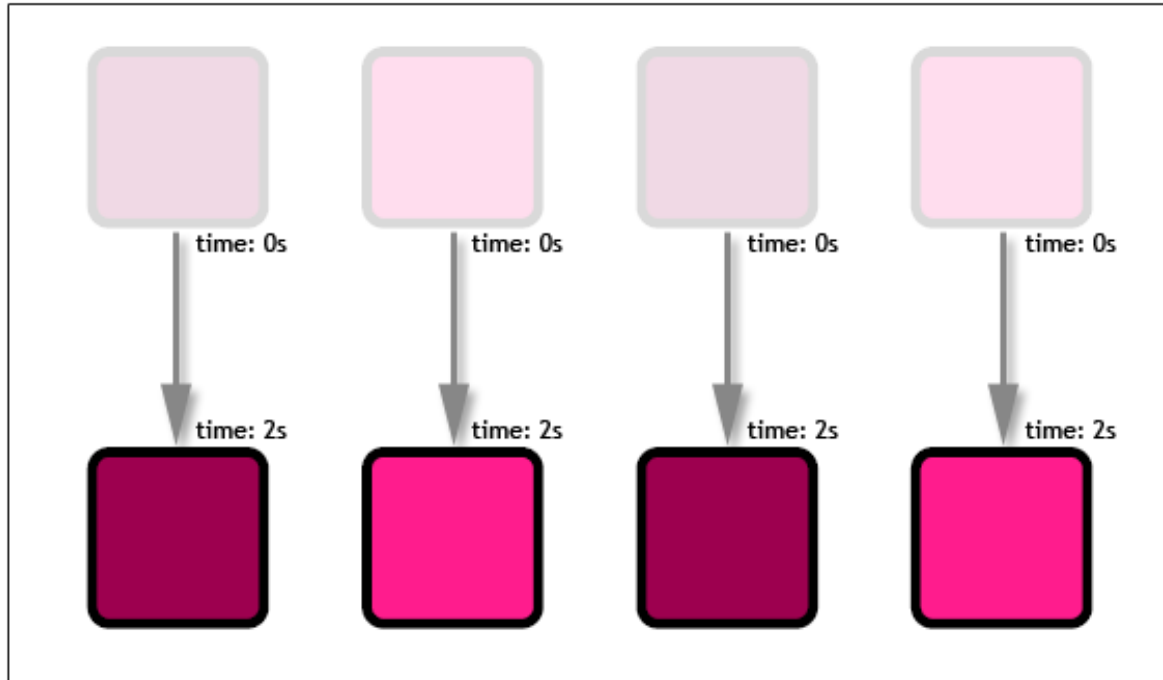
<use x="100" fill="rgb(255,28,141)" xlink:href="#aMovingRect" />

<use x="180" xlink:href="referencedRect.svg#aMovingRect" />

<use x="260" fill="rgb(255,28,141)" xlink:href="referencedRect.svg#aMovingRect" />
</g>
</svg>

```

The image below shows the correct rendering of the use example above. The arrows indicates the animation. The grayed rectangles shows the initial state (i.e. time=0), the colored rectangles shows the final state (animations are completed).



14.1.5 Externally referenced documents

In this section, the term "primary document" refers to a rootmost SVG document or SVG document fragment. If an SVG document is loaded directly by a web browser (e.g., the browser views file foo.svg), then it is the primary document. If an SVG document as a whole is referenced for inclusion by a parent document, such as using the HTML [object](#) or SVG [animation](#) elements, then that document itself shall also be a primary document. If an SVG document fragment is embedded inline within a non-SVG document, then the [svg](#) element shall define the root element for a subtree which acts as a primary document. In other words, it is the [rootmost svg element](#) for an [SVG document fragment](#).

Implementations are free to optimize for the case of large resource library loaded into multiple primary documents, but logically each primary document shall represent its own separate, self-contained document instance. In particular, the conceptual processing model is that events (e.g., the document load event) are fired; scripts are executed in normal fashion and resource documents are modifiable by scripts; coordinate systems transformations are applied; timelines start and animations are triggered; etc. For example, if document A.svg includes an [animation](#) element which refers to B.svg, then both are primary documents, and both shall represent separate, self-contained documents with their own scripting contexts and animation timelines.

The term "resource document" refers to a complete, self-contained SVG document which has at least one of its elements referenced as a resource by a primary document. For example, suppose document A.svg is loaded into a browser for viewing, and this document refers to a gradient element within B.svg via an [IRI](#) reference. In this case, A.svg is a primary document and B.svg is a resource document.

Each primary document must have an associated dictionary that maps all [IRIs](#) for resource documents it references; initially it is populated only with the primary document itself. Each resource or subresource loaded directly or indirectly must be resolved through that dictionary with resource documents downloaded as needed.

The conceptual model is that each resource document is loaded only once; if the same resource document is referenced multiple times directly or indirectly by the same primary document, that resource document is only retrieved and processed one time.

The conceptual processing model for resource documents is that the document is processed as a complete and separate SVG document instance. The only difference between a resource document and a primary document is that the primary document is rendered directly to the canvas, whereas all resource documents are conceptually rendered to an invisible (offscreen) canvas. In particular, the conceptual processing model is that events (e.g., the document load event) are fired; scripts are executed in normal fashion and resource documents are modifiable by scripts; coordinate systems transformations are applied; stylesheets are applied and the CSS cascade is run (not relevant to SVG1.2); timelines are initiated and animations execute; sXBL transformations are applied (not relevant to SVG1.2); etc.

Because of HTTP redirects, the same source document might be retrieved from multiple different source [IRIs](#). The rule for SVG is that documents are considered to be unique based on string comparisons of the full [IRI](#) after resolving relative IRIs into absolute [IRIs](#) and after taking into account HTTP redirects (i.e., use the post-redirect [IRI](#) instead of the original source [IRI](#)).

14.2 Links out of SVG content: the 'a' element

SVG provides an ['a'](#) element, analogous to HTML's ['a'](#) element, to indicate links (also known as *hyperlinks* or *Web links*). SVG uses XLink ([XLink](#)) for all link definitions.

SVG Tiny 1.2 only requires that user agents support XLink's notion of [simple links](#). Each simple link associates exactly two resources, one local and one remote, with an arc going from the former to the latter.

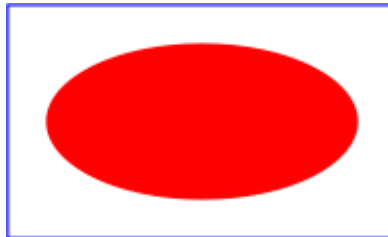
A simple link is defined for each separate rendered element contained within the ['a'](#) element; thus, if the ['a'](#) element contains three ['circle'](#) elements, a link is created for each circle. For each rendered element within an ['a'](#) element, the given rendered element is the local resource (the source anchor for the link).

The remote resource (the destination for the link) is defined by an [IRI](#) specified by the XLink [href](#) attribute on the ['a'](#) element. The remote resource may be any Web resource (e.g., an image, a video clip, a sound bite, a program, another SVG document, an HTML document, etc.). By activating these links (by clicking with the mouse, through keyboard input, voice commands, etc.), users may traverse hyperlinks to these resources.

Example [link01](#) assigns a link to an ellipse.

Example: [17_01.svg](#)

```
<?xml version="1.0"?>
<svg width="5cm" height="3cm" viewBox="0 0 5 3" version="1.2" baseProfile="tiny"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example link01 - a link on an ellipse
</desc>
<rect x=".01" y=".01" width="4.98" height="2.98"
  fill="none" stroke="blue" stroke-width=".03"/>
<a xlink:href="http://www.w3.org">
  <ellipse cx="2.5" cy="1.5" rx="2" ry="1"
    fill="red" />
</a>
</svg>
```



If the above SVG file is viewed by a user agent that supports both SVG and HTML, then clicking on the ellipse will cause the current window or frame to be replaced by the W3C home page.

Schema: a

```
<element name='a'>
  <ref name='a.AT' />
  <!-- the content model of 'a' is defined in the
       context of its containing element -->
</element>

<define name='a.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.Properties.attr' />
  <ref name='svg.FocusHighlight.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.XLinkReplace.attr' />
  <optional>
    <attribute name='target' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <value>_replace</value>
        <value>_self</value>
        <value>_parent</value>
        <value>_top</value>
        <value>_blank</value>
        <ref name='XML-Name.datatype' />
      </choice>
    </attribute>
  </optional>
</define>
```

Attribute definitions:

xlink:type = 'simple'

See generic description of [xlink:type](#) attribute.

`xlink:role = '<iri>'`

See generic description of [xlink:role](#) attribute.

`xlink:arcrole = '<iri>'`

See generic description of [xlink:arcrole](#) attribute.

`xlink:title = '<string>'`

See generic description of [xlink:title](#) attribute.

`xlink:show = 'new' | 'replace'`

This attribute is provided for backwards compatibility with SVG 1.1. It provides documentation to XLink-aware processors. If `target="_blank"` then use `xlink:show="new"` else use "replace". In case of a conflict, the target attribute has priority, since it can express a wider range of values. Refer to the "XML Linking Language (XLink)" [[XLink](#)].

Animatable: no.

`xlink:actuate = 'onRequest'`

This attribute is provided for backwards compatibility with SVG 1.1. It provides documentation to XLink-aware processors that an application should traverse from the starting resource to the ending resource only on a post-loading event triggered for the purpose of traversal. Refer to the "XML Linking Language (XLink)" [[XLink](#)].

Animatable: no.

`xlink:href = "<iri>"`

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

An empty attribute value (`xlink:href=""`) means that no link traversal occurs upon activation of the `a` element. If the attribute is not specified, the effect is as if an empty value (`""`) was specified.

Animatable: yes.

`target = '_replace' | '_self' | '_parent' | '_top' | '_blank' | "<frame-target>"`

This attribute should be used when there are multiple possible targets for the ending resource, such as when the parent document is a multi-frame HTML or XHTML document. This attribute specifies the name or portion of the target window, frame, pane, tab, or other relevant presentation context (e.g., an HTML or XHTML frame) into which a document is to be opened when the link is activated. The values and semantics of this attribute are the same as the WebCGM Picture Behavior values [[WebCGM](#)]:

_replace

The current SVG image is replaced by the linked content in the same rectangular area in the same frame as the current SVG image. This is the default value, if the target attribute is not specified.

_self

The current SVG image is replaced by the linked content in the same frame as the current SVG image.

_parent

The immediate frameset parent of the SVG image is replaced by the linked content.

_top

The content of the full window or tab, including any frames, is replaced by the linked content

_blank

A new un-named window or tab is requested for the display of the linked content. If this fails, the result is the same as `_top`

"<frame-target>"

Specifies the name of the frame, pane, or other relevant presentation context for display of the linked content. If this already exists, it is re-used, replacing the existing content. If it does not exist, it is created (the same as `_blank`, except that it now has a name). Note that `frame-target` must be an XML Name [XML11].

Note: The value `_new` is *not* a legal value for target (use `_blank`).

Animatable: yes.

`focusable = "true" | "false" | "auto"`

See [attribute definition](#) for description.

Animatable: Yes

Navigation Attributes

See [definition](#).

14.3 Linking into SVG content: IRI fragments and SVG views

14.3.1 Introduction: IRI fragments and SVG views

Because SVG content often represents a picture or drawing of something, a common need is to link into a particular **view** of the document, where a view indicates the initial transformations so as to present a closeup of a particular section of the document.

14.3.2 SVG fragment identifiers

To link into a particular view of an SVG document, the [IRI](#) fragment identifier must be a correctly formed SVG fragment identifier. An SVG fragment identifier defines the meaning of the "selector" or "fragment identifier" portion of [IRIs](#) that locate resources of MIME media type "image/svg+xml".

An SVG fragment identifier can come in two forms:

1. 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 "XPointer Framework" [[XPTRFW](#)].
2. **SVG view specification** (e.g., `MyDrawing.svg#svgView(transform(scale(2)))`). 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 is `transform(...)` whose parameters have the same meaning as the corresponding attribute has on a `g` element).

An SVG fragment identifier is defined as follows:

```

SVGFragmentIdentifier ::= BareName |
                        SVGViewSpec

BareName ::= XML_Name
SVGViewSpec ::= 'svgView(' SVGViewAttributes ')'
SVGViewAttributes ::= SVGViewAttribute |
                    SVGViewAttribute ';' SVGViewAttributes

SVGViewAttribute ::= transformSpec
transformSpec ::= 'transform(' TransformParams ')'

```

where:

- **XML_Name** conforms to the XML production for '[XML Name](#)'
- **TransformParams** corresponds to the parameter values for the [transform](#) attribute that is available on many elements. For example, `transform(scale(5))`.

An SVG fragment identifier must match the specified grammar. Spaces shall not be allowed in fragment specifications and commas must be used to separate numeric values within an SVG view specification.

Note: since fragment identifiers are stripped from IRIs before resolution, there is no need to escape any characters in fragments that are outside the repertoire of US-ASCII.

When a source document performs a link into an SVG document via an HTML [HTML4](#) anchor element (i.e., `` element in HTML) or an XLink specification [XLINK](#), then the SVG fragment identifier shall specify the initial view into the SVG document, as follows:

- If no SVG fragment identifier is provided (e.g. the specified [IRI](#) did not contain a "#" character, such as `MyDrawing.svg`), then the initial view into the SVG document shall be established using the view specification attributes (i.e., `viewBox`, etc.) on the `'svg'` element.
- If the SVG fragment identifier addresses any element, such as `MyDrawing.svg#rectId`, then the document defined by the [rootmost svg element](#) shall be displayed in the viewport using the view specification attributes on the [rootmost svg element](#).

14.4 Hyperlinking Module

The Hyperlinking Module contains the following element:

- [a](#)

[RNG] [Feature String](#)

14.5 XLink Attribute Module

The XLink Attribute Module contains the following attributes:

- [xlink:type](#)
- [xlink:href](#)
- [xlink:role](#)
- [xlink:arcrole](#)
- [xlink:title](#)
- [xlink:show](#)
- [xlink:actuate](#)

[RNG] [Feature String](#)

15 Scripting

Contents

- 15.1 [Specifying the scripting language](#)
 - 15.1.1 [Specifying the default scripting language](#)
 - 15.1.2 [Local declaration of a scripting language](#)
- 15.2 [The 'script' element](#)
- 15.3 [XML Events](#)
- 15.4 [The listener element](#)
- 15.5 [The handler element](#)
 - 15.5.1 [Parameters to handler elements](#)
- 15.6 [Event handling](#)
- 15.7 [Scripting Module](#)
- 15.8 [Handler Module](#)
- 15.9 [Listener Module](#)

15.1 Specifying the scripting language

15.1.1 Specifying the default scripting language

The `'contentScriptType'` attribute on the `'svg'` element specifies the default scripting language for the given document fragment.

15.1.2 Local declaration of a scripting language

It is also possible to specify the scripting language for each individual `script` or `handler` elements by specifying a `type` attribute on the `script` and `handler` elements.

15.2 The 'script' element

A `script` element may either contain or point to executable content (e.g., ECMAScript [\[ECMAScript\]](#) or Java [\[JAVA\]](#) JAR file). Executable content can come either in the form of a script (textual code) or in the form of compiled code. If the code is textual, it can either be placed inline in the `script` element (as character data) or as an external resource, referenced through `xlink:href` attribute. Compiled code must be an external resource.

Scripting languages (such as ECMAScript) that have a notion of a "global scope" or a "global object" such that a single global object must be associated with the document (unique for each uDOM [Document](#) node). This object is shared by all elements contained in that document. Thus, an ECMAScript function defined within any `script` element must be in the "global" scope of the entire document to which the script belongs. The global object must have all the methods and attributes from the [SVGGlobal](#) interface. It must be made available from an [SVGDocument](#) through the [global](#) attribute. Event listeners attached through event attributes and `handler` elements are also evaluated using the global scope of the document in which they are defined.

For compiled languages (such as Java) that don't have a notion of "global scope", each `script` element, in effect, provides a separate scope object. This scope object must perform an initialization as described in the [uDOM chapter](#) and serves as event listener factory for the `handler` element.

Any scripting logic associated with a given `script` element is executed at most once. Script execution happens just after the `load` event occurs for the given `script` element. Removing, inserting or altering `script` elements after script execution has taken place has no effect.

Exact details on how this element works depend on the executable content's type. SVG Tiny 1.2 does not require support for any particular programming language. However, SVG defines the behavior for two specific script types in the case where an implementation supports it:

application/ecmascript

This type of executable content must be source code for the ECMAScript programming language. This code must be executed in the context of this element's owner document's global scope as explained above.

SVG implementations that load external resources through protocols such as HTTP that support content coding must accept external script files that have been encoded using gzip compression (flagged using "Content-Encoding: gzip" for HTTP).

application/java-archive

This type of executable content must be an external resource that contains a Java JAR archive. The manifest file in the JAR archive must have an entry named SVG-Handler-Class. The entry's value must be a fully-qualified Java class name for a class contained in this archive. The user agent must instantiate the class from the JAR file and cast it to the [EventListenerInitializer2](#) interface. Then the [initializeEventListeners](#) method must be called with the `script` element object itself as a parameter. If a class listed in SVG-Handler-Class does not implement [EventListenerInitializer2](#), it is an error.

Note that the user agent may reuse classes loaded from the same URL, so the code must not assume that every `script` element or every document will create its own separate class object. Thus, one cannot assume, for instance, that static fields in the class are private to a document.

Implementations should also accept the script type **"text/ecmascript"** for backwards compatibility with SVG 1.1. However, this type is deprecated and should not be used by content authors.

Other language bindings are encouraged to adopt a similar approach to either of the two described above.

[Example 18_01](#) defines a function `circle_click` which is called when the **'circle'** element is being clicked. The drawing below on the left is the initial image. The drawing below on the right shows the result after clicking on the circle. The example uses the [handler](#) element which is described further down in this chapter.

Note that this example demonstrates the use of the [click](#) event for explanatory purposes. The example presupposes the presence of an input device with the same behavioral characteristics as a mouse, which will not always be the case. To support the widest range of users, the [activate](#) event attribute should be used instead of the [click](#) event attribute.

Example: 18_01.svg

```
<?xml version="1.0"?>
<svg width="6cm" height="5cm" viewBox="0 0 600 500"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:ev="http://www.w3.org/2001/xml-events">
  <desc>Example: invoke an ECMAScript function from an click event
  </desc>
  <!-- ECMAScript to change the radius with each click -->
  <script type="application/ecmascript"> <![CDATA[
    function circle_click(evt) {
      var circle = evt.target;
      var currentRadius = circle.getFloatTrait("r");
      if (currentRadius == 100)
        circle.setFloatTrait("r", currentRadius*2);
      else
        circle.setFloatTrait("r", currentRadius*0.5);
    }
  ]]> </script>

  <!-- Outline the drawing area with a blue line -->
  <rect x="1" y="1" width="598" height="498" fill="none" stroke="blue"/>
  <!-- Act on each click event -->
  <circle cx="300" cy="225" r="100" fill="red">
  <handler type="application/ecmascript" ev:event="ev:click">
    circle_click(evt);
  </handler>
  </circle>

  <text x="300" y="480" font-family="Verdana" font-size="35" text-anchor="middle">
    Click on circle to change its size
  </text>
</svg>
```



Click on circle to change its size



Click on circle to change its size

Schema: script

```
<define name='script'>
  <element name='script'>
    <ref name='script.AT' />
    <ref name='script.CM' />
  </element>
</define>
```



```

<define name='script.AT' combine='interleave'>
  <ref name='svg.CorePreserve.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.ContentType.attr' />
</define>

<define name='script.CM'>
  <choice>
    <group>
      <ref name='svg.XLinkRequired.attr' />
    </group>
    <text />
  </choice>
</define>

```

Attribute definitions:

type = "content-type"

Identifies the programming language for the given **script** element. The value **content-type** specifies a media type, per per [Multipurpose Internet Mail Extensions \(MIME\) PartTwo \[RFC2046\]](#). If a 'type' is not provided, the value of '**contentScriptType**' on the **svg** element shall be used, which in turn defaults to "application/ecmascript".

Animatable: no.

xlink:href = "<iri>"

An [IRI reference](#) to an external resource containing the script code.

Animatable: no.

15.3 XML Events

XML Events is an XML syntax for integrating event listeners and handlers with DOM 2 and DOM 3 Event interfaces. Declarative event handling in SVG 1.1 was hardwired into the language, in that the developer was required to embed the event handler in the element syntax (e.g. an element with an onclick attribute). SVG Tiny 1.2 does not support the event attributes (onload, onclick, onactivate, etc.). Instead SVG Tiny 1.2 uses XML Events to provide the ability to specify the event listener separately from the graphical content.

SVG Tiny 1.2 uses XML Events and makes the following modifications:

1. **Namespaced events:** As SVG Tiny 1.2 supports DOM Level 3 Events, which are namespaced, XML Events in SVG Tiny 1.2 must allow namespaced events.
2. **IRIREFs instead of IDREFs:** the **observer**, **handler** and **target** attributes from XML Events are currently IDREFs. Since SVG 1.2 requires a declarative syntax for event handling in more than one document, it uses IRIREFs for those attributes, with the following restriction: only documents that are declaratively referenced as part of the current document, via the **use** and **animation** elements, can be referred to. Referring to any other arbitrary external document is unsupported and User Agents must ignore such references.
3. **No capture phase:** SVG Tiny 1.2 does not have an event capture phase, only a bubble and a target phase.
4. **XML Events attributes shall only apply to the listener and handler elements.** The specific attributes allowed on each element are defined in the respective sections below.

The [list of events](#) supported by SVG Tiny 1.2 is given in the Interactivity chapter.

There are two ways to place a handler in SVG Tiny 1.2 content. The first method is most suitable for simple cases:

Example: simplehandler.svg

```

<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:ev="http://www.w3.org/2001/xml-events">

  <rect x="10" y="20" width="10" height="20" fill="red">
    <handler type="application/ecmascript" ev:event="ev:click">
      var myRect = evt.target;
      var width = myRect.getFloatTrait("width");
      myRect.setFloatTrait("width", (width+10));
    </handler>
  </rect>

</svg>

```

In this method the **handler** element is a child element of the **observer**. For instance one can place a **handler** as a child of a **rect** element, which becomes the **observer**. This causes the **handler** element to be invoked whenever the event that it is interested in (e.g.: "click") occurs on the **rect**.

The following is an example of an SVG document using XML Events where the **handler** element can be reused on several objects. The **listener** element from XML Events is used to specify the **observer** and **handler** for a particular **event**.

Example: [handler.svg](#)

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:ev="http://www.w3.org/2001/xml-events">

  <desc>An example of the handler element.</desc>

  <rect xml:id="myRect1" x="10" y="20" width="10" height="20" fill="red"/>
  <rect xml:id="myRect2" x="10" y="40" width="10" height="20" fill="green"/>

  <ev:listener event="ev:click" observer="#myRect1"
    handler="#myClickHandler"/>
  <ev:listener event="ev:click" observer="#myRect2"
    handler="#myClickHandler"/>

  <handler xml:id="myClickHandler" type="application/ecmascript">
    var myRect = evt.target;
    var width = myRect.getFloatTrait("width");
    myRect.setFloatTrait("width", (width+10));
  </handler>

</svg>
```

In the above example, the **ev:listener** element registers that the myClickHandler element should be invoked whenever a [click](#) event happens on "myRect1" or "myRect2".

The combination of the XML Events syntax and the new **handler** element allows event handling to be more easily processed in a compiled language. Below is an example of an event handler using the Java language:

Example: [javahandler.svg](#)

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:foo="http://www.example.com/foo"
  xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>Example of a Java handler</desc>

  <rect xml:id="myRect" x="10" y="20" width="200" height="300"
    fill="red"/>

  <!-- reference a jar containing an EventListenerInitializer2 object -->
  <script type="application/java-archive" xml:id="init" xlink:href="http://example.com/myJar.jar"/>

  <!-- register a listener for a myRect.click event -->
  <ev:listener event="ev:click" observer="#myRect"
    handler="#myClickHandler" />

  <handler xml:id="myClickHandler" type="application/java-archive" xlink:href="#init" foo:offset="10"/>

</svg>
```

In this case, the **handler** element specifies a reference to the **script** element that specifies the location of compiled code that conforms to the [EventListenerInitializer2](#) interface. The user agent invokes the [createEventListener](#) method within the targeted interface.

In this case, the MyEventListenerInitializer2 referenced by the SVG-Handler-Class entry of the myJar.jar manifest has the following definition:

MyEventListenerInitializer2

```
package com.example;

import org.w3c.svg.EventListenerInitializer2;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.events.Event;
import org.w3c.dom.events.EventListener;

public class MyEventListenerInitializer2 implements EventListenerInitializer2 {

    Document document;

    public void initializeEventListeners(Element scriptElement) {
        document = scriptElement.getOwnerDocument();
    }

    public EventListener createEventListener(final Element handlerElement) {
        return new EventListener() {
            public void handleEvent(Event event) {
                Element myRect = document.getElementById("myRect");
                float width = Float.parseFloat(myRect.getAttributeNS(null, "width"));
                float offset = Float.parseFloat(handlerElement.getAttributeNS("http://www.example.com/foo", "offset"));
                myRect.setAttributeNS(null, width, "" + (width + 10));
            }
        };
    }
};
```

```
}  
}
```

The [EventListenerInitializer2](#) interface is currently defined in the SVG package. Future specifications may move this package though it is guaranteed to always be available in the SVG package.

15.4 The listener element

The **'listener'** element from XML Events [\[XML-EVENTS\]](#) must be supported. The definition for the **'listener'** element is provided in [\[XML-EVENTS\]](#). Any additional restrictions from this specification must also apply.

Whenever a **'listener'** element is removed (via uDOM, discard, ...), the behavior is the same as if the [removeEventListener](#) method was called on the observer element on which the event listener was registered. Note that if the **'observer'** attribute is not present, the observer is the parent of the **'listener'** element.

Please note that the **'listener'** element must be specified in the XML Events namespace, and that an element in the SVG namespace with 'listener' as its local name must not be understood as being the element described in this chapter. Furthermore, the XML Events attributes that are available on other elements only when they are in the XML Events namespace, are only available on this element when they are in no namespace.

Schema: listener

```
<define name='listener'>  
  <element name='listener'>  
    <ref name='listener.AT' />  
    <ref name='listener.CM' />  
  </element>  
</define>  
  
<define name='listener.AT' combine='interleave'>  
  <ref name='svg.Core.attr' />  
  <optional>  
    <attribute name='event' svg:animatable='false' svg:inheritable='false'>  
      <ref name='XSLT-QName.datatype' />  
    </attribute>  
  </optional>  
  <optional>  
    <attribute name='phase' svg:animatable='false' svg:inheritable='false'>  
      <choice>  
        <value>default</value>  
        <value>capture</value>  
      </choice>  
    </attribute>  
  </optional>  
  <optional>  
    <attribute name='propagate' svg:animatable='false' svg:inheritable='false'>  
      <choice>  
        <value>continue</value>  
        <value>stop</value>  
      </choice>  
    </attribute>  
  </optional>  
  <optional>  
    <attribute name='defaultAction' a:defaultValue='perform' svg:animatable='false' svg:inheritable='false'>  
      <choice>  
        <value>perform</value>  
        <value>cancel</value>  
      </choice>  
    </attribute>  
  </optional>  
  <optional>  
    <attribute name='observer' svg:animatable='false' svg:inheritable='false'>  
      <ref name='IRI.datatype' />  
    </attribute>  
  </optional>  
  <optional>  
    <attribute name='target' svg:animatable='false' svg:inheritable='false'>  
      <ref name='IRI.datatype' />  
    </attribute>  
  </optional>  
  <optional>  
    <attribute name='handler' svg:animatable='false' svg:inheritable='false'>  
      <ref name='IRI.datatype' />  
    </attribute>  
  </optional>  
</define>  
  
<define name='listener.CM'>  
  <empty />  
</define>
```

Attribute definitions:

event = **'Event Identifier'**

The **'event'** attribute must be a valid SVG Tiny 1.2 'Event Identifier' as defined in the [list of supported events](#).

Animatable: no.

observer = "**<iri>**

The **'observer'** attribute is defined in XML Events. In SVG Tiny 1.2 this attribute must be an [IRI reference](#) rather than an IDREF as defined in XML Events. Restrictions specified in this chapter as to which IRIs are acceptable must be a enforced.

Animatable: no.

target = "**<iri>**

The **'target'** attribute is defined in XML Events. In SVG Tiny 1.2 this attribute must be an [IRI reference](#) rather than an IDREF as defined in XML Events. Restrictions specified in this chapter as to which IRIs are acceptable must be a enforced.

Animatable: no.

handler = "**<iri>**

The **'handler'** attribute is defined in XML Events. In SVG Tiny 1.2 this attribute must be an [IRI reference](#) rather than an IDREF as defined in XML Events. Restrictions specified in this chapter as to which IRIs are acceptable must be a enforced.

Animatable: no.

phase = **'default'**

The **'phase'** attribute is defined in XML Events. In SVG Tiny 1.2 this attribute must only accept the value 'default'. The capture phase is not supported in SVG Tiny 1.2.

Animatable: no.

propagate = **'stop'|'continue'**

The **'propagate'** attribute is defined in XML Events.

Animatable: no.

defaultAction = **'cancel'|'perform'**

The **'defaultAction'** attribute is defined in XML Events.

Animatable: no.

15.5 The handler element

The **handler** element is similar to the **script** element: its contents, either included inline or referenced, are code that is to be executed by the scripting engine(s) used by user agent.

However, where the **script** element executes its contents when the document is loaded, the **handler** element must only execute its contents in response to an event. This means that SVG Tiny 1.2 uses **handler** to get the functionality equivalent to that provided by SVG Full [event attributes](#). Removing, inserting or altering **handler** elements after their content has been loaded has no effect on executable content, but changes made to the parameters that are available from a given **handler** element must be reflected when the **handler** content is executed.

For example, consider the following SVG 1.1 document:

Example: SVG 1.1 scripting

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect id="myRect" x="10" y="20" width="200" height="300" fill="red"
    onclick="evt.target.width.baseVal.value += 10"/>
</svg>
```

The above example must be rewritten to use the **handler** element and XML Events (described below) as shown:

Example: handler2.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
  xmlns:ev="http://www.w3.org/2001/xml-events">
  <desc>handler element example</desc>
  <rect xml:id="myRect" x="10" y="20" width="200" height="300" fill="red">

    <handler type="application/ecmascript" ev:event="ev:click">
      var myRect = evt.target;
      var width = myRect.getFloatTrait("width");
      myRect.setFloatTrait("width", (width+10));
    </handler>

  </rect>
</svg>
```

In ECMAScript, the contents of the **handler** element behave as if they are the contents of a new Function object, created as shown:

```
Function(evt) {
```

```

    //contents of handler
}

```

Other interpreted languages should behave in a similar manner.

The 'evt' parameter shown above is an [Event object](#) corresponding to the event that has triggered the [handler](#).

Schema: handler

```

<define name='handler'>
  <element name='handler'>
    <ref name='handler.AT' />
    <ref name='handler.CM' />
  </element>
</define>

<define name='handler.AT' combine='interleave'>
  <ref name='svg.CorePreserve.attr' />
  <ref name='svg.External.attr' />
  <attribute name='ev:event' svg:animatable='false' svg:inheritable='false'>
    <ref name='XSLT-QName.datatype' />
  </attribute>
  <ref name='svg.ContentType.attr' />
</define>

<define name='handler.CM'>
  <choice>
    <group>
      <ref name='svg.XLinkRequired.attr' />
    </group>
    <text />
  </choice>
</define>

```

Attribute definitions:

type = "content-type"

Identifies the language used for the handler element. The value specifies a media type, per [RFC2045](#). If a 'type' is not provided, the value of '[contentScriptType](#)' on the '[svg](#)' element shall be used.

Animatable: no.

xlink:href = "<iri>"

If this attribute is present, then the script content of the [handler](#) element must be loaded from this resource and what content the [handler](#) element may have must not be executable.

Animatable: no.

ev:event = "<string>"

The name of the event to handle. This attribute is in the [XML Events](#) namespace. See [event list](#) for a list of all supported events and [XML-Events](#) for the definition of the [ev:event](#) attribute.

Animatable: no.

For compiled languages, the xlink:href attribute must reference a [script](#) element that itself references a JAR archive holding a manifest with an SVG-Handler-Class entry pointing to an [EventListenerInitializer2](#) implementation.

15.5.1 Parameters to handler elements

In many situations, the script author uses the [handler](#) as a template for calling other functions, using the content of the [handler](#) element to pass parameters. However, for compiled languages the [handler](#) element does not have any executable content.

In this case, the author should embed the parameters into the [handler](#) as custom content under the form of element children in a foreign namespace, or attributes on the [handler](#) element also in foreign namespaces.

Below is an example of using parameters on the [handler](#) element:

Example: [handlerparam.svg](#)

```

<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:foo="http://www.example.com/foo"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>An example of parameters on the handler element.</desc>

  <rect xml:id="myRect" x="10" y="20" width="200" height="300"
    fill="red"/>

```

```

<!-- reference a jar containing an EventListenerInitializer2 object -->
<script type="application/java-archive" xml:id="init" xlink:href="http://example.com/myJar.jar"/>

<!-- register a listener for a myRect.click event -->
<ev:listener event="ev:click" observer="#myRect"
  handler="#myClickHandler" />

<handler xml:id="myClickHandler" type="application/java-archive"
  xlink:href="#init">
  <foo:offset value="10"/>
  <foo:person>
    <foo:name>Victor Vector</foo:name>
    <foo:age>42</foo:age>
  </foo:person>
</handler>

</svg>

```

In this case, the object referenced by the SVG-Handler-Class entry of the myJar.jar manifest has its [createEventListener](#) method called and the returning [EventListener](#) registered. Whenever a [click](#) event on the 'myRect' object is observed, the [handleEvent](#) method of the listener is called. The object can then access of the [handler](#) element in order to obtain its parameters from elements in the "foo" namespace.

15.6 Event handling

Events must cause scripts to execute when either of the following has occurred:

- Events are assigned to particular elements and connected with script through the [handler](#) element. The script is executed when the given event occurs.
- [Event listeners](#) as described in "Document Object Model Events" [[DOM3-EVENTS](#)] are defined which are invoked when a given event happens on a given object

Related sections of the spec:

- [User interface events](#) describes how an SVG user agent handles events such as pointer movements events (e.g., mouse movement) and activation events (e.g., mouse click).

15.7 Scripting Module

The Scripting Module contains the following element:

- [script](#)

[[RNG](#)] [[Feature String](#)]

15.8 Handler Module

The Handler Module contains the following element:

- [handler](#)

[[RNG](#)] [[Feature String](#)]

15.9 Listener Module

The Listener Module contains the following element:

- [listener](#)

[[RNG](#)] [[Feature String](#)]

16 Animation

Contents

- 16.1 [Introduction](#)
- 16.2 [Animation elements](#)
 - 16.2.1 [Overview](#)
 - 16.2.2 [Relationship to SMIL 2.0 Animation](#)
 - 16.2.3 [Animation elements example](#)
 - 16.2.4 [Attributes to identify the target element for an animation](#)
 - 16.2.5 [Attributes to identify the target attribute or property for an animation](#)
 - 16.2.6 [Paced animation and complex types](#)
 - 16.2.7 [Attributes to control the timing of the animation](#)
 - 16.2.8 [Attributes that define animation values over time](#)
 - 16.2.9 [Attributes that control whether animations are additive](#)
 - 16.2.10 [Inheritance](#)
 - 16.2.11 [The 'animate' element](#)
 - 16.2.12 [The 'set' element](#)
 - 16.2.13 [The 'animateMotion' element](#)
 - 16.2.14 [The 'mpath' element](#)
 - 16.2.15 [The 'animateColor' element](#)
 - 16.2.16 [The 'animateTransform' element](#)
 - 16.2.17 [Attributes and properties that can be animated](#)
- 16.3 [Animation using the SVG DOM](#)
- 16.4 [Timed Animation Module](#)

16.1 Introduction

SVG supports the ability to change vector graphics over time. SVG content can be animated in the following ways:

- Using SVG's [animation elements](#). SVG document fragments can describe time-based modifications to the document's elements. Using the various animation elements, you can define motion paths, fade-in or fade-out effects, and objects that grow, shrink, spin or change color.
- Using the [SVG uDOM](#). The SVG uDOM conforms to the [Document Object Model \(DOM\) Level 3](#) specification [[DOM3](#)]. Some of the SVG attributes are accessible to scripting and by manipulating them many kind of animations can be achieved. The timer facility in the uDOM can be used to start up and control the animations. (See [example](#) below.)

16.2 Animation elements

16.2.1 Overview

SVG's animation elements were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the [Synchronized Multimedia Integration Language \(SMIL\) 2.0](#) Specification [[SMIL 2.0](#)].

SVG incorporates the animation features defined in the SMIL 2.0 specification and provides some SVG-specific extensions.

For an introduction to the approach and features available in any language that supports SMIL 2.0 Animation, see [SMIL 2.0 Animation overview](#) and [SMIL 2.0 animation model](#). For the list of animation features which go beyond SMIL Animation, see [SVG extensions to SMIL 2.0 Animation](#).

16.2.2 Relationship to SMIL 2.0 Animation

SVG is a host language in terms of SMIL 2.0 Animation and therefore introduces additional constraints and features as permitted by that specification. Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for SVG's animation elements and attributes are the [SMIL 2.0 Animation Modules](#).

SVG supports the following four animation elements which are defined in the SMIL 2.0 Animation Modules:

- 'animate'** allows scalar attributes and properties to be assigned different values over time
- 'set'** a convenient shorthand for **'animate'**, which is useful for assigning animation values to non-numeric attributes and properties, such as the **'visibility'** property
- 'animateMotion'** moves an element along a motion path

'animateColor' modifies the color value of particular attributes or properties over time

Additionally, SVG includes the following compatible extensions to SMIL 2.0:

'animateTransform' modifies one of SVG's transformation attributes over time, such as the [transform](#) attribute

[path](#) attribute SVG allows any feature from SVG's [path data](#) syntax to be specified in a [path](#) attribute to the **'animateMotion'** element (SMIL 2.0 Animation only allows a subset of SVG's path data syntax within a [path](#) attribute)

'mpath' element SVG allows an **'animateMotion'** element to contain a child **'mpath'** element which references an SVG **'path'** element as the definition of the motion path

[keyPoints](#) attribute SVG adds a [keyPoints](#) attribute to the **'animateMotion'** to provide precise control of the velocity of motion path animations

[rotate](#) attribute SVG adds a [rotate](#) attribute to the **'animateMotion'** to control whether an object is automatically rotated so that its x-axis points in the same direction (or opposite direction) as the directional tangent vector of the motion path

[discard](#) element SVG adds a [discard](#) element which removes the target element from the DOM tree at a specified time

For compatibility with other aspects of the language, SVG uses [IRI references](#) via an [xlink:href](#) attribute to identify the elements which are to be targets of the animations.

SMIL 2.0 Animation requires that the host language define the meaning for **document begin** and **document end**. The **document begin** for a given SVG document fragment depends on the value of the **'timelineBegin'** attribute. The **document begin** for:

- **timelineBegin=onLoad** The exact time at which the **'svg'** element's [load event](#) is triggered
- **timelineBegin=onStart** The exact time at which the **'svg'** element's open tag is fully parsed and processed

The **document end** of an **SVG document fragment** is the point at which the document fragment has been released and is no longer being processed by the user agent.

For SVG, the term **presentation time** indicates the position in the timeline relative to the **document begin** of a given document fragment.

16.2.3 Animation elements example

Example [anim01](#) below demonstrates each of SVG's five animation elements.

Example: [19_01.svg](#)

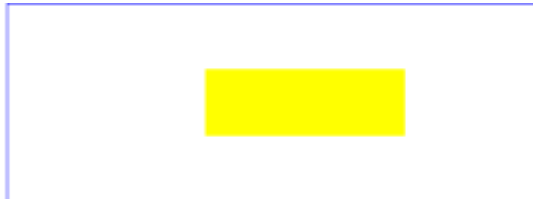
```
<?xml version="1.0"?>
<svg width="8cm" height="3cm" viewBox="0 0 800 300"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example anim01 - demonstrate animation elements</desc>
  <rect x="1" y="1" width="798" height="298"
    fill="none" stroke="blue" stroke-width="2" />
  <!-- The following illustrates the use of the 'animate' element
    to animate a rectangles x, y, and width attributes so that
    the rectangle grows to ultimately fill the viewport. -->
  <rect xml:id="RectElement" x="300" y="100" width="300" height="100"
    fill="rgb(255,255,0)" >
    <animate attributeName="x"
      begin="0s" dur="9s" fill="freeze" from="300" to="0" />
    <animate attributeName="y"
      begin="0s" dur="9s" fill="freeze" from="100" to="0" />
    <animate attributeName="width"
      begin="0s" dur="9s" fill="freeze" from="300" to="800" />
    <animate attributeName="height"
      begin="0s" dur="9s" fill="freeze" from="100" to="300" />
  </rect>
  <!-- Set up a new user coordinate system so that
    the text string's origin is at (0,0), allowing
    rotation and scale relative to the new origin -->
  <g transform="translate(100,100)" >
    <!-- The following illustrates the use of the 'set', 'animateMotion',
    'animateColor' and 'animateTransform' elements. The 'text' element
    below starts off hidden (i.e., invisible). At 3 seconds, it:
    * becomes visible
    * continuously moves diagonally across the viewport
    * changes color from blue to dark red
```



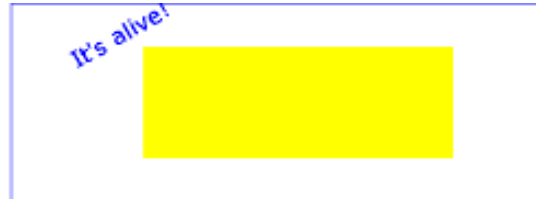
```

    * rotates from -30 to zero degrees
    * scales by a factor of three. -->
<text xml:id="TextElement" x="0" y="0"
      font-family="Verdana" font-size="35.27" visibility="hidden" >
  It's alive!
  <set attributeName="visibility" to="visible"
    begin="3s" dur="6s" fill="freeze" />
  <animateMotion path="M 0 0 L 100 100"
    begin="3s" dur="6s" fill="freeze" />
  <animateColor attributeName="fill"
    from="rgb(0,0,255)" to="rgb(128,0,0)"
    begin="3s" dur="6s" fill="freeze" />
  <animateTransform attributeName="transform"
    type="rotate" from="-30" to="0"
    begin="3s" dur="6s" fill="freeze" />
  <animateTransform attributeName="transform"
    type="scale" from="1" to="3" additive="sum"
    begin="3s" dur="6s" fill="freeze" />
</text>
</g>
</svg>

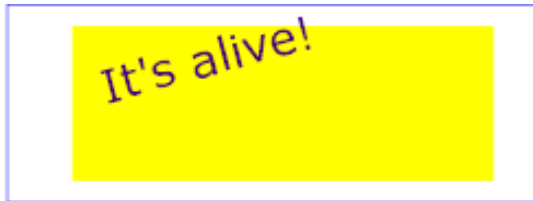
```



At zero seconds



At three seconds



At six seconds



At nine seconds

The sections below describe the various animation attributes and elements.

16.2.4 Attributes to identify the target element for an animation

The following attributes are common to all animation elements and identify the target element for the animation. If the target element is not capable of being a target of the animation, then the animation is ignored.

Refer to the descriptions of the individual animation elements for any restrictions on what types of elements can be targets of particular types of animations.

Schema: animatecommon

```

<define name='svg.AnimateCommon.attr'>
  <ref name='svg.XLink.attr' />
</define>

```

Attribute definitions:

xlink:href = "[<iri>](#)"

An [IRI reference](#) to the element which is the target of this animation and which therefore will be modified over time.

The target element must be part of the [current SVG document fragment](#). If the target element is not part of the current SVG document fragment, then the animation is ignored.

If the **xlink:href** attribute is not provided, then the target element will be the immediate parent element of the current animation element.

<iri> must point to exactly one target element. If **<iri>** points to multiple target elements then it shall be treated as an unsupported value and processed as if the attribute had not been specified. If **<iri>** is the empty string, it shall be treated as if the **xlink:href** attribute was not specified.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0 Animation Modules: Specifying the animation target](#).

Animatable: no.

16.2.5 Attributes to identify the target attribute or property for an animation

The following attributes identify the target attribute or property for the given [target element](#) whose value changes over time.

Schema: animateattribute

```
<define name='svg.AnimateAttributeCommon.attr'>
  <attribute name='attributeName'  svg:animatable='false'  svg:inheritable='false'><text/></attribute>
  <optional>
    <attribute name='attributeType'  svg:animatable='false'  svg:inheritable='false'>
      <choice>
        <value>XML</value>
        <value>CSS</value>
        <value>auto</value>
      </choice>
    </attribute>
  </optional>
</define>
```

Attribute definitions:

attributeName = <attributeName>

Specifies the name of the target attribute. A prefix may be used to indicate the XML namespace for the attribute. The prefix will be interpreted in the scope of the current (i.e., the referencing) animation element.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0 Animation Modules: Specifying the animation target](#).

Animatable: no.

attributeType = "CSS | XML | auto"

Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following (values are case-sensitive):

"CSS"

This specifies that the value of "attributeName" is the name of a CSS property defined as animatable in this specification.

"XML"

This specifies that the value of "attributeName" is the name of an XML attribute defined in the default XML namespace for the target element. If the value for attributeName has a prefix, the implementation must use the associated namespace as defined in the scope of the target element. The attribute must be defined as animatable in this specification.

"auto"

The implementation should match the attributeName to an attribute for the target element. The implementation must first search through the list of CSS properties for a matching property name, and if none is found, search the default XML namespace for the element.

The default value is "auto".

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0 Animation Modules: Specifying the animation target](#).

Animation of presentation attributes is equivalent to animating the corresponding property. Thus, for [properties listed in SVG Tiny 1.2](#), the same effect occurs from animating the presentation attribute with [attributeType="XML"](#) as occurs with animating the corresponding property with [attributeType="CSS"](#).

16.2.6 Paced animation and complex types

Paced animations assume a notion of distance between the various animation values defined by the to, from, by and values attributes. The following table explains how the distance between values of different types should be computed.

Distance is defined for types which can be expressed as a list of values, where each value is a vector of scalars in an n dimensional space. For example, an angle value is a list of one value in a 1 dimensional space and a color is a list of 1 value in a 3 dimensional space.

Animation is based on the computed value of properties. Thus, keywords such as 'inherit' which yield a computed value may be inherited because the computed value is a scalar or list of scalars. For example, fill="inherit" may be animated, because the computed value of fill will be a color.

The following table uses the following notation to describe two values for which a distance should be computed:

$$V_a = \{v_{a0}, v_{a1}, \dots, v_{an}\}$$

$$V_b = \{v_{b0}, v_{b1}, \dots, v_{bn}\}$$

Value Type	Description	Distance	Examples
angle, integer, length, coordinate	single 1 dimensional value. $v_{a0}[0] = \text{scalarA}$ $v_{b0}[0] = \text{scalarB}$	$ V_a V_b = \text{abs}(\text{scalarA} - \text{scalarB})$	x attribute on <rect> stroke-width on <circle>

color	single 3 dimensional value. $v_{a0}[0] = \text{colorA}$ $v_{b0}[0] = \text{colorB}$	$ VaVb = \text{sqrt}(((\text{colorA.getRed()} - \text{colorB.getRed()}))^2 + (\text{colorA.getGreen()} - \text{colorB.getGreen()}))^2 + (\text{colorA.getBlue}() - \text{colorB.getBlue}())^2)$	fill attribute on <ellipse>
list of length	n 1 dimensional values.	$ VaVb = \text{sum}(\text{for } i = 1 \text{ to } n, \text{abs}(v_{ai}[0] - v_{bi}[0])) / n$	stroke-dasharray on <path>
list of points	n 2 dimensional values	$ VaVb = \text{sum}(\text{for } i = 1 \text{ to } n, \text{dist}(v_{ai}, v_{bi})) / n$ $\text{dist}(v_{ai}, v_{bi}) = \text{sqrt}((v_{ai}[0] - v_{bi}[0])^2 + (v_{ai}[1] - v_{bi}[1])^2)$	points on <polygon>
path	n 2 dimensional values where each value is a control point in the path definition.	$ VaVb = \text{sum}(\text{for } i = 1 \text{ to } n, \text{dist}(v_{ai}, v_{bi})) / n$ $\text{dist}(v_{ai}, v_{bi}) = \text{sqrt}((v_{ai}[0] - v_{bi}[0])^2 + (v_{ai}[1] - v_{bi}[1])^2)$	d on <path>
transform list	type: translate one 2 dimensional value $v_{a0}[0] = \text{txa}$ $v_{a0}[1] = \text{tya}$ $v_{b0}[0] = \text{txb}$ $v_{b0}[1] = \text{tyb}$ type: rotate one 1 dimensional value and 1 2 dimensional value $v_{a0}[0] = \text{angleA}$ $v_{a1}[0] = \text{cxa}$ $v_{a1}[1] = \text{cya}$	type: translate $ VaVb = \text{dist}(v_{_a0}, v_{_b0}) = \text{sqrt}((v_{_a0}[0] - v_{_b0}[0])^2 + (v_{_a0}[1] - v_{_b0}[1])^2)$ type: rotate $ VaVb = (\text{abs}(\text{angleA} - \text{angleB}) + \text{sqrt}((v_{_a1}[0] - v_{_b1}[0])^2 + (v_{_a1}[1] - v_{_b1}[1])^2))) / 2$ type: scale $ VaVb = (\text{abs}(\text{scaleXa} - \text{scaleXb}) + \text{abs}(\text{scaleYa} - \text{scaleYb})) / 2$	transform attribute on <g> using <animateTransform>

$v_{b0}[0] = \text{angleB}$ $v_{b1}[0] = \text{cxb}$ $v_{b1}[1] = \text{cyb}$ type: scale two 1 dimensional values $v_{a0}[0] = \text{scaleXa}$ $v_{a1}[0] = \text{scaleYa}$ $v_{b0}[0] = \text{scaleXb}$ $v_{b1}[0] = \text{scaleYb}$ type: skewX, skewY single 1 dimension value $v_{a0}[0] = \text{skewXorYa}$ $v_{b0}[0] = \text{skewXorYb}$	type: skewX, skewY $ V_a V_b = \text{abs}(\text{skewXorYa} - \text{skewXorYb})$
--	--

16.2.7 Attributes to control the timing of the animation

The following attributes are common to all animation elements and control the timing of the animation, including what causes the animation to start and end, whether the animation runs repeatedly, and whether to retain the end state the animation once the animation ends.

The timing attributes also applies to [Media Elements](#).

Schema: animatetiming

```

<define name='svg.AnimateBegin.attr' combine='interleave'>
  <optional><attribute name='begin' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
</define>

<define name='svg.AnimateTimingNoFillNoMinMax.attr' combine='interleave'>
  <ref name='svg.AnimateBegin.attr' />
  <optional><attribute name='dur' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='end' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='repeatCount' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='repeatDur' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional>
    <attribute name='restart' a:defaultValue='always' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>always</value>
        <value>never</value>
        <value>whenNotActive</value>
      </choice>
    </attribute>
  </optional>
</define>

<define name='svg.AnimateTimingNoMinMax.attr' combine='interleave'>
  <ref name='svg.AnimateTimingNoFillNoMinMax.attr' />

```

```

<optional>
  <attribute name='fill' a:defaultValue='remove' svg:animatable='false' svg:inheritable='false'>
    <choice>
      <value>remove</value>
      <value>freeze</value>
    </choice>
  </attribute>
</optional>
</define>

<define name='svg.AnimateTiming.attr' combine='interleave'>
  <ref name='svg.AnimateTimingNoMinMax.attr' />
  <optional><attribute name='min' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='max' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
</define>

```

In the syntax specifications that follow, optional white space is indicated as "S", defined as follows:

```
S ::= (#x20 | #x9 | #xD | #xA)*
```

Attribute definitions:

begin : [begin-value-list](#)

Defines when the element should begin (i.e. become active).

The attribute value is a semicolon separated list of values.

begin-value-list ::= [begin-value](#) (S? ";" S? [begin-value-list](#))?

A semicolon separated list of begin values. The interpretation of a list of begin times is detailed in SMIL 2.0 section on ["Evaluation of begin and end time lists"](#).

begin-value : ([offset-value](#) | [syncbase-value](#) | [event-value](#) | [repeat-value](#) | [accessKey-value](#) | ["indefinite"](#))

Describes the element begin.

offset-value ::= (S? "+" | "-" S?)? ([Clock-value](#))

For SMIL 2.0, this describes the element begin as an offset from an implicit syncbase. For SVG, the implicit syncbase begin is defined to be relative to the document begin. Negative begin times are entirely valid and easy to compute, as long as there is a resolved document begin time.

syncbase-value ::= ([Id-value](#) "." ("begin" | "end")) (S? ("+"|"-") S? [Clock-value](#))?

Describes a [syncbase](#) and an optional offset from that [syncbase](#). The element begin is defined relative to the begin or active end of another animation. A [syncbase](#) consists of an ID reference to another animation element followed by either `begin` or `end` to identify whether to synchronize with the beginning or active end of the referenced animation element.

event-value ::= ([Id-value](#) ".")? ([event-ref](#)) (S? ("+"|"-") S? [Clock-value](#))?

Describes an event and an optional offset that determine the element begin. The animation begin is defined relative to the time that the event is raised. The list of event-symbols available for a given event-base element is listed in the 'Animation event name' column in [Complete list of supported events](#). Details of event-based timing are described in [SMIL 2.0: Unifying Event-based and Scheduled Timing](#).

repeat-value ::= ([Id-value](#) ".")? "repeat(" integer ")" (S? ("+"|"-") S? [Clock-value](#))?

Describes a qualified repeat event. The element begin is defined relative to the time that the repeat event is raised with the specified iteration value.

accessKey-value ::= "accessKey(" character ")" (S? ("+"|"-") S? [Clock-value](#))?

Describes an `accessKey` that determines the element begin. The element begin is defined relative to the time of the keydown event corresponding to the specified key. From a formal processing model perspective, `accessKey` is a keydown event listener on the document which behaves as if `stopPropagation()` and `preventDefault()` have both been invoked. The "character" value can be any of the keyboard event identifier strings listed in Appendix A.2 of the [DOM3 Events](#) specification [[DOM3Events](#)]. user.

"indefinite"

The begin of the animation will be determined by a `beginElement()` method call or a hyperlink targeted to the element.

The animation UDOM methods are described in the [ElementTimeControl](#) interface.

Hyperlink-based timing is described in [SMIL 2.0 Timing and Synchronization: Hyperlinks and timing](#).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'begin' attribute](#).

Animatable: no.

dur : [Clock-value](#) | ["media"](#) | ["indefinite"](#)

Specifies the simple duration.

The attribute value can be either of the following:

[Clock-value](#)

Specifies the length of the simple duration in [presentation time](#). Value must be greater than 0.

["media"](#)

Specifies the simple duration as the intrinsic media duration. This is only valid for elements that define media.

(For SVG's [animation elements](#), if ["media"](#) is specified, the attribute will be ignored.)

["indefinite"](#)

Specifies the simple duration as indefinite.

If the animation does not have a `dur` attribute, the simple duration is indefinite. Note that interpolation will not work if the simple duration is indefinite (although this may still be useful for ["set"](#) elements). Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'dur' attribute](#).

Animatable: no.

end : [end-value-list](#)

Defines an end value for the animation that can constrain the active duration. The attribute value is a semicolon separated list of values.
end-value-list ::= [end-value](#) (S? ";" S? end-value-list)?

A semicolon separated list of end values. The interpretation of a list of end times is detailed below.

end-value : ([offset-value](#) | [syncbase-value](#) | [event-value](#) | [repeat-value](#) | [accessKey-value](#) | "indefinite")

Describes the active end of the animation.

A value of "indefinite" specifies that the end of the animation will be determined by a "endElement()" method call (the animation UDOM methods are described in [ElementTimeControl](#) interface).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see description of [SMIL 2.0: 'end' attribute](#).

Animatable: no.

min : [Clock-value](#) | "media"

Specifies the minimum value of the active duration.

The attribute value can be either of the following:

[Clock-value](#)

Specifies the length of the minimum value of the active duration, measured in local time.

Value must be greater than 0.

"media"

Specifies the minimum value of the active duration as the intrinsic media duration. This is only valid for elements that define media. (For SVG's [animation elements](#), if "media" is specified, the attribute will be ignored.)

The default value for **min** is "0". This does not constrain the active duration at all.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'min' attribute](#).

Animatable: no.

max : [Clock-value](#) | "media"

Specifies the maximum value of the active duration.

The attribute value can be either of the following:

[Clock-value](#)

Specifies the length of the maximum value of the active duration, measured in local time.

Value must be greater than 0.

"media"

Specifies the maximum value of the active duration as the intrinsic media duration. This is only valid for elements that define media. (For SVG's [animation elements](#), if "media" is specified, the attribute will be ignored.)

There is no default value for **max**. This does not constrain the active duration at all.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'max' attribute](#).

Animatable: no.

restart : "always" | "whenNotActive" | "never"

always

The animation can be restarted at any time.

This is the default value.

whenNotActive

The animation can only be restarted when it is not active (i.e. after the active end). Attempts to restart the animation during its active duration are ignored.

never

The element cannot be restarted for the remainder of the current simple duration of the parent time container. (In the case of SVG, since the parent time container is the SVG document fragment, then the animation cannot be restarted for the remainder of the document duration.)

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'restart' attribute](#).

Animatable: no.

repeatCount : numeric value | "indefinite"

Specifies the number of iterations of the animation function. It can have the following attribute values:

numeric value

This is a (base 10) "floating point" numeric value that specifies the number of iterations. It can include partial iterations expressed as fraction values. A fractional value describes a portion of the [simple duration](#). Values must be greater than 0.

"indefinite"

The animation is defined to repeat indefinitely (i.e. until the document ends).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'repeatCount' attribute](#).

Animatable: no.

repeatDur : [Clock-value](#) | "indefinite"

Specifies the total duration for repeat. It can have the following attribute values:

[Clock-value](#)

Specifies the duration in [presentation time](#) to repeat the animation function $\mathcal{F}(t)$.

"indefinite"

The animation is defined to repeat indefinitely (i.e. until the document ends).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'repeatDur' attribute](#).

Animatable: no.

fill : "freeze" | "remove"

This attribute can have the following values:

freeze

The animation effect [F\(t\)](#) is defined to freeze the effect value at the last value of the active duration. The animation effect is "frozen" for the remainder of the document duration (or until the animation is restarted - see [SMIL 2.0: Restarting animation](#)).

remove

The animation effect is removed (no longer applied) when the active duration of the animation is over. After the active end of the animation, the animation no longer affects the target (unless the animation is restarted - see [SMIL 2.0: Restarting animation](#)).

This is the default value.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'fill' attribute](#).

[Animatable](#): no.

The [SMIL 2.0](#) specification defines the detailed processing rules associated with the above attributes. Except for any SVG-specific rules explicitly mentioned in this specification, the [SMIL 2.0](#) specification is the normative definition of the processing rules for the above attributes.

Clock values

Clock values have a subsetted syntax of the clock values syntax in [SMIL 2.0: Clock values](#):

```
Clock-val      ::= Timecount-val
Timecount-val  ::= Timecount ("." Fraction)? (Metric)?
Metric         ::= "s" | "ms"
Fraction       ::= DIGIT+
Timecount      ::= DIGIT+
DIGIT          ::= [0-9]
```

For Timecount values, the default metric suffix is "s" (for seconds). No embedded white space is allowed in clock values, although leading and trailing white space characters will be ignored.

Clock values describe [presentation time](#).

The following are examples of legal clock values:

- Timecount values:

30s = 30 seconds

5ms = 5 milliseconds

12.467 = 12 seconds and 467 milliseconds

Fractional values are just (base 10) floating point definitions of seconds. Thus:

00.5s = 500 milliseconds

16.2.8 Attributes that define animation values over time

The following attributes are common to elements ['animate'](#), ['animateMotion'](#), ['animateColor'](#) and ['animateTransform'](#). These attributes define the values that are assigned to the target attribute or property over time. The attributes below provide control over the relative timing of keyframes and the interpolation method between discrete values.

Schema: animatevalue

```
<define name='svg:AnimateToCommon.attr' combine='interleave'>
  <optional><attribute name='to' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
</define>

<define name='svg:AnimateValueCommon.attr'>
  <ref name='svg:AnimateToCommon.attr' />
  <optional>
    <attribute name='calcMode' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>discrete</value>
        <value>linear</value>
        <value>paced</value>
        <value>spline</value>
      </choice>
    </attribute>
  </optional>
  <optional><attribute name='values' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='keyTimes' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='keySplines' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='from' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
  <optional><attribute name='by' svg:animatable='false' svg:inheritable='false'><text/></attribute></optional>
</define>
```

Attribute definitions:

calcMode = "discrete | linear | paced | spline"

Specifies the interpolation mode for the animation. This can take any of the following values. The default mode is "linear", however if the attribute does not support linear interpolation (e.g. for strings), the `calcMode` attribute is ignored and discrete interpolation is used.

discrete

This specifies that the animation function will jump from one value to the next without any interpolation.

linear

Simple linear interpolation between values is used to calculate the animation function. Except for '[animateMotion](#)', this is the default `calcMode`.

paced

Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.). If "paced" is specified, any `keyTimes` or `keySplines` will be ignored. For '[animateMotion](#)', this is the default `calcMode`.

spline

Interpolates from one value in the `values` list to the next according to a time function defined by a cubic Bzier spline. The points of the spline are defined in the `keyTimes` attribute, and the control points for each interval are defined in the `keySplines` attribute.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'calcMode' attribute](#).

Animatable: no.

values = "<list>"

A semicolon-separated list of one or more values. Vector-valued attributes are supported using the vector syntax of the `attributeType` domain. Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'values' attribute](#).

Animatable: no.

keyTimes = "<list>"

A semicolon-separated list of time values used to control the pacing of the animation. Each time in the list corresponds to a value in the `values` attribute list, and defines when the value is used in the animation function. Each time value in the `keyTimes` list is specified as a floating point value between 0 and 1 (inclusive), representing a proportional offset into the simple duration of the animation element.

If a list of `keyTimes` is specified, there must be exactly as many values in the `keyTimes` list as in the `values` list.

Each successive time value must be greater than or equal to the preceding time value.

The `keyTimes` list semantics depends upon the interpolation mode:

- o For linear and spline animation, the first time value in the list must be 0, and the last time value in the list must be 1. The `keyTime` associated with each value defines when the value is set; values are interpolated between the `keyTimes`.
- o For discrete animation, the first time value in the list must be 0. The time associated with each value defines when the value is set; the animation function uses that value until the next time defined in `keyTimes`.

If the interpolation mode is "paced", the `keyTimes` attribute is ignored.

If there are any errors in the `keyTimes` specification (bad values, too many or too few values) the `keyTimes` attribute has an unsupported value and is processed as if the attribute had not been specified

If the simple duration is indefinite, any `keyTimes` specification will be ignored.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'keyTimes' attribute](#).

Animatable: no.

keySplines = "<list>"

A set of Bzier control points associated with the `keyTimes` list, defining a cubic Bzier function that controls interval pacing. The attribute value is a semicolon separated list of control point descriptions. Each control point description is a set of four values: `x1 y1 x2 y2`, describing the Bzier control points for one time segment. The `keyTimes` values that define the associated segment are the Bzier "anchor points", and the `keySplines` values are the control points. Thus, there must be one fewer sets of control points than there are `keyTimes`.

The values must all be in the range 0 to 1.

This attribute is ignored unless the `calcMode` is set to "spline".

If there are any errors in the `keySplines` specification (bad values, too many or too few values) the `keySplines` attribute has an unsupported value and is processed as if the attribute had not been specified

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'keySplines' attribute](#).

Animatable: no.

from = "<value>"

Specifies the starting value of the animation.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'from' attribute](#).

Animatable: no.

to = "<value>"

Specifies the ending value of the animation.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL Animation: 'to' attribute](#).

Animatable: no.

by = "<value>"

Specifies a relative offset value for the animation.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'by' attribute](#).

Animatable: no.

The [SMIL 2.0](#) specification defines the detailed processing rules associated with the above attributes. Except for any SVG-specific rules explicitly mentioned in this specification, the [SMIL 2.0](#) specification is the normative definition of the processing rules for the above attributes.

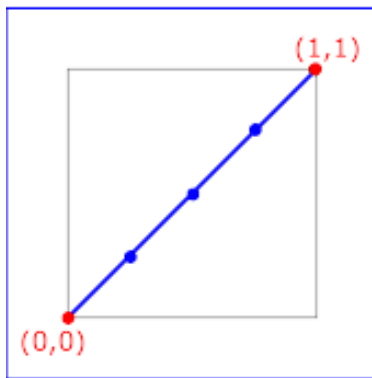
The animation values specified in the animation element must be legal values for the specified attribute. Leading and trailing white space, and white space before and after semicolon separators, will be ignored.

All values specified must be legal values for the specified attribute (as defined in the associated namespace). If any values are not legal, they are considered to be unsupported and is processed as if they had not been specified.

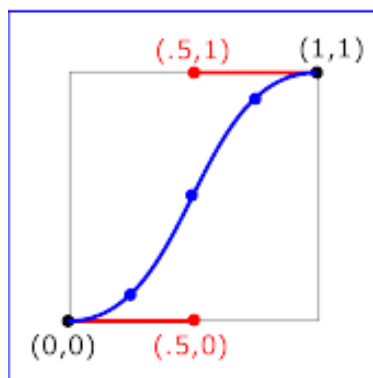
If a list of values is used, the animation will apply the values in order over the course of the animation. If a list of *values* is specified, any *from*, *to* and *by* attribute values are ignored.

The processing rules for the variants of *from/by/to* animations are described in [Animation function values](#).

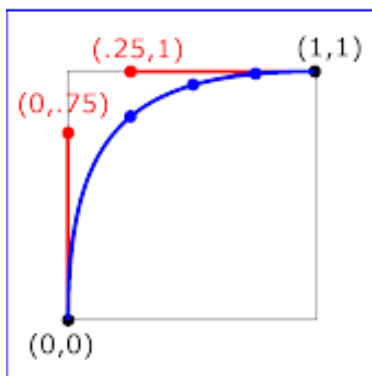
The following figure illustrates the interpretation of the `keySplines` attribute. Each diagram illustrates the effect of `keySplines` settings for a single interval (i.e. between the associated pairs of values in the `keyTimes` and `values` lists). The horizontal axis can be thought of as the input value for the *unit progress* of interpolation within the interval - i.e. the pace with which interpolation proceeds along the given interval. The vertical axis is the resulting value for the *unit progress*, yielded by the `keySplines` function. Another way of describing this is that the horizontal axis is the input *unit time* for the interval, and the vertical axis is the output *unit time*. See also the section [Timing and real-world clock times](#).



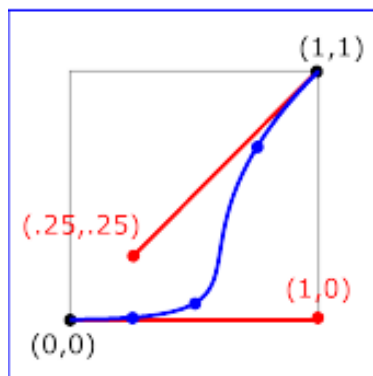
keySplines="0011" (the default)



keySplines=".50.51"



keySplines="0.75.251"



keySplines="10.25.25"

Examples of keySplines

To illustrate the calculations, consider the simple example:

```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
  calcMode="spline" keySplines={as in table} />
```

Using the keySplines values for each of the four cases above, the approximate interpolated values as the animation proceeds are:

keySplines values	Initial value	After 1s	After 2s	After 3s	Final value
0 0 1 1	10.0	12.5	15.0	17.5	20.0
.5 0 .5 1	10.0	11.0	15.0	19.0	20.0
0 .75 .25 1	10.0	18.0	19.3	19.8	20.0
1 0 .25 .25	10.0	10.1	10.6	16.9	20.0

For a formal definition of Bzier spline calculation, see [\[FOLEY-VANDAM\]](#).

16.2.9 Attributes that control whether animations are additive

It is frequently useful to define animation as an offset or delta to an attribute's value, rather than as absolute values. A simple "grow" animation can increase the width of an object by 10 pixels:

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
    additive="sum"/>
</rect>
```

It is frequently useful for repeated animations to build upon the previous results, accumulating with each iteration. The following example causes the rectangle to continue to grow with each repeat of the animation:

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
    additive="sum" accumulate="sum" repeatCount="5"/>
</rect>
```

At the end of the first repetition, the rectangle has a width of 30 pixels. At the end of the second repetition, the rectangle has a width of 40 pixels. At the end of the fifth repetition, the rectangle has a width of 70 pixels.

For more information about additive animations, see [SMIL 2.0: Additive animation](#). For more information on cumulative animations, see [SMIL 2.0: Controlling behavior of repeating animation - Cumulative animation](#).

The following attributes are common to elements ['animate'](#), ['animateMotion'](#), ['animateColor'](#) and ['animateTransform'](#).

Schema: animateaddition

```
<define name='svg.AnimateAdditionCommon.attr'>
  <optional>
    <attribute name='additive' a:defaultValue='replace' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>replace</value>
        <value>sum</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='accumulate' a:defaultValue='none' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>none</value>
        <value>sum</value>
      </choice>
    </attribute>
  </optional>
</define>
```

Attribute definitions:

additive = "replace | sum"

Controls whether or not the animation is additive.

sum

Specifies that the animation will add to the underlying value of the attribute and other lower priority animations.

replace

Specifies that the animation will override the underlying value of the attribute and other lower priority animations. This is the default, however the behavior is also affected by the animation value attributes `by` and `to`, as described in [SMIL 2.0: Simple animation functions specified by from, to and by](#).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'additive' attribute](#).

Animatable: no.

accumulate = "none | sum"

Controls whether or not the animation is cumulative.

sum

Specifies that each repeat iteration after the first builds upon the last value of the previous iteration.

none

Specifies that repeat iterations are not cumulative. This is the default.

This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.

Cumulative animation is not defined for "to animation".

This attribute will be ignored if the animation function is specified with only the `to` attribute.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'accumulate' attribute](#).

Animatable: no.

16.2.10 Inheritance

SVG allows both attributes and properties to be animated. If a given attribute or property is inheritable by descendants, then animations on a parent element such as a `'g'` element has the effect of propagating the attribute or property animation values to descendant elements as the animation proceeds; thus, descendant elements can inherit animated attributes and properties from their ancestors.

16.2.11 The 'animate' element

The `'animate'` element is used to animate a single attribute or property over time. For example, to make a rectangle repeatedly fade away over 5 seconds, you can specify:

```
<rect>
  <animate attributeType="CSS" attributeName="fill-opacity"
    from="1" to="0" dur="5s" repeatCount="indefinite" />
</rect>
```

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'animate' element](#).

Schema: animate

```
<define name='animate'>
  <element name='animate'>
    <ref name='animate.AT' />
    <zeroOrMore><ref name='animateCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='animate.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.AnimateCommon.attr' />
  <ref name='svg.AnimateAttributeCommon.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateValueCommon.attr' />
  <ref name='svg.AnimateAdditionCommon.attr' />
</define>
```

For a list of attributes and properties that can be animated using the `'animate'` element, see [Attributes and properties that can be animated](#).

16.2.12 The 'set' element

The **'set'** element provides a simple means of just setting the value of an attribute for a specified duration. It supports all attribute types, including those that cannot reasonably be interpolated, such as string and boolean values. The **'set'** element is non-additive. The additive and accumulate attributes are not allowed, and will be ignored if specified.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'set' element](#).

Schema: set

```
<define name='set'>
  <element name='set'>
    <ref name='set.AT' />
    <zeroOrMore><ref name='animateCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='set.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.AnimateCommon.attr' />
  <ref name='svg.AnimateAttributeCommon.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateToCommon.attr' />
</define>
```

Attribute definitions:

to = "<value>"

Specifies the value for the attribute during the duration of the **'set'** element. The argument value must match the attribute type.

[Animatable](#): no.

For a list of attributes and properties that can be animated using the **'set'** element, see [Attributes and properties that can be animated](#).

16.2.13 The **'animateMotion'** element

The **'animateMotion'** element causes a referenced element to move along a motion path.

The following lists all of the elements which can be animated by the **'animateMotion'** element:

- ['g'](#)
- ['defs'](#)
- ['use'](#)
- ['image'](#)
- ['switch'](#)
- ['path'](#)
- ['rect'](#)
- ['circle'](#)
- ['ellipse'](#)
- ['line'](#)
- ['polyline'](#)
- ['polygon'](#)
- ['text'](#)
- ['textArea'](#)
- ['animation'](#)
- ['video'](#)
- ['a'](#)
- ['foreignObject'](#)

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'animateMotion' element](#).

Schema: animateMotion

```
<define name='animateMotion'>
  <element name='animateMotion'>
    <ref name='animateMotion.AT' />
    <zeroOrMore>
      <ref name='animateCommon.CM' />
    </zeroOrMore>
    <optional>
      <ref name='mpath' />
    </optional>
    <zeroOrMore>
      <ref name='animateCommon.CM' />
    </zeroOrMore>
  </element>
</define>
```

```

</zeroOrMore>
</element>
</define>

<define name='animateMotion.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.AnimateCommon.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateAdditionCommon.attr' />
  <ref name='svg.AnimateValueCommon.attr' />
  <ref name='svg.AnimateTypeCommon.attr' />
  <optional><attribute name='path' svg:animatable='false' svg:inheritable='false'><text /></attribute></optional>
  <optional><attribute name='keyPoints' svg:animatable='false' svg:inheritable='false'><text /></attribute></optional>
  <optional><attribute name='rotate' svg:animatable='false' svg:inheritable='false'><text /></attribute></optional>
  <optional><attribute name='origin' svg:animatable='false' svg:inheritable='false'><text /></attribute></optional>
</define>

```

Attribute definitions:

calcMode = "discrete | linear | paced | spline"

Specifies the interpolation mode for the animation. Refer to general description of the [calcMode](#) attribute above. The only difference is that the default value for the [calcMode](#) for **'animateMotion'** is **paced**. See [SMIL 2.0: 'calcMode' attribute for 'animateMotion'](#).

Animatable: no.

path = "<path-data>"

The motion path, expressed in the same format and interpreted the same way as the [d=](#) attribute on the **'path'** element. The effect of a motion path animation is to add a supplemental transformation matrix onto the CTM for the referenced object which causes a translation along the x- and y-axes of the current user coordinate system by the computed X and Y values computed over time.

Animatable: no.

keyPoints = "<list-of-numbers>"

keyPoints takes a semicolon-separated list of floating point values between 0 and 1 and indicates how far along the motion path the object shall move at the moment in time specified by corresponding [keyTimes](#) value. Distance calculations use the user agent's [distance along the path](#) algorithm. Each progress value in the list corresponds to a value in the [keyTimes](#) attribute list.

If a list of [keyPoints](#) is specified, there must be exactly as many values in the [keyPoints](#) list as in the [keyTimes](#) list.

If there are any errors in the [keyPoints](#) specification (bad values, too many or too few values) the [keyPoints](#) attribute has an unsupported value and is processed as if the attribute had not been specified

Animatable: no.

rotate = "<angle> | auto | auto-reverse"

auto indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path. **auto-reverse** indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path plus 180 degrees. An actual angle value can also be given, which represents an angle relative to the x-axis of current user coordinate system. The [rotate](#) attribute adds a supplemental transformation matrix onto the CTM to apply a rotation transformation about the origin of the current user coordinate system. The rotation transformation is applied after the supplemental translation transformation that is computed due to the [path](#) attribute. The default value is 0.

Animatable: no.

origin = "default"

The [origin](#) attribute is defined in [SMIL 2.0: 'origin'](#). It has no effect in SVG.

Animatable: no.

For **'animateMotion'**, the specified values for [from](#), [by](#), [to](#) and [values](#) consists of x, y coordinate pairs, with a single comma and/or white space separating the x coordinate from the y coordinate. For example, [from](#)="33,15" specifies an x coordinate value of 33 and a y coordinate value of 15.

If provided, the [values](#) attribute must consists of a list of x, y coordinate pairs. Coordinate values are separated by at least one white space character or a comma. Additional white space around the separator is allowed. For example, [values](#)="10,20;30,20;30,40" or [values](#)="10mm,20mm;30mm,20mm;30mm,40mm". Each coordinate represents a [length](#). Attributes [from](#), [by](#), [to](#) and [values](#) specify a shape on the current canvas which represents the motion path.

Two options are available which allow definition of a motion path using any of SVG's [path data](#) commands:

- the [path](#) attribute defines a motion path directly on **'animateMotion'** element using any of SVG's [path data](#) commands.
- the **'mpath'** sub-element provides the ability to reference an external **'path'** element as the definition of the motion path.

Note that SVG's [path data](#) commands can only contain values in user space, whereas [from](#), [by](#), [to](#) and [values](#) can specify coordinates in user space or using unit identifiers. See [Units](#).

The various (x,y) points of the shape provide a supplemental transformation matrix onto the CTM for the referenced object which causes a translation along the x- and y-axes of the current user coordinate system by the (x,y) values of the shape computed over time. Thus, the referenced object is translated over time by the offset of the motion path relative to the origin of the current user coordinate system. The supplemental transformation is applied on top of any transformations due to the target element's [transform](#) attribute or any animations on that attribute due to **'animateTransform'** elements on the target element.

The [additive](#) and [accumulate](#) attributes apply to **'animateMotion'** elements. Multiple **'animateMotion'** elements all simultaneously referencing the same

target element can be additive with respect to each other; however, the transformations which result from the 'animateMotion' elements are always supplemental to any transformations due to the target element's [transform](#) attribute or any 'animateTransform' elements.

The default calculation mode ([calcMode](#)) for [animateMotion](#) is "paced". This will produce constant velocity motion along the specified path. Note that while [animateMotion](#) elements can be additive, it is important to observe that the addition of two or more "paced" (constant velocity) animations might not result in a combined motion animation with constant velocity.

When a [path](#) is combined with "discrete", "linear" or "spline" [calcMode](#) settings, and if attribute [keyPoints](#) is not provided, the number of values is defined to be the number of points defined by the path, unless there are "move to" commands within the path. A "move to" command within the [path](#) (i.e. other than at the beginning of the [path](#) description) A "move to" command does not count as an additional point when dividing up the duration, or when associating [keyTimes](#), [keySplines](#) and [keyPoints](#) values. When a [path](#) is combined with a "paced" [calcMode](#) setting, all "move to" commands are considered to have 0 length (i.e. they always happen instantaneously), and is not considered in computing the pacing.

For more flexibility in controlling the velocity along the motion path, the [keyPoints](#) attribute provides the ability to specify the progress along the motion path for each of the [keyTimes](#) specified values. If specified, [keyPoints](#) causes [keyTimes](#) to apply to the values in [keyPoints](#) rather than the points specified in the [values](#) attribute array or the points on the [path](#) attribute.

The override rules for 'animateMotion' are as follows. Regarding the definition of the motion path, the 'mpath' element overrides the the [path](#) attribute, which overrides [values](#), which overrides [from/by/to](#). Regarding determining the points which correspond to the [keyTimes](#) attributes, the [keyPoints](#) attribute overrides [path](#), which overrides [values](#), which overrides [from/by/to](#).

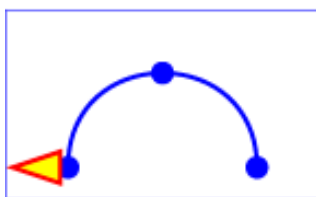
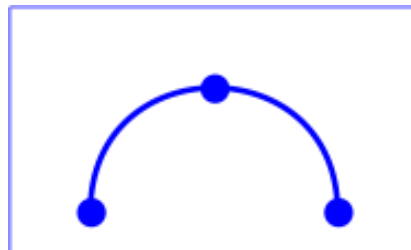
At any time t within a motion path animation of duration dur , the computed coordinate (x,y) along the motion path is determined by finding the point (x,y) which is t/dur distance along the motion path using the user agent's [distance along the path](#) algorithm.

The following example demonstrates the supplemental transformation matrices that are computed during a motion path animation.

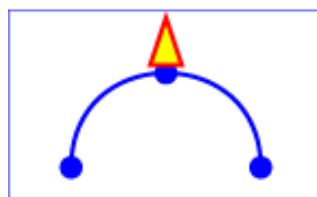
Example [animMotion01](#) shows a triangle moving along a motion path.

Example: 19_02.svg

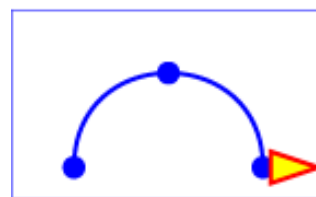
```
<?xml version="1.0"?>
<svg width="5cm" height="3cm" viewBox="0 0 500 300"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
  xmlns:xlink="http://www.w3.org/1999/xlink" >
  <desc>Example animMotion01 - demonstrate motion animation computations</desc>
  <rect x="1" y="1" width="498" height="298"
    fill="none" stroke="blue" stroke-width="2" />
  <!-- Draw the outline of the motion path in blue, along
    with three small circles at the start, middle and end. -->
  <path xml:id="path1" d="M100,250 C 100,50 400,50 400,250"
    fill="none" stroke="blue" stroke-width="7.06" />
  <circle cx="100" cy="250" r="17.64" fill="blue" />
  <circle cx="250" cy="100" r="17.64" fill="blue" />
  <circle cx="400" cy="250" r="17.64" fill="blue" />
  <!-- Here is a triangle which will be moved about the motion path.
    It is defined with an upright orientation with the base of
    the triangle centered horizontally just above the origin. -->
  <path d="M-25,-12.5 L25,-12.5 L 0,-87.5 z"
    fill="yellow" stroke="red" stroke-width="7.06" >
  <!-- Define the motion path animation -->
  <animateMotion dur="6s" repeatCount="indefinite" rotate="auto" >
    <mpath xlink:href="#path1"/>
  </animateMotion>
</path>
</svg>
```



At zero seconds



At three seconds



At six seconds

The following table shows the supplemental transformation matrices that are applied to achieve the effect of the motion path animation.

	After 0s	After 3s	After 6s
Supplemental transform due to movement along motion path	translate(100,250)	translate(250,100)	translate(400,250)
Supplemental transform due to rotate="auto"	rotate(-90)	rotate(0)	rotate(90)

16.2.14 The 'mpath' element

The '**mpath**' element is a sub element to the '**animateMotion**' element (its only place in the document tree is as a child of an '**animateMotion**'). '**mpath**' reference an external '**path**' element that will serve as the definition of the motion path.

Example:

Example: mpath01.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="100%" height="100%" viewBox="0 0 80 60">
  <desc>mpath example</desc>
  <path xml:id="mpathRef" d="M15,43 C15,43 36,20 65,33" fill="none" stroke="black" stroke-width="1"/>
  <animateMotion begin="0s" dur="6s" calcMode="linear" fill="freeze">
    <mpath xlink:href="#mpathRef"/>
  </animateMotion>
</svg>
```

Schema: mpath

```
<define name='mpath'>
  <element name='mpath'>
    <ref name='mpath.AT' />
    <zeroOrMore><ref name='svg.Desc.group' /></zeroOrMore>
  </element>
</define>

<define name='mpath.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.XLinkRequired.attr' />
</define>
```

Attribute definitions:

xlink:href = "[<iri>](#)"

A [URI reference](#) to the '**path**' element which defines the motion path.

Animatable: no.

16.2.15 The 'animateColor' element

The '**animateColor**' element specifies a color transformation over time.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the [SMIL 2.0](#) specification. In particular, see [SMIL 2.0: 'animateColor' element](#).

Schema: animateColor

```
<define name='animateColor'>
  <element name='animateColor'>
    <ref name='animateColor.AT' />
    <zeroOrMore><ref name='animateCommon.CM' /></zeroOrMore>
  </element>
</define>
```

```

<define name='animateColor.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.AnimateCommon.attr' />
  <ref name='svg.AnimateAttributeCommon.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateValueCommon.attr' />
  <ref name='svg.AnimateAdditionCommon.attr' />
</define>

```

The [from](#), [by](#) and [to](#) attributes take color values, where each color value is expressed using the following syntax (the same syntax as used in SVG's properties that can take color values):

[<color>](#)

The [values](#) attribute for the 'animateColor' element consists of a semicolon-separated list of color values, with each color value expressed in the above syntax.

Out of range color values can be provided, but user agent processing will be implementation dependent. User agents should clamp color values to allow color range values as late as possible, but note that system differences might preclude consistent behavior across different systems.

For a list of attributes and properties that can be animated using the 'animateColor' element, see [Attributes and properties that can be animated](#).

16.2.16 The 'animateTransform' element

The 'animateTransform' element animates a transformation attribute on a target element, thereby allowing animations to control translation, scaling, rotation and/or skewing.

Schema: animateTransform

```

<define name='animateTransform'>
  <element name='animateTransform'>
    <ref name='animateTransform.AT' />
    <zeroOrMore><ref name='animateCommon.CM' /></zeroOrMore>
  </element>
</define>

<define name='animateTransform.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.AnimateCommon.attr' />
  <ref name='svg.AnimateAttributeCommon.attr' />
  <ref name='svg.AnimateTiming.attr' />
  <ref name='svg.AnimateValueCommon.attr' />
  <ref name='svg.AnimateAdditionCommon.attr' />
  <ref name='svg.AnimateTypeCommon.attr' />
</define>

```

Attribute definitions:

type = "translate | scale | rotate | skewX | skewY"

Indicates the type of transformation which is to have its values change over time. If **type** has an unsupported value (e.g. **type**="foo" or **type**="ref (svg)") the 'animateTransform' element is ignored.

[Animatable](#): no.

The [from](#), [by](#) and [to](#) attributes take a value expressed using the same syntax that is available for the given transformation type:

- For a **type**="translate", each individual value is expressed as **<tx>** [**<ty>**].
- For a **type**="scale", each individual value is expressed as **<sx>** [**<sy>**].
- For a **type**="rotate", each individual value is expressed as **<rotate-angle>** [**<cx>** **<cy>**].
- For a **type**="skewX" and **type**="skewY", each individual value is expressed as **<skew-angle>**.

(See [The transform attribute](#).)

The [values](#) attribute for the 'animateTransform' element consists of a semicolon-separated list of values, where each individual value is expressed as described above for [from](#), [by](#) and [to](#).

If [from](#) or [to](#) attributes are not specified, the *underlying value* (see [SMIL discussion of animation function values](#)) is the corresponding identity transformation value. Thus, the underlying value for

- **type**="translate" is tx = ty = 0

- **type="scale"** is $s_x = s_y = 1$
- **type="rotate"** is $\text{rotate-angle} = c_x = c_y = 0$
- **type="skew"** is $\text{skew-angle} = 0$

If **calcMode** has the value **paced**, then the "distance" for the transformation is calculated consisting of the sum of the absolute values of the differences between each pair of values as further described in [Paced animations and complex types](#).

When an animation is active, the effect of non-additive '**animateTransform**' (i.e., **additive="replace"**) is to replace the given attribute's value with the transformation defined by the '**animateTransform**'. The effect of additive (i.e., **additive="sum"**) is to post-multiply the transformation matrix corresponding to the transformation defined by this '**animateTransform**'. To illustrate:

```
<rect transform="skewX(30)"...>
  <animateTransform attributeName="transform" attributeType="XML"
    type="rotate" from="0" to="90" dur="5s"
    additive="replace" fill="freeze"/>
  <animateTransform attributeName="transform" attributeType="XML"
    type="scale" from="1" to="2" dur="5s"
    additive="replace" fill="freeze"/>
</rect>
```

In the code snippet above, because the both animations have **additive="replace"**, the first animation overrides the transformation on the rectangle itself and the second animation overrides the transformation from the first animation; therefore, at time 5 seconds, the visual result of the above two animations would be equivalent to the following static rectangle:

```
<rect transform="scale(2)" ... />
```

whereas in the following example:

```
<rect transform="skewX(30)"...>
  <animateTransform attributeName="transform" attributeType="XML"
    type="rotate" from="0" to="90" dur="5s"
    additive="sum" fill="freeze"/>
  <animateTransform attributeName="transform" attributeType="XML"
    type="scale" from="1" to="2" dur="5s"
    additive="sum" fill="freeze"/>
</rect>
```

In this code snippet, because the both animations have **additive="sum"**, the first animation post-multiplies its transformation to any transformations on the rectangle itself and the second animation post-multiplies its transformation to any transformation from the first animation; therefore, at time 5 seconds, the visual result of the above two animations would be equivalent to the following static rectangle:

```
<rect transform="skewX(30) rotate(90) scale(2)" ... />
```

For a list of attributes and properties that can be animated using the '**animateTransform**' element, see [Attributes and properties that can be animated](#).

16.2.17 Attributes and properties that can be animated

Each attribute or property within this specification indicates whether or not it can be animated by SVG's animation elements. Animatable attributes and properties are designated as follows:

Animatable: yes.

whereas attributes and properties that cannot be animated are designated:

Animatable: no.

SVG has a defined set of [basic data types](#) for its various supported attributes and properties. For those attributes and properties that can be animated, the following table indicates which animation elements can be used to animate each of the basic data types. If a given attribute or property has a computed value which is a keyword (which are not additive) or numeric values (which are additive), then additive animations are possible if the subsequent animation uses a numeric value even if the base animation uses a keyword value; however, if the subsequent animation uses a keyword value which does not result in a numeric computed value, additive animation is not possible.

Data type	Additive?	<u>'animate'</u>	<u>'set'</u>	<u>'animate Color'</u>	<u>'animate Transform'</u>	Notes
<color>	yes	yes	yes	yes	no	Only additive if the computed value is a color.

<coordinate>	yes	yes	yes	no	no	
<integer>	yes	yes	yes	no	no	
<length>	yes	yes	yes	no	no	
<list of xxx>	no	yes	yes	no	no	
<number>	yes	yes	yes	no	no	
<paint>	yes	yes	yes	yes	no	Only RGB color values are additive.
<transform-list>	yes	no	no	no	yes	Additive means that a transformation is post-multiplied to the base set of transformations.
<iri>	no	yes	yes	no	no	
All other data types used in animatable attributes and properties	no	yes	yes	no	no	

Any deviation from the above table or other special note about the animation capabilities of a particular attribute or property is included in the section of the specification where the given attribute or property is defined.

16.3 Animation using the SVG DOM

Example `dom_animate` shows a simple animation using the DOM.

Example: `dom_animate.svg`

```
<?xml version="1.0"?>
<svg width="4cm" height="2cm" viewBox="0 0 400 200" xml:id="root"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>A simple animation using the uDOM and the SVGTimer interface of the uDOM.</desc>
  <script type="application/ecmascript"><![CDATA[
    var timeValue = 0;
    var timerIncrement = 50;
    var maxTime = 5000;
    var textElement;
    var svgRoot;
    var mytimer;

    function init() {
      textElement = document.getElementById("svgtext");
      svgRoot = document.getElementById("root");
      launchTimer();
    }

    function launchTimer() {
      mytimer = createTimer();
      // Fire timer event as soon as possible
      mytimer.delay = 0;
      // Timer event must be sent every 50 ms
      mytimer.interval = 50;
      // This instruction is clear enough, isn't it ?
      mytimer.start();
    }

    function showAndGrowElement() {
      timeValue += timerIncrement;
      // Scale the text string gradually until it is 20 times larger
      scalefactor = (timeValue * 20.) / maxTime;
      var matrix = svgRoot.createSVGMatrixComponents(scalefactor, 0, 0, scalefactor, 0, 0);
      textElement.setMatrixTrait("transform", matrix);
      // Make the string more opaque
      var opacityFactor = timeValue / maxTime;
      textElement.setFloatTrait("fill-opacity", opacityFactor);

      if (timeValue >= maxTime)
        mytimer.stop();
    }
  ]]></script>

  <handler type="application/ecmascript" ev:event="load">
    init();
  </handler>
```

```

<handler type="application/ecmascript" ev:event="timer">
  if (evt.target == mytimer) {
    showAndGrowElement();
  }
</handler>

<rect x="1" y="1" width="398" height="198" fill="none" stroke="blue" stroke-width="2"/>
<g transform="translate(50,150)" font-size="7" stroke="none">
  <text fill="red" fill-opacity="0" xml:id="svgtext">SVG</text>
</g>

</svg>

```



At zero seconds



At 2.5 seconds



At five seconds

The above SVG file contains a text element that says "SVG". The animation loops for 5 seconds. The text string starts out small and transparent and grows to be large and opaque. Here is an explanation of how this example works:

- This is the first **'handler'** in the example:

```

<handler type="application/ecmascript" ev:event="load">
  init();
</handler>

```

Once the document has been fully loaded and processed, this **'handler'** invokes the ECMAScript function `init()`.

- The `'script'` element defines the ECMAScript which makes the animation happen. The `init()` function is only called once to give a value to global variables `textElement` and `svgRoot` and to create and launch an `SVGTimer` object via the `launchTimer()` method. `showAndGrowElement()` sets the `transform` and `fill-opacity` attributes on the text element to new values each time it is called.
- Because a `'timer'` event is triggered regularly, `showAndGrowElement()` is called every 50 milliseconds from the second **'handler'** in the example.

```

<handler type="application/ecmascript" ev:event="timer">
  if (evt.target == mytimer) {
    showAndGrowElement();
  }
</handler>

```

`showAndGrowElement()` checks each time it is invoked whether the maximum duration of the animation has been reached and calls the `stop()` method on the `timer` object when this happens, so that the animation stops.

If an attribute/property value is modified while an animation element is animating the same attribute/property, the animations are required to adjust dynamically to the new value.

16.4 Timed Animation Module

The Timed Animation Module contains the following elements:

- [animate](#)
- [animateColor](#)
- [animateMotion](#)
- [animateTransform](#)
- [mpath](#)
- [set](#)

[RNG] [Feature String]

17 Fonts

Contents

- 17.1 [Introduction](#)
 - 17.1.1 [Describing fonts available to SVG](#)
 - 17.1.2 [Defining fonts in SVG](#)
- 17.2 [Overview of SVG fonts](#)
- 17.3 [The 'font' element](#)
- 17.4 [The 'glyph' element](#)
- 17.5 [The 'missing-glyph' element](#)
- 17.6 [Glyph selection rules](#)
- 17.7 [The 'hkern' element](#)
- 17.8 [Describing a font](#)
 - 17.8.1 [Overview of font descriptions](#)
 - 17.8.2 [The 'font-face' element](#)
 - 17.8.3 [The 'font-face-src' element](#)
 - 17.8.4 [The 'font-face-uri' element](#)
 - 17.8.5 [The 'font-face-name' element](#)
- 17.9 [Font Module](#)

17.1 Introduction

Reliable delivery of fonts is a requirement for SVG. Designers need to create SVG content with arbitrary fonts and know that the same graphical result will appear when the content is viewed by all end users, even when end users do not have the necessary fonts installed on their computers. This parallels the print world, where the designer uses a given font when authoring a drawing for print, and the graphical content appears exactly the same in the printed version as it appeared on the designer's authoring system.

Historically, one approach has been to convert all text to paths representing the glyphs used. This preserves the visual look, but means that the text is lost; it cannot be dynamically updated and is not accessible. Another approach is to hope that a font with a given name is available to all renderers. This assumption does not work well on the desktop and works very badly in a heterogeneous mobile environment. SVG solves this problem by allowing the text to be converted to paths, but storing those paths as an SVG font. The text is retained and remains dynamically modifiable and accessible.

17.1.1 Describing fonts available to SVG

SVG utilizes an XML version of the [WebFonts](#) facility defined in the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)] to reference fonts. Described fonts may be in a variety of different formats. By describing key details of the font such as its family name, weight, whether it is italic and so on, text can continue to use the font properties without having to explicitly indicate the font that is to be used for each span of text.

17.1.2 Defining fonts in SVG

One disadvantage to the [WebFont](#) facility to date is that specifications such as [[CSS2](#)] do not require support of particular font formats. The result is that different implementations support different Web font formats, thereby making it difficult for Web site creators to post a single Web site using WebFonts that work across all user agents.

To provide a common font format for SVG that is guaranteed to be supported by all [conforming SVG viewers](#), SVG also defines a font format, in SVG, which uses the same geometric mechanism for glyphs as is used by the SVG path element. This facility is called **SVG fonts**. SVG implementations must support the SVG font format, and may also support other formats. [WebFonts](#) may be contained in the SVG document which uses them, or stored in a separate document (for example, to allow sharing of the same font by multiple SVG documents).

Taken together, these two mechanisms ensure reliable delivery of font data to end users, preserving graphical richness and enabling accessible access to the textual data. . In a common scenario, SVG authoring applications generate compressed, subsetted [WebFonts](#) for all text elements used by a given SVG document fragment. SVG fonts can improve the semantic richness of graphics that represent text. For example, many company logos consist of the company name drawn artistically. In some cases, [accessibility](#) may be enhanced by expressing the logo as a series of glyphs in an SVG font and then rendering the logo as a **'text'** element which references this font.

17.2 Overview of SVG fonts

An **SVG font** is a font defined using SVG's **'font'** element.

The purpose of SVG fonts is to allow for delivery of glyph outlines in display-only environments. SVG fonts that accompany Web pages must be supported only in browsing and viewing situations. Graphics editing applications or file translation tools must not attempt to convert SVG fonts into system fonts. The intent is that SVG files be interchangeable between two content creators, but not the SVG fonts that might accompany these SVG files. Instead, each content creator will need to license the given font before being able to successfully edit the SVG file. The **font-face-name** element indicates the name of licensed font to use for editing.

SVG fonts contain unhinted font outlines. Because of this, on many implementations there will be limitations regarding the quality and legibility of text in small font sizes. For increased quality and legibility in small font sizes, content creators may want to use an alternate font technology, such as fonts that ship with operating systems or an alternate [WebFont](#) format.

Because SVG fonts are expressed using SVG elements and attributes, in some cases the SVG font will take up more space than if the font were expressed in a different 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 as a first choice, with a fallback to an SVG font for interoperability.

A key value of SVG fonts is guaranteed availability in SVG user agents. In some situations, it might be appropriate for an SVG font to be the first choice for rendering some text. In other situations, the SVG font might be an alternate, back-up font in case the first choice font (perhaps a hinted system font) is not available to a given user.

The characteristics and attributes of SVG fonts correspond closely to the font characteristics and parameters described in the ["Fonts"](#) chapter of the ["Cascading Style Sheets \(CSS\) level 2"](#) specification [CSS2]. In this model, various font metrics, such as advance values and baseline locations, and the glyph outlines themselves, are expressed in units that are relative to an abstract square whose height is the intended distance between lines of type in the same type size. This square is called the **em square** and it is the design grid on which the glyph outlines are defined. The value of the **units-per-em** attribute on the **'font'** element specifies how many units the em square is divided into. Common values for other font types are, for example, 250 (Intellifont), 1000 (Type 1) and 2048 (TrueType, TrueType GX and Open-Type). Unlike standard graphics in SVG, where the initial coordinate system has the y-axis pointing downward (see [The initial coordinate system](#)), the design grid for SVG fonts, along with the initial coordinate system for the glyphs, has the y-axis pointing upward for consistency with accepted industry practice for many popular font formats.

SVG fonts and their associated glyphs do not specify bounding box information. Because the glyph outlines are expressed as SVG path elements, the implementation has the option to render the glyphs either using standard graphics calls or by using special-purpose font rendering technology, in which case any necessary maximum bounding box and overhang calculations can be performed from analysis of the path elements contained within the glyph outlines.

An SVG font can be either embedded within the same document that uses the font or saved as part of an external resource.

Here is an example of how you might embed an SVG font inside of an SVG document.

Example: 20_01.svg

```
<?xml version="1.0"?>
<svg width="400px" height="300px" version="1.2" baseProfile="tiny"
  xmlns = 'http://www.w3.org/2000/svg'>
  <defs>
    <font xml:id="Font1" horiz-adv-x="1000">
      <font-face font-family="Super Sans" font-weight="bold" font-style="normal"
        units-per-em="1000" cap-height="600" x-height="400"
        ascent="700" descent="300" alphabetic="0" mathematical="350" ideographic="400" hanging="500">
        <font-face-src>
          <font-face-name name="Super Sans Bold"/>
        </font-face-src>
      </font-face>
      <missing-glyph d="M0,0h200v200h-200z"/>
      <glyph unicode="!" horiz-adv-x="300" d="--Outline of exclam. pt. glyph--"/>
      <glyph unicode="@" d="--Outline of @ glyph--"/>
      <!-- more glyphs -->
    </font>
  </defs>
  <desc>An example using an embedded font.</desc>
  <text x="100" y="100" font-family="Super Sans, Helvetica, sans-serif"
    font-weight="bold" font-style="normal">Text
    using embedded font</text>
</svg>
```

17.3 The 'font' element

The **'font'** element defines an SVG font.

Schema: font

```
<define name='font'>
```

```

    <element name='font'>
      <ref name='font.AT' />
      <ref name='font.CM' />
    </element>
  </define>

  <define name='font.AT' combine='interleave'>
    <ref name='svg.Core.attr' />
    <ref name='svg.External.attr' />
    <ref name='svg.FontAdvOrigCommon.attr' />
    <optional>
      <attribute name='horiz-origin-x' svg:animatable='false' svg:inheritable='false'>
        <ref name='Number.datatype' />
      </attribute>
    </optional>
  </define>

  <define name='font.CM'>
    <zeroOrMore>
      <choice>
        <ref name='svg.Desc.group' />
        <ref name='font-face' />
        <ref name='missing-glyph' />
        <ref name='glyph' />
        <ref name='hkern' />
      </choice>
    </zeroOrMore>
  </define>

  <define name='svg.FontAdvOrigCommon.attr' combine='interleave'>
    <optional>
      <attribute name='horiz-adv-x' svg:animatable='false' svg:inheritable='false'>
        <ref name='Number.datatype' />
      </attribute>
    </optional>
  </define>

```

Attribute definitions:

horiz-origin-x = "<number>"

The X-coordinate in the font coordinate system of the origin of a glyph to be used when drawing horizontally oriented text. (Note that the origin applies to all glyphs in the font.)

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: no.

horiz-adv-x = "<number>"

The default horizontal advance after rendering a glyph in horizontal orientation. Glyph widths are required to be non-negative, even if the glyph is typically rendered right-to-left, as in Hebrew and Arabic scripts. If the attribute is not specified, the effect is as if a value of "0" was specified.

Animatable: no.

Each **'font'** element must have a **'font-face'** child element which describes various characteristics of the font.

17.4 The 'glyph' element

The **'glyph'** element defines the graphics for a given glyph. The coordinate system for the glyph is defined by the various attributes in the **'font'** element.

The graphics that make up the **'glyph'** consist of a single [path data](#) specification within the **d** attribute.

Schema: glyph

```

<define name='glyph'>
  <element name='glyph'>
    <ref name='glyph.AT' />
    <ref name='glyph.CM' />
  </element>
</define>

<define name='glyph.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.FontAdvOrigCommon.attr' />
  <ref name='svg.D.attr' />
  <attribute name='unicode' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  <optional><attribute name='glyph-name' svg:animatable='false' svg:inheritable='false'><text/></attribute>
</optional>

```

```

<optional><attribute name='arabic-form' svg:animatable='false' svg:inheritable='false'><text/></attribute>
</optional>
<optional>
  <attribute name='lang' svg:animatable='false' svg:inheritable='false'>
    <ref name='LanguageCodes.datatype' />
  </attribute>
</optional>
</define>

<define name='glyph.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

unicode = "<string>"

One or more Unicode characters indicating the sequence of Unicode characters which corresponds to this glyph. If a character is provided, then this glyph corresponds to the given Unicode character. If multiple characters are provided, then this glyph corresponds to the given sequence of Unicode characters. One use of a sequence of characters is ligatures. For example, if **unicode="ffi"**, then the given glyph will be used to render the sequence of characters "f", "i", and "i".

It is often useful to refer to characters using XML character references expressed in hexadecimal notation or decimal notation. For example, **unicode="ffi"** could be expressed as XML character references in hexadecimal notation as **unicode="ffl"** or in decimal notation as **unicode="ffl"**.

The **unicode** attribute contributes to the process for deciding which glyph(s) are used to represent which character(s). See [glyph selection rules](#). If the **unicode** attribute is not provided for a given **'glyph'**, the glyph cannot be accessed in SVG Tiny 1.2.

Animatable: no.

glyph-name = "<name> [, <name>]* "

An optional name for the glyph. Glyph names should be unique within a font. The glyph names can be used in situations where Unicode character numbers do not provide sufficient information to access the correct glyph, such as when there are multiple glyphs per Unicode character. The glyph names can be referenced in [kerning](#) definitions.

Animatable: no.

d = "path data"

The definition of the outline of a glyph, using the same syntax as for the **d** attribute on a **'path'** element. See [Path data](#).

See below for a discussion of this attribute.

Animatable: no.

arabic-form = "initial | medial | terminal | isolated"

For Arabic glyphs, indicates which of the four possible forms this glyph represents. If **arabic-form** is not specified for a glyph that requires it, the glyph is taken to be the isolated form and the initial, medial, and terminal forms will render with missing-glyph unless separately specified.

Additionally, if initial, medial, or terminal are specified on a glyph that does not require the **arabic-form** to be specified, the **arabic-form** attribute shall have no effect.

Animatable: no.

lang = "%LanguageCodes;"

The attribute value is a comma-separated list of language tags as defined in IETF Best Current Practice 47 [\[BCP 47\]](#). The glyph can be used if the **xml:lang** attribute exactly matches one of the languages given in the value of this parameter, or if the **xml:lang** attribute exactly equals a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-". If the attribute is not specified, then the glyph can be used in all languages.

Animatable: no.

horiz-adv-x = "<number>"

The horizontal advance after rendering the glyph in horizontal orientation. If the attribute is not specified, the effect is as if the attribute were set to the value of the font's [horiz-adv-x](#) attribute.

Glyph widths are required to be non-negative, even if the glyph is typically rendered right-to-left, as in Hebrew and Arabic scripts.

Animatable: no.

The graphics for the **'glyph'** are specified using the **d** attribute. The path data within this attribute must be processed as follows:

- Any relative coordinates within the path data specification are converted into equivalent absolute coordinates
- Each of these absolute coordinates is transformed from the font coordinate system into the **'text'** element's current coordinate system such that the origin of the font coordinate system is properly positioned and rotated to align with the [current text position](#) and orientation for the glyph, and scaled so that the correct **'font-size'** is achieved.
- The resulting, transformed path specification is rendered as if it were a **'path'** element, using the styling properties that apply to the

characters which correspond to the given glyph, and ignoring any styling properties specified on the **'font'** element or the **'glyph'** element.

In general, the **d** attribute renders in the same manner as system fonts. For example, a dashed pattern will usually look the same if applied to a system font or to an SVG font which defines its glyphs using the **d** attribute. Many implementations will be able to render glyphs quickly and will be able to use a font cache for further performance gains.

Example font01 below contains a font for just three letters - S, V, and G - plus a missing glyph for all other characters. There is also a kern pair, to bring the V and the G glyphs closer together. The font, Anglepoise, was designed by Ray Larabie of [Larabie Fonts](http://www.larabiefonts.com/) and is used by permission.

Example: font01.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" viewBox="0 0 160 70" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Font example</title>
  <defs>
    <font horiz-adv-x="313" id="la">
      <metadata>Converted from Larabie Anglepoise by Batik ttf2svg
      See http://www.larabiefonts.com/ </metadata>
      <font-face font-family="larabie-anglepoise" units-per-em="1000"
        panose-1="0 0 4 0 0 0 0 0 0" ascent="703" descent="-300" alphabetic="0"/>
      <missing-glyph horiz-adv-x="500" d="M63 0V700H438V0H63ZM125 63H375V638H125V63Z"/>
      <glyph unicode="S" glyph-name="S" horiz-adv-x="385" d="M371 1H29V144H264Q264 151 264
        166Q265 180 265 188Q265 212 249 212H132Q83 212 55 247Q29 279 29
        329V566H335V422H136V375Q136 360 144 356Q148 355 168 355H279Q327 355 352 309Q371 273
        371 221V1Z"/>
      <glyph unicode="V" glyph-name="V" horiz-adv-x="351" d="M365 563L183 -33L0 563H101L183
        296L270 563H365Z"/>
      <glyph unicode="G" glyph-name="G" horiz-adv-x="367" d="M355
        1H18V564H355V420H125V144H248V211H156V355H355V1Z"/>
      <hkern g1="V" g2="G" k="-40"/>
    </font>
  </defs>
  <text x="40" y="50" font-family="larabie-anglepoise" font-size="70" fill="#933">SVG</text>
  <rect x="00" y="00" width="160" height="70" stroke="#777" fill="none"/>
</svg>
```



Example font02 below contains a font for a single character, the Arabic letter • - plus a space and missing glyph. There are four glyphs for this character, each corresponding to the same Unicode code point U+062E. They are distinguished by the values of the arabic-form attribute. The text string demonstrates each of the four forms.

Example: font02.svg

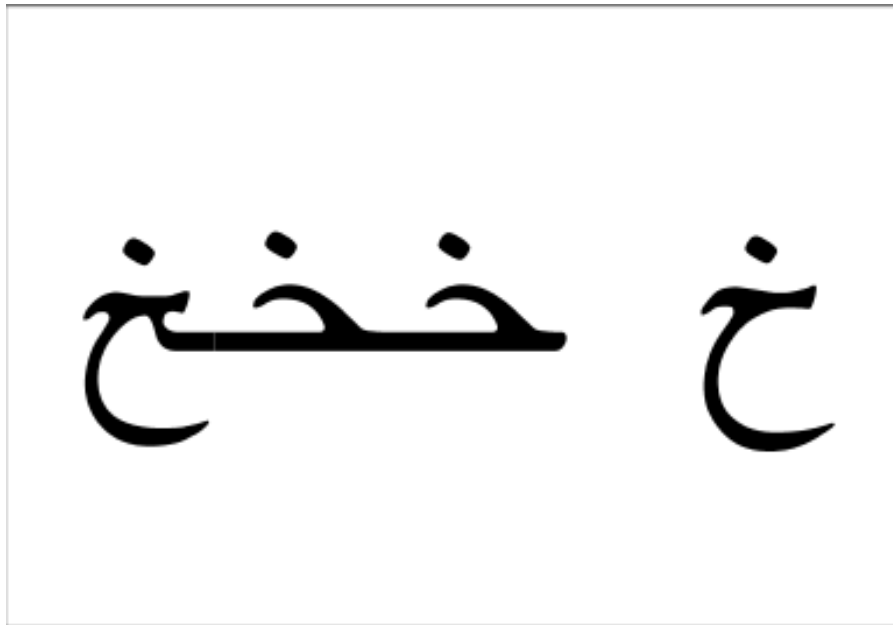
```
<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" viewBox="80 100 260 180" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Font example for arabic-form</title>
  <defs>
    <font horiz-adv-x="573">
      <font-face font-family="SVGAr" units-per-em="1000" panose-1="5 1 1 1 1 1 1 1 1"
        ascent="1025" descent="-399" alphabetic="0"/>
      <missing-glyph horiz-adv-x="500" d="M31 0V800H469V0H31ZM438 31V769H62V31H438Z"/>
      <glyph unicode=" " glyph-name="space" horiz-adv-x="370"/>
      <glyph unicode="&#x62e;" glyph-name="khah-isolated" arabic-form="isolated"
        horiz-adv-x="562" d="M309 360Q309 353 297 335T271 317Q260 317 227 337T194 370Q194
        380 205 397T232 415Q246 415 277 395T309 360ZM518 -265Q516 -269 509 -275Q507 -277 502
        -281Q447 -319 424 -330Q356 -363 281 -363Q228 -363 186 -347T110 -294Q69 -249 54
        -199Q44 -167 44 -127Q44 -96 50 -65T76 12Q88 39 110 70Q152 127 152 137Q152 151 148
        156T121 161Q95 161 85 156Q72 146 62 140Q44 128 35 130Q35 138 35 146Q37 151 43 162Q77
        208 98 219T159 231Q170 231 234 221Q255 218 298 210H340Q347 210 382 218T425 230T435
        235Q446 239 449 234Q454 226 444 189T426 152Q418 152 393 154T360 156Q327 156 297
        149T228 120Q180 93 142 36Q96 -33 96 -110Q96 -209 168 -257Q223 -294 300 -294Q343 -294
```



```

371 -291Q429 -285 489 -267Q506 -260 511 -260Q514 -262 518 -265Z"/>
<glyph unicode="#x62e;" glyph-name="khah-initial" arabic-form="initial"
horiz-adv-x="728" d="M297 372Q297 365 285 347T259 329Q248 329 215 349T182 382Q182
392 193 409T220 427Q234 427 265 407T297 372ZM600 0H0V68H373Q396 68 396 86Q396 89 394
95Q377 137 347 159Q308 188 243 188Q210 188 183 171Q165 160 157 158T145 156Q138 156
138 164L140 174Q145 196 191 220Q228 240 269 240Q313 240 355 221T447 160Q488 120 530
81Q543 73 563 71T609 68Q619 68 620 68T625 68Q645 68 645 46Q645 30 633 15T600 0Z"/>
<glyph unicode="#x62e;" glyph-name="khah-medial" arabic-form="medial"
horiz-adv-x="625" d="M296 376Q296 369 284 351T258 333Q247 333 214 353T181 386Q181
396 192 413T219 431Q233 431 264 411T296 376ZM625 0H0V68H373Q396 68 396 86Q396 89 394
95Q377 137 347 159Q308 188 243 188Q210 188 183 171Q165 160 157 158T145 156Q138 156
138 164L140 174Q145 196 191 220Q228 240 269 240Q313 240 355 221T447 160Q488 120 530
81Q543 73 563 71T609 68Q619 68 620 68T625 68V0Z"/>
<glyph unicode="#x62e;" glyph-name="khah-terminal" arabic-form="terminal"
horiz-adv-x="514" d="M296 352Q296 345 284 327T258 309Q247 309 214 329T181 362Q181
372 192 389T219 407Q233 407 264 387T296 352ZM514 0H375Q313 0 298 64T261 128Q209 128
153 62Q91 -12 91 -101Q91 -199 162 -243Q220 -279 319 -279Q367 -279 390 -276T463
-259Q466 -258 475 -255T488 -252Q490 -252 491 -254T489 -263Q484 -272 466 -286T433
-307Q408 -320 401 -323Q349 -344 277 -344Q169 -344 104 -274Q44 -210 44 -118Q44 -88 51
-53T73 14Q80 31 97 56Q132 108 132 118Q132 127 126 134T110 141Q92 141 85 137Q72 127
59 117Q49 112 44 112Q38 112 38 119Q38 122 40 128Q49 156 80 182Q116 212 157 212Q170
212 188 208Q232 198 258 198H320Q345 198 357 201Q374 207 383 209T398 214T412 216Q420
216 421 212Q424 202 414 170T396 137Q394 137 382 140T362 143Q346 143 337 135T327
104Q327 91 341 80T379 68H514V0Z"/>
</font>
</defs>
<g font-family="SVGAr" font-size="80">
  <!-- this should produce isolated khah, followed by initial,medial and terminal khah -->
  <text x="100" y="200">&#x62e; &#x62e;&#x62e;&#x62e;</text>
  <rect x="80" y="100" width="260" height="180" fill="none" stroke="#777"/>
</g>
</svg>

```



17.5 The 'missing-glyph' element

The **'missing-glyph'** element defines a graphic to use if there is an attempt to draw a glyph from a given font and the given glyph has not been defined. The attributes on the **'missing-glyph'** element have the same meaning as the corresponding attributes on the **'glyph'** element.

Schema: missing-glyph

```

<define name='missing-glyph'>
  <element name='missing-glyph'>
    <ref name='missing-glyph.AT' />
    <ref name='missing-glyph.CM' />
  </element>
</define>

<define name='missing-glyph.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.FontAdvOrigCommon.attr' />
  <ref name='svg.D.attr' />
</define>

<define name='missing-glyph.CM'>
  <zeroOrMore>
    <choice>

```

```

    <ref name='svg.Desc.group' />
  </choice>
</zeroOrMore>
</define>

```

17.6 Glyph selection rules

When determining the glyph(s) to draw a given character sequence, the **'font'** element must be searched from its first **'glyph'** element to its last in logical order to see if the upcoming sequence of Unicode characters to be rendered matches the sequence of Unicode characters specified in the **unicode** attribute for the given **'glyph'** element. The first successful match must be used. Thus, if an "ffi" ligature is defined in the font after the "f" glyph, it will not be used.

17.7 The **'hkern'** element

The **'hkern'** element defines kerning pairs for horizontally-oriented pairs of glyphs.

Kern pairs identify pairs of glyphs within a single font whose inter-glyph spacing is adjusted when the pair of glyphs are rendered next to each other. In addition to the requirement that the pair of glyphs are from the same font, SVG font kerning happens only when the two glyphs correspond to characters which have the same values for properties **'font-family'**, **'font-size'**, **'font-style'** and **'font-weight'**.

An example of a kerning pair are the letters "Va", where the typographic result might look better if the letters "V" and the "a" were rendered slightly closer together.

Right-to-left and bidirectional text in SVG is laid out in a two-step process, which is described in [Relationship with bidirectionality](#). If SVG fonts are used, before kerning is applied, characters are re-ordered into left-to-right (or top-to-bottom, for vertical text) visual rendering order. Kerning from SVG fonts is then applied on pairs of glyphs which are rendered contiguously. The first glyph in the kerning pair is the left (or top) glyph in visual rendering order. The second glyph in the kerning pair is the right (or bottom) glyph in the pair.

For convenience to font designers and to minimize file sizes, a single **'hkern'** can define a single kerning adjustment value between one set of glyphs (e.g., a range of Unicode characters) and another set of glyphs (e.g., another range of Unicode characters).

The **'hkern'** element defines kerning pairs and adjustment values in the horizontal advance value when drawing pairs of glyphs which the two glyphs are contiguous and are both rendered horizontally (i.e., side-by-side). The spacing between characters is reduced by the kerning adjustment. (Negative kerning adjustments increase the spacing between characters.)

Schema: hkern

```

<define name='hkern'>
  <element name='hkern'>
    <ref name='hkern.AT' />
    <ref name='hkern.CM' />
  </element>
</define>

<define name='hkern.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <optional>
    <attribute name='u1' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>
  <optional>
    <attribute name='g1' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>
  <optional>
    <attribute name='u2' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>
  <optional>
    <attribute name='g2' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>
  <attribute name='k' svg:animatable='false' svg:inheritable='false'>
    <ref name='Number.datatype' />
  </attribute>
</define>

<define name='hkern.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

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

A sequence (comma-separated) of Unicode characters (refer to the description of the [unicode](#) attribute to the ['glyph'](#) element for a description of how to express individual Unicode characters) and/or ranges of Unicode characters (see description of ranges of Unicode characters in [\[CSS2\]](#)) which identify a set of possible first glyphs in the kerning pair. If a given Unicode character within the set has multiple corresponding ['glyph'](#) elements (i.e., there are multiple ['glyph'](#) elements with the same [unicode](#) attribute value, but different [glyph-name](#) values), then all such glyphs are included in the set. Comma is the separator character; thus, to kern a comma, specify the comma as part of a range of Unicode characters or as a glyph name using the [g1](#) attribute. The total set of possible first glyphs in the kerning pair is the union of glyphs specified by the [u1](#) and [g1](#) attributes.

Animatable: no.

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

A sequence (comma-separated) of glyph names (i.e., values that match [glyph-name](#) attributes on ['glyph'](#) elements) which identify a set of possible first glyphs in the kerning pair. All glyphs with the given glyph name are included in the set. The total set of possible first glyphs in the kerning pair is the union of glyphs specified by the [u1](#) and [g1](#) attributes.

Animatable: no.

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

Same as the [u1](#) attribute, except that [u2](#) specifies possible second glyphs in the kerning pair.

Animatable: no.

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

Same as the [g1](#) attribute, except that [g2](#) specifies possible second glyphs in the kerning pair.

Animatable: no.

k = "<number>"

The amount to decrease the spacing between the two glyphs in the kerning pair. The value is in the font coordinate system. If the attribute is not specified, the effect is as if a value of "0" was specified.

Animatable: no.

At least one each of [u1](#) or [g1](#) and at least one of [u2](#) or [g2](#) must be provided.

17.8 Describing a font

17.8.1 Overview of font descriptions

A font description provides the bridge between an author's font specification and the font data, which is the data needed to format text and to render the abstract glyphs to which the characters map - the actual scalable outlines or bitmaps. Fonts are referenced by properties, such as the ['font-family'](#) property.

Each specified font description is added to the font database and so that it can be used to select the relevant font data. The font description contains descriptors such as the location of the font data on the Web, and characterizations of that font data. The font descriptors are also needed to match the font properties to particular font data. The level of detail of a font description can vary from just the name of the font up to a list of glyph widths.

For more about font descriptions, refer to the font chapter in the CSS2 specification [\[CSS2 Fonts\]](#).

17.8.2 The ['font-face'](#) element

The ['font-face'](#) element is an XML structure which corresponds directly to the [@font-face](#) facility in CSS2. It can be used to describe the characteristics of any font, SVG font or otherwise.

When used to describe the characteristics of an SVG font contained within the same document, it is recommended that the ['font-face'](#) element be a child of the ['font'](#) element it is describing so that the ['font'](#) element can be self-contained and fully-described. In this case, any ['font-face-src'](#) elements within the ['font-face'](#) element are ignored as it is assumed that the ['font-face'](#) element is describing the characteristics of its parent ['font'](#) element.

Schema: font-face

```
<define name='font-face'>
  <element name='font-face'>
    <ref name='font-face.AT' />
    <ref name='font-face.CM' />
  </element>
</define>

<define name='font-face.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.External.attr' />
  <optional><attribute name='font-family' svg:animatable='false' svg:inheritable='false'><text/></attribute>
```



```

    </attribute>
  </optional>
<optional>
  <attribute name='strikethrough-position' svg:animatable='false' svg:inheritable='false'>
    <ref name='Number.datatype' />
  </attribute>
</optional>
<optional>
  <attribute name='strikethrough-thickness' svg:animatable='false' svg:inheritable='false'>
    <ref name='Number.datatype' />
  </attribute>
</optional>
<optional>
  <attribute name='overline-position' svg:animatable='false' svg:inheritable='false'>
    <ref name='Number.datatype' />
  </attribute>
</optional>
<optional>
  <attribute name='overline-thickness' svg:animatable='false' svg:inheritable='false'>
    <ref name='Number.datatype' />
  </attribute>
</optional>
</define>

<define name='font-face.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='font-face-src' />
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

font-family = "<string>"

Same syntax and semantics as the '[font-family](#)' descriptor within an [@font-face](#) rule.

Animatable: no.

font-style = "all | [normal | italic | oblique] [, [normal | italic | oblique]]*"

Same syntax and semantics as the '[font-style](#)' descriptor within an [@font-face](#) rule. The style of a font. Takes on the same values as the '[font-style](#)' property, except that a comma-separated list is permitted.

If the attribute is not specified, the effect is as if a value of "all" were specified.

Animatable: no.

font-weight = "all | [normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900] [, [normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900]]*"

Same syntax and semantics as the '[font-weight](#)' descriptor within an [@font-face](#) rule.

The weight of a face relative to others in the same font family. Takes on the same values as the '[font-weight](#)' property with three exceptions:

- o relative keywords (bolder, lighter) must not be used
- o a comma-separated list of values may be used, for fonts that contain multiple weights
- o an additional keyword, 'all', may be used. 'all' indicates that the font will match for all possible weights; either because it contains multiple weights, or because that face only has a single weight.

If the attribute is not specified, the effect is as if a value of "all" were specified.

Animatable: no.

unicode-range = "<urange> [, <urange>]"

Same syntax and semantics as the '[unicode-range](#)' descriptor within an [@font-face](#) rule. The range of ISO 10646 characters [UNICODE] possibly covered by the glyphs in the font. Except for any additional information provided in this specification, the normative definition of the attribute is in [CSS2].

If the attribute is not specified, the effect is as if a value of "U+0-10FFFF" were specified.

Animatable: no.

units-per-em = "<number>"

Same syntax and semantics as the '[units-per-em](#)' descriptor within an [@font-face](#) rule. The number of coordinate units on the em square, the size of the design grid on which glyphs are laid out.

This value should be specified as nearly every other attribute requires the definition of a design grid.

If the attribute is not specified, the effect is as if a value of "1000" were specified.

Animatable: no.

panose-1 = "[<integer>]{10}"

Same syntax and semantics as the '[panose-1](#)' descriptor within an [@font-face](#) rule. The Panose-1 number, consisting of ten decimal integers, separated by whitespace. Except for any additional information provided in this specification, the normative definition of the attribute is in [CSS2].

If the attribute is not specified, the effect is as if a value of "0 0 0 0 0 0 0 0 0 0" were specified.

Animatable: no.

stemv = "<number>"

Same syntax and semantics as the '**stemv**' descriptor within an [@font-face](#) rule.

Animatable: no.

stemh = "<number>"

Same syntax and semantics as the '**stemh**' descriptor within an [@font-face](#) rule.

Animatable: no.

slope = "<number>"

Same syntax and semantics as the '**slope**' descriptor within an [@font-face](#) rule. The vertical stroke angle of the font. Except for any additional information provided in this specification, the normative definition of the attribute is in [\[CSS2\]](#).

If the attribute is not specified, the effect is as if a value of "0" were specified.

Animatable: no.

cap-height = "<number>"

Same syntax and semantics as the '**cap-height**' descriptor within an [@font-face](#) rule. The height of uppercase glyphs in the font within the font coordinate system.

Animatable: no.

x-height = "<number>"

Same syntax and semantics as the '**x-height**' descriptor within an [@font-face](#) rule. The height of lowercase glyphs in the font within the font coordinate system.

Animatable: no.

accent-height = "<number>"

The distance from the origin to the top of accent characters, measured by a distance within the font coordinate system.

If the attribute is not specified, the effect is as if the attribute were set to the value of the [ascent](#) attribute. If this attribute is used, the [units-per-em](#) attribute must also be specified.

Animatable: no.

ascent = "<number>"

Same syntax and semantics as the '**ascent**' descriptor within an [@font-face](#) rule. The maximum unaccented height of the font within the font coordinate system.

If the attribute is not specified, the effect is as if the attribute were set to the difference between the [units-per-em](#) value and the [vert-origin-y](#) value for the corresponding font.

Animatable: no.

descent = "<number>"

Same syntax and semantics as the '**descent**' descriptor within an [@font-face](#) rule. The maximum unaccented depth of the font within the font coordinate system.

If the attribute is not specified, the effect is as if the attribute were set to the [vert-origin-y](#) value for the corresponding font.

Animatable: no.

widths = "<string>"

Same syntax and semantics as the '**widths**' descriptor within an [@font-face](#) rule.

Animatable: no.

bbox = "<string>"

Same syntax and semantics as the '**bbox**' descriptor within an [@font-face](#) rule.

Animatable: no.

ideographic = "<number>"

For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve ideographic baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

alphabetic = "<number>"

Same syntax and semantics as the '**baseline**' descriptor within an [@font-face](#) rule. For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve alphabetic baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

mathematical = "<number>"

Same syntax and semantics as the '**mathline**' descriptor within an [@font-face](#) rule. For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve mathematical baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

hanging = "<number>"

For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve hanging baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

underline-position = "<number>"

The ideal position of an underline within the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

underline-thickness = "<number>"

The ideal thickness of an underline, expressed as a length within the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

strikethrough-position = "<number>"

The ideal position of a strike-through within the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

strikethrough-thickness = "<number>"

The ideal thickness of a strike-through, expressed as a length within the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

overline-position = "<number>"

The ideal position of an overline within the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

overline-thickness = "<number>"

The ideal thickness of an overline, expressed as a length within the font coordinate system. If this attribute is provided, the [units-per-em](#) attribute must also be specified.

Animatable: no.

17.8.3 The 'font-face-src' element

The 'font-face-src' element, together with the 'font-face-uri' and 'font-face-name' elements described further down correspond to the 'src' descriptor within an @font-face rule. (Refer to the descriptions of the [[@font-face rule](#)] and [['src' descriptor](#)] in the CSS2 specification.)

A 'font-face-src' element contains either a 'font-face-uri' or a 'font-face-name' element. The 'font-face-src' element is used for referencing fonts defined elsewhere.

Schema: font-face-src

```
<define name='font-face-src'>
  <element name='font-face-src'>
    <ref name='font-face-src.AT' />
    <ref name='font-face-src.CM' />
  </element>
</define>

<define name='font-face-src.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
</define>

<define name='font-face-src.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
      <ref name='font-face-uri' />
      <ref name='font-face-name' />
    </choice>
  </zeroOrMore>
</define>
```

17.8.4 The 'font-face-uri' element

The 'font-face-uri' element is used within a 'font-face-src' element to reference a font defined inside or outside of the current SVG document.

When a 'font-face-uri' is referencing an [SVG font](#), then that reference must be to an SVG 'font' element, therefore requiring the use of a fragment identifier (see [[IRI](#)]). The referenced 'font' element can be local (i.e., within the same document as the 'font-face-uri' element) or remote (i.e., within a different document).

Schema: font-face-uri

```

<define name='font-face-uri'>
  <element name='font-face-uri'>
    <ref name='font-face-uri.AT' />
    <ref name='font-face-uri.CM' />
  </element>
</define>

<define name='font-face-uri.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.XLinkRequired.attr' />
</define>

<define name='font-face-uri.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

xlink:href = " <iri>"
 An [IRI reference](#)
Animatable: no.

17.8.5 The 'font-face-name' element

The 'font-face-name' element is used within a 'font-face-src' element to reference locally installed fonts by name.

Schema: font-face-name

```

<define name='font-face-name'>
  <element name='font-face-name'>
    <ref name='font-face-name.AT' />
    <ref name='font-face-name.CM' />
  </element>
</define>

<define name='font-face-name.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <optional>
    <attribute name='name' svg:animatable='false' svg:inheritable='false'><text/></attribute>
  </optional>
</define>

<define name='font-face-name.CM'>
  <zeroOrMore>
    <choice>
      <ref name='svg.Desc.group' />
    </choice>
  </zeroOrMore>
</define>

```

Attribute definitions:

name = " <string>"
 The name of the font to use.
Animatable: no.

Example font03 references an external SVG font, which was defined in example font01.svg.

Example: font03.svg

```

<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" viewBox="0 0 110 70" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Font example</title>
  <defs>
    <font-face font-family="larabie-anglepoise">
      <font-face-src>
        <font-face-uri xlink:href="font01.svg#la" />
      </font-face-src>

```



```
</font-face>
</defs>
<rect x="00" y="00" width="110" height="70" stroke="#777" fill="none"/>
<text x="10" y="50" font-family="larabie-anglepoise" font-size="70" fill="#FDD" stroke="#533"
stroke-width="1.6">SVG</text>
</svg>
```



17.9 Font Module

The Font Module contains the following elements:

Font

- [font](#)
- [glyph](#)
- [font-face](#)
- [missing-glyph](#)
- [hkern](#)
- [font-face-src](#)
- [font-face-uri](#)
- [font-face-name](#)

[\[RNG\]](#) [\[Feature String\]](#)

18 Metadata

Contents

- 18.1 [Introduction](#)
- 18.2 [The 'metadata' element](#)
- 18.3 [An example](#)

18.1 Introduction

Metadata is structured data about data.

In the computing industry, there are ongoing standardization efforts towards metadata with the goal of promoting industry interoperability and efficiency. Content creators should track these developments and include appropriate metadata in their SVG content which conforms to these various metadata standards as they emerge.

The W3C has a [Semantic Web Activity](#) which has been established to serve a leadership role, in both the design of enabling specifications and the open, collaborative development of technologies that support the automation, integration and reuse of data across various applications. The Semantic Web Activity builds upon the earlier W3C Metadata Activity, including the definition of Resource Description Framework (RDF). The set of six specifications for RDF can be found starting [Resource Description Framework Primer \[RDF\]](#)

Another activity relevant to most applications of metadata is the [Dublin Core \[DCORE\]](#), which is a set of generally applicable core metadata properties (e.g., Title, Creator/Author, Subject, Description, etc.).

Individual industries or individual content creators are free to define their own metadata schema but are encouraged to follow existing metadata standards and use standard metadata schema wherever possible to promote interchange and interoperability. If a particular standard metadata schema does not meet your needs, then it is usually better to define an additional metadata schema in an existing framework such as RDF and to use custom metadata schema in combination with standard metadata schema, rather than totally ignore the standard schema.

18.2 The 'metadata' element

Metadata which is included with SVG content should be specified within **'metadata'** elements. The contents of the **'metadata'** should be elements from other XML namespaces, with these elements from these namespaces expressed in a manner conforming with the "Namespaces in XML 1.1" Recommendation [\[XML-NS\]](#).

Authors should provide a **'metadata'** child element to the **'svg'** element within a stand-alone SVG document. The **'metadata'** child element to an **'svg'** element serves the purposes of identifying document-level metadata.

Schema: metadata

```
<define name='metadata'>
  <element name='metadata'>
    <ref name='DTM.AT' />
    <ref name='DTM.CM' />
  </element>
</define>
```

18.3 An example

Here is an example of how metadata can be included in an SVG document. The example uses the Dublin Core version 1.1 schema. (Other XML-compatible metadata languages, including ones not based on RDF, can be used also.)

Example: 21_01.svg

```
<?xml version="1.0"?>
<svg width="4in" height="3in" version="1.2" baseProfile="tiny"
  xmlns = 'http://www.w3.org/2000/svg'>
  <desc xmlns:myfoo="http://example.org/myfoo">
    <myfoo:title>This is a financial report</myfoo:title>
    <myfoo:descr>The global description uses markup from the
      <myfoo:emph>myfoo</myfoo:emph> namespace.</myfoo:descr>
    <myfoo:scene><myfoo:what>widget $growth</myfoo:what>
    <myfoo:contains>$three $graph-bar</myfoo:contains>
    <myfoo:when>1998 $through 2000</myfoo:when> </myfoo:scene>
  </desc>
  <metadata>
    <rdf:RDF
      xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
      xmlns:dc = "http://purl.org/dc/elements/1.1/" >
      <rdf:Description about="http://example.org/myfoo"
        dc:title="MyFoo Financial Report"
        dc:description="$three $bar $thousands $dollars $from 1998 $through 2000"
        dc:publisher="Example Organization"
        dc:date="2000-04-11"
        dc:format="image/svg+xml"
        dc:language="en" >
        <dc:creator>
          <rdf:Bag>
            <rdf:li>Irving Bird</rdf:li>
            <rdf:li>Mary Lambert</rdf:li>
          </rdf:Bag>
        </dc:creator>
      </rdf:Description>
    </rdf:RDF>
  </metadata>
</svg>
```

19 Extensibility

Contents

- 19.1 [Foreign namespaces and private data](#)
- 19.2 [Embedding foreign object types](#)
 - 19.2.1 [The 'foreignObject' element](#)
- 19.3 [An example](#)
- 19.4 [Extensibility Module](#)

19.1 Foreign namespaces and private data

SVG allows inclusion of elements from foreign namespaces anywhere with the SVG content. In general, the SVG user agent will include the unknown elements in the DOM but will otherwise ignore unknown elements. (The notable exception is described under [Embedding Foreign Object Types](#).)

Extension elements in the SVG namespace must not be used.

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 will otherwise ignore unknown attributes. Unprefixed attributes on elements in the SVG namespace must not be used for extensions.

SVG's ability to include foreign namespaces can be used for the following purposes:

- Application-specific information so that authoring applications can include model-level data in the SVG content to serve their "roundtripping" purposes (i.e., the ability to write, then read a file without loss of higher-level information).
- Supplemental data for extensibility. For example, suppose you have an extrusion extension which takes any 2D graphics and extrudes it in three dimensions. When applying the extrusion extension, you probably will need to set some parameters. The parameters can be included in the SVG content by inserting elements from an extrusion extension namespace.

To illustrate, a business graphics authoring application might want to include some private data within an SVG document so that it could properly reassemble the chart (a pie chart in this case) upon reading it back in:

Example: 23_01.svg

```
<?xml version="1.0"?>
<svg width="4in" height="3in" version="1.2" baseProfile="tiny"
  xmlns = 'http://www.w3.org/2000/svg'>
  <defs>
    <myapp:piechart xmlns:myapp="http://example.org/myapp"
      title="Sales by Region">
      <myapp:pieslice label="Northern Region" value="1.23"/>
      <myapp:pieslice label="Eastern Region" value="2.53"/>
      <myapp:pieslice label="Southern Region" value="3.89"/>
      <myapp:pieslice label="Western Region" value="2.04"/>
      <!-- Other private data goes here -->
    </myapp:piechart>
  </defs>
  <desc>This chart includes private data in another namespace
  </desc>
  <!-- In here would be the actual SVG graphics elements which
    draw the pie chart -->
</svg>
```

19.2 Embedding foreign object types

One goal for SVG is to provide a mechanism by which other XML language processors can render into an area within an SVG drawing, with those renderings subject to the various transformations and compositing parameters that are currently active at a given point within the SVG content tree. One particular example of this is to provide a frame for XML content styled with CSS or XSL so that dynamically reflowing text (subject to SVG transformations and compositing) could be inserted into the middle of some SVG content. Another example is inserting a [MathML](#) expression into an SVG drawing.

The **'foreignObject'** element allows for inclusion of a foreign namespace which has its graphical content drawn by a different user agent. The included foreign graphical content is subject to SVG transformations and compositing.

The user agent must treat all of the content within a **'foreignObject'** as foreign content which is to be handed off to an appropriate content handler for rendering. User agents are not required to support any particular content types via **'foreignObject'**. In particular, user agents are not required to support SVG content embedded within or referenced by **'foreignObject'**; SVG content within **'foreignObject'** represents an extension just as with any other type of content.

Usually, a **'foreignObject'** will be used in conjunction with the **'switch'** element and the [requiredExtensions](#) attribute to provide proper checking for user agent support and provide an alternate rendering in case user agent support is not available.

19.2.1 The **'foreignObject'** element

The **'foreignObject'** element is an extensibility point which allows user agents to offer graphical rendering features beyond those which are defined within this specification.

A conformant SVG user agent is not required to support any particular foreign namespace content within **'foreignObject'** itself nor is it required to invoke other user agents to handle particular embedded foreign object types. Ultimately, it is expected that Web browsers will support the ability for SVG to embed content from other XML grammars which use CSS or XSL to format their content, with the resulting CSS- or XSL-formatted content then subject to SVG transformations and compositing. At this time, such a capability is not a requirement. The CDF Working Group is expected to provide this functionality.

The rendered content of a **foreignObject** must be treated as atomic from the point of view of SVG compositing and transformation, as if it was a single replaced element.

All [MouseEvent](#)s that are dispatched to a **'foreignObject'** element, including mouse events that bubble from the embedded content, must have their clientX/clientY attributes adjusted so that they represent values within the initial viewport coordinate system.

Schema: foreignObject

```
<define name='foreignObject'>
  <element name='foreignObject'>
    <ref name='foreignObject.AT' />
    <ref name='foreignObject.CM' />
  </element>
</define>

<define name='foreignObject.AT' combine='interleave'>
  <ref name='svg.Core.attr' />
  <ref name='svg.FocusHighligh.attr' />
  <ref name='svg.Conditional.attr' />
  <ref name='svg.XLinkEmbed.attr' />
  <ref name='svg.Focus.attr' />
  <ref name='svg.External.attr' />
  <ref name='svg.Properties.attr' />
  <ref name='svg.Transform.attr' />
  <ref name='svg.XYWH.attr' />
</define>

<define name='foreignObject.CM'>
  <empty />
</define>
```

Attribute definitions:

x = "[<coordinate>](#)"

The x-axis coordinate of one corner of the rectangular region into which the graphics associated with the contents

of the **'foreignObject'** will be rendered.

If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

y = "<coordinate>"

The y-axis coordinate of one corner of the rectangular region into which the referenced document is placed.

If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

width = "<length>"

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

A negative value is unsupported. A value of zero disables rendering of the element. If the attribute is not specified, the effect is as if a value of "0" were specified.

[Animatable](#): yes.

height = "<length>"

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

A negative value is unsupported. A value of zero disables rendering of the element.

[Animatable](#): yes.

xlink:href = "<iri>"

An [IRI reference](#).

[Animatable](#): yes.

focusable = "true" | "false" | "auto"

See [attribute definition](#) for description.

[Animatable](#): Yes

Navigation Attributes

See [definition](#).

19.3 An example

Here is an example using a switch and foreignObject:

Example: 23_02.svg

```
<?xml version="1.0"?>
<svg width="4in" height="3in" version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg">
  <desc>This example uses the 'switch' element to provide a fallback graphical
  representation of the text, if weirdML is not supported.</desc>
  <!-- The 'switch' element will process the first child element
  whose testing attributes evaluate to true.-->
  <switch>
    <!-- Process the embedded weirdML if the requiredExtensions attribute
    evaluates to true (i.e., the user agent supports weirdML
    embedded within SVG). -->
    <foreignObject x="50" y="20" width="100" height="50"
    requiredExtensions="http://example.com/weirdMLplusSVG">
      <!-- weirdML content goes here -->
      <FreakyText xmlns="http://example.com/weirdML">
        <sparklies q="42"/>
        <throbber seed="1234"/>
        <swirl twist="yeah, baby"/>
        <txt>This is throbbing, swirly text with sparkly bits</txt>
      </FreakyText>
    </foreignObject>
    <!-- Else, process the following alternate SVG.
    Note that there are no testing attributes on the 'textArea' element.
    If no testing attributes are provided, it is as if there
    were testing attributes and they evaluated to true.-->
    <textArea x="50" y="20" width="100" height="50"
    font-size="10" font-family="Verdana">
      This is plain, conservative SVGT 1.2 text in a
      textArea.
    </textArea>
  </switch>
</svg>
```

19.4 Extensibility Module

The Extensibility Module contains the following element:

- [foreignObject](#)

[RNG] [Feature String](#)

A The SVG Micro DOM (uDOM)

Contents

- A.1 [Introduction](#)
- A.2 [Overview of the SVG uDOM](#)
 - A.2.1 [Document Access](#)
 - A.2.2 [Tree Navigation](#)
 - A.2.3 [Element Creation](#)
 - A.2.4 [Element Addition](#)
 - A.2.5 [Element Removal](#)
 - A.2.6 [Attribute and Property Access](#)
 - A.2.7 [Attribute/Property Normalization](#)
 - A.2.8 [Text Node Access](#)
 - A.2.9 [Event Listener Registration and Removal](#)
 - A.2.10 [Animation](#)
 - A.2.11 [Java package naming](#)
- A.3 [Module: dom](#)
 - A.3.1 [DOMException](#)
 - A.3.2 [Node](#)
 - A.3.3 [Element](#)
 - A.3.4 [Document](#)
 - A.3.5 [DOMImplementation](#)
- A.4 [Module: events](#)
 - A.4.1 [EventTarget](#)
 - A.4.2 [EventListener](#)
 - A.4.3 [Event](#)
 - A.4.4 [OriginalEvent](#)
 - A.4.5 [MouseEvent](#)
 - A.4.6 [TextEvent](#)
 - A.4.7 [KeyboardEvent](#)
 - A.4.8 [TimeEvent](#)
 - A.4.9 [UIEvent](#)
 - A.4.10 [WheelEvent](#)
 - A.4.11 [ProgressEvent](#)
 - A.4.12 [ConnectionEvent](#)
- A.5 [Module: smil](#)
 - A.5.1 [ElementTimeControl](#)
- A.6 [Module: global](#)
 - A.6.1 [Global](#)
 - A.6.2 [GlobalException](#)
 - A.6.3 [Connection](#)
 - A.6.4 [Timer](#)
- A.7 [Module: svg](#)
 - A.7.1 [SVGException](#)
 - A.7.2 [SVGDocument](#)
 - A.7.3 [SVGElementInstance](#)
 - A.7.4 [SVGSVGElement](#)
 - A.7.5 [SVGRGBColor](#)
 - A.7.6 [SVGRect](#)
 - A.7.7 [SVGPoint](#)
 - A.7.8 [SVGPath](#)
 - A.7.9 [SVGMatrix](#)
 - A.7.10 [SVGLocatable](#)
 - A.7.11 [SVGLocatableElement](#)
 - A.7.12 [TraitAccess](#)
 - A.7.13 [Additional accessing rules](#)
 - A.7.14 [ElementTraversal](#)
 - A.7.15 [SVGElement](#)
 - A.7.16 [SVGAnimationElement](#)
 - A.7.17 [SVGVisualMediaElement](#)
 - A.7.18 [EventListenerInitializer2](#)
 - A.7.19 [SVGGlobal](#)
 - A.7.20 [AsyncStatusCallback](#)
 - A.7.21 [AsyncURLStatus](#)

During the later stages of the SVG Mobile 1.1 specification it became obvious that there was a requirement to subset the SVG and XML DOM in order to reduce the burden on implementations. SVGT 1.2 adds new features to the uDOM, allowing for as much necessary functionality as possible, still being suitable for SVG Tiny implementations.

Furthermore, it should be possible to implement the uDOM on devices that support SVG Tiny 1.1 although, in this case, the scripting would be external to the SVG document (since SVG Tiny 1.1 does not support inline scripting).

The goal of the uDOM definition is to provide an API that allows access to initial and computed attribute and property values, to reduce the number of interfaces, to reduce run-time memory footprint using necessary features of the core XML DOM, as well as the most useful SVG features (such as transformation matrices).

The [IDL definition](#) for the uDOM is provided.

A.1 Introduction

This appendix consists of the following parts:

- [Overview of the SVG uDOM](#) An introduction to the uDOM, including a summary of supported features and descriptions by topic of the key features and constraints.
- A definition of all the interfaces in the SVG uDOM. ([DOM Core APIs](#), [DOM Events APIs](#), [SVG DOM APIs](#), [SMIL DOM APIs](#) and [Global DOM APIs](#).)

A.2 Overview of the SVG uDOM

The following sections provides an overview of the SVG uDOM's key features and constraints.

Note: Like other W3C DOM definitions, the SVG uDOM is programming-language independent. Although this appendix only contain ECMAScript and Java™ language examples, the SVG uDOM is compatible with other programming languages.

A.2.1 Document Access

The SVG uDOM must offer access to a [Document](#) object which is the root for accessing other features. The way the [Document](#) object becomes available depends on the usage context. One way to gain access to the [Document](#) object is to implement the [EventListenerInitializer2](#) interface. The SVG Tiny user agent will invoke the implementation's `initializeEventListeners` (in `dom::Element scriptElement`) method once the programming logic has been loaded and is ready to bind to the document. The [Document](#) object is sometimes accessible through other means, for example as the global 'document' variable in ECMAScript.

A.2.2 Tree Navigation

SVG uDOM only allows navigation of the element nodes in the DOM tree. Two options are available for navigating the hierarchy of elements:

- Individual element nodes with an ID value must be accessible directly via the `getElementById` method on the [Document](#) interface.
- The hierarchy of element nodes must be traversable using the facilities on the [ElementTraversal](#) interface, along with the `parentNode` attribute on the [Node](#) interface.

The [ElementTraversal](#) interface provides `firstElementChild`, `lastElementChild`, `previousElementSibling` and `nextElementSibling`, which are particularly suitable for constrained devices. These traversal mechanisms skip over intervening nodes between element nodes, such as text nodes which might only contain spaces, tabs and newlines.

A.2.3 Element Creation

SVG uDOM must allow creation of new [Elements](#).

Example: Element creation (Java)

```
String svgNS = "http://www.w3.org/2000/svg";
Element myRect = document.createElementNS(svgNS, "rect");
```

A.2.4 Element Addition

[Node](#) addition is the ability to add new elements to a document tree.

SVG uDOM must allow addition and insertion of a [Node](#).

Example: Element addition (ECMAScript)

```
var svgNS = "http://www.w3.org/2000/svg";
// Create a new <rect> element
var myRect = document.createElementNS(svgNS, "rect");
// Set the various <rect> properties before appending
...

// Add element to the root of the document
var svgRoot = document.documentElement;
svgRoot.appendChild(myRect);

// Create a new <ellipse> element
var myEllipse = document.createElementNS(svgNS, "ellipse");

// Set the various <ellipse> properties before insertion
...

// Insert the ellipse before the rectangle
svgRoot.insertBefore(myEllipse, myRect);
```

A.2.5 Element Removal

[Node](#) removal is the ability to remove an element from a document tree. SVG uDOM must allow removal of element [Nodes](#).

Example: Element removal (ECMAScript)

```
var myRect = ...; // See Element creation
var myGroup = document.getElementById("myGroup");
myGroup.appendChild(myRect);
...
myGroup.removeChild(myRect);
```

A.2.6 Attribute and Property Access

SVGT 1.2 uDOM supports two ways of accessing XML attributes and CSS properties; the standard way via [getAttributeNS](#) and [setAttributeNS](#) on the [Element](#) interface and via a new concept called *Traits*.

A trait is a potentially animatable parameter associated with an element. Trait is the typed value (e.g. a number, not just a string) that gets assigned through an XML attribute or a CSS property. Traits can be thought of as a unification and generalization of some of the notions of XML attributes and CSS properties. The trait facilities in the SVG uDOM allow for strongly-typed access to certain attribute and property values. For example, there is a [getFloatTrait\(...\)](#) method for getting an attribute or property value directly as a `float`. This contrasts the [getAttributeNS](#) method which always returns a string. The trait facilities in the SVG uDOM are available on the [TraitAccess](#) interface.

Example: Trait Access (Java)

```
float width = myRect.getFloatTrait('width');
width += 10;
myRect.setFloatTrait('width', width);
```

An important difference between [getTraitNS](#) (and all other variants of `getTrait` methods) and [getAttributeNS](#) is that [getTraitNS](#) returns the computed attribute value but [getAttributeNS](#) returns the specified attribute value (which might not exactly match the original specified value due to the possibility of user agent value normalization as described in [Attribute Normalization](#)).

Example: Difference between traits and getAttributeNS (Java)

```
<g fill="red">
<rect id="r1" x="1" y="1" width="5" height="5"/>
<rect id="r2" fill="inherit" x="1" y="1" width="5" height="5"/>
</g>
```

`r1.getTraitNS(null, "fill")` returns "red" (or equivalent normalized form, see below).

`r2.getTraitNS(null, "fill")` returns "red" (or equivalent normalized form, see below).

`r1.getAttributeNS(null, "fill")` returns "".

`r2.getAttributeNS(null, "fill")` returns "inherit".

A.2.7 Attribute/Property Normalization

A viewer implementing the uDOM is allowed to return normalized attribute values (defined in [DOM 3](#)) from [getAttributeNS](#) and the various [getTrait](#) methods (`getTrait`, `getTraitNS`, `getFloatTrait`, ...) and [getPresentationTrait](#) methods (`getPresentationTrait`, `getPresentationTraitNS`, `getFloatPresentationTrait`, ...). Following is a list of possible attribute normalizations:

- **Color normalization** "red" may be returned as "`rgb(255,0,0)`", "`#ff0000`", or other semantically identical form.
- **Out-of-range normalization** Values that are only of relevance within a certain range may be returned as a value clamped to that range. E.g. `fill-opacity="1.3"` may be returned as "1".
- **Numerical precision** "3.0" may be returned as "3", "3.00" or other semantically identical form.
- **Whitespace normalization** " 3.0 " may be returned as "3.0". Whitespace normalization also includes unquoted font names in the `css font-family` property: Font family names containing whitespace should be quoted. If quoting is omitted, any whitespace characters before and after the font name shall be ignored and any sequence of whitespace characters inside the font name shall be converted to a single space.
- **Relative to absolute IRI conversion** "exampleImage.png" may be returned as "`http://example.org/exampleImage.png`".
- **font-weight normalization** "normal" may be returned as "400", "bold" may be returned as "700".
- **Transform normalization** Any transform value may be returned as the corresponding matrix. E.g. "`scale(2,2)`" may be returned as "`matrix(0,2,0,2,0,0)`", "`scale(2,2) translate(10,5) rotate(45)`" may be returned as "`matrix(1.41, -1.41, 20, 1.41, 1.41, 10)`".
- **Path normalization** The full set of `'d/'path'` commands may be mapped down on a smaller set of commands.
 - Relative commands (c, h, l, m, q, s, t, and v) are converted to their absolute counterparts.
 - Horizontal and Vertical lines (H, h, V, and v) are converted to general lines (L and l).
 - Translate command S to command C.
 - Translate command T to command Q.
- **Display normalization** All possible `'display'` values may be mapped to `'none'`, `'inline'` or `'inherit'` since they cover all the possible `'display'` outputs for a pure SVGT1.2 viewer. E.g. "block" may be returned as "inline". For viewers in multiple namespaces, e.g. a CDF viewer, the different `'display'` properties are of importance and therefore an SVGT1.2 viewer intended for use in a multiple namespace environment is strongly recommended to keep the full range of `'display'` values.

A.2.8 Text Node Access

In the SVG uDOM, text node access is available in two alternative ways. Text access via the [TraitAccess](#) interface is available on all [SVGElement](#)s. This was available in the SVG Tiny 1.1 uDOM (used in the [JSR-226](#) specification) and is still available in order to keep backward compatibility. The SVG Tiny 1.2 uDOM specification introduces the [textContent](#) attribute on the [Node](#) interface as a more generic text access mechanism.

To access or set the text string value for an element via traits you invoke [getTrait\(\)](#) or [setTrait\(\)](#) on that element and pass `#text` as the name of the trait you want to get or set. For example, `MyTextElement.setTrait("#text", "Hello");` Text access via the `#text` mechanism must be supported on `TextContent`, `'desc'`, `'title'` and `'metadata'` elements. Text access to other elements defined within this specification (see [list of elements](#)) is not supported and an implementation should ignore any text on these elements.

The result of getting and setting text content via the `#text` mechanism is exactly the same as when using the [textContent](#) attribute. Therefore the user should be aware of the fact that styling by `'tspan'` elements will be lost if she gets a text string from the model and sets it back again.

The `#text` trait is included for compatibility with [JSR-226](#). It is recommended that where compatibility with JSR-226 implementations is not required content developers use `textContent` instead as it is more generally applicable and supports better compatibility with DOM Level 3 Core [\[DOM3\]](#).

A.2.9 Event Listener Registration and Removal

Event Listener Registration and Removal is the ability to add and remove new event listeners from a [Document](#). SVG uDOM must allow adding and removing [EventListener](#)s.

Example: Event Listeners (Java)

```
class MyEventListener implements EventListener {
    public void handleEvent(Event evt) {
        // Do whatever is needed here
    }
}
...
EventListener l = new MyEventListener();

SVGElement myRect = (SVGElement)document.getElementById("myRect");
//Listen to click events, during the bubbling phase
myRect.addEventListenerNS("http://www.w3.org/2001/xml-events", "click", 1, false, null);
...

// Remove the click listener
myRect.removeEventListenerNS("http://www.w3.org/2001/xml-events","click", 1, false);
```

The SVG uDOM only supports the bubble phase. Any attempt to specify event operations on the capture phase must raise a [DOMException](#) of type NOT_SUPPORTED_ERR.

Refer to the [DOM Events Level 3 specification](#) or the [XML Events specification introduction](#) for an explanation of the SVG event flow and the meaning of event targets, event current target, bubble and capture.

A.2.10 Animation

SVG uDOM allows code to start or end animation elements (i.e. elements implementing [SVGAnimationElement](#)).

Example: animation (ECMAScript)

```
var animateColor = document.getElementById("myAnimation");

// Start the animation 2.5 seconds from now.
animateColor.beginElementAt(2.5);
```

A.2.11 Java package naming

The SVG uDOM will use the same Java package names as the SVG 1.2 Full DOM (e.g., org.w3c.dom, org.w3c.dom.events, org.w3c.dom.svg). This allows Java applications which restrict themselves to the features in the SVG uDOM to also run in implementations that support the SVG 1.2 Full DOM.

A.3 Module: dom

A.3.1 DOMException

IDL Definition

```
exception DOMException
{
    unsigned short code;
};

// ExceptionCode
const unsigned short INDEX_SIZE_ERR = 1;
const unsigned short HIERARCHY_REQUEST_ERR = 3;
const unsigned short WRONG_DOCUMENT_ERR = 4;
const unsigned short INVALID_CHARACTER_ERR = 5;
const unsigned short NO_MODIFICATION_ALLOWED_ERR = 7;
const unsigned short NOT_FOUND_ERR = 8;
const unsigned short NOT_SUPPORTED_ERR = 9;
const unsigned short INVALID_STATE_ERR = 11;
const unsigned short INVALID_MODIFICATION_ERR = 13;
const unsigned short NAMESPACE_ERR = 14;
const unsigned short INVALID_ACCESS_ERR = 15;
const unsigned short TYPE_MISMATCH_ERR = 17;
```

Constants

INDEX_SIZE_ERR

If index or size is negative, or greater than the allowed value.

HIERARCHY_REQUEST_ERR

If any [Node](#) is inserted somewhere it doesn't belong.

WRONG_DOCUMENT_ERR

If a [Node](#) is used in a different document than the one that created it (that doesn't support it).

INVALID_CHARACTER_ERR

If an invalid or illegal character is specified, such as in an XML name.

NO_MODIFICATION_ALLOWED_ERR

If an attempt is made to modify an object where modifications are not allowed.

NOT_FOUND_ERR

If an attempt is made to reference a [Node](#) in a context where it does not exist. See [insertBefore](#) for example.

NOT_SUPPORTED_ERR

If the implementation does not support the requested type of object or operation.

INVALID_STATE_ERR

If an attempt is made to use an object that is not, or is no longer, usable.

INVALID_MODIFICATION_ERR

If an attempt is made to modify the type of the underlying object.

NAMESPACE_ERR

If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

INVALID_ACCESS_ERR

If a parameter or an operation is not supported by the underlying object.

TYPE_MISMATCH_ERR

If the type of an object is incompatible with the expected type of the parameter associated to the object.

No defined attributes

No defined methods

A.3.2 Node

The [Node](#) interface describes generic nodes in an SVG document tree.

This interface is a subset of the [Node](#) interface defined in the [DOM Level 3 Core](#). Node types that must be supported in the uDOM are [Element](#) nodes and [Document](#) nodes.

IDL Definition

```
interface Node
{
    readonly attribute DOMString namespaceURI;
    readonly attribute DOMString localName;
    readonly attribute Node parentNode;
    readonly attribute Document ownerDocument;
    attribute DOMString textContent;
    Node appendChild(in Node newChild) raises(DOMException);
    Node insertBefore(in Node newChild, in Node refChild) raises(DOMException);
    Node removeChild(in Node oldChild) raises(DOMException);
    Node cloneNode(in boolean deep);
};
```

No defined constants

Attributes

namespaceURI

Returns the namespace URI of the [Node](#).

localName

Returns the local part of the qualified name of this [Node](#). If the [Node](#) is of type [Element](#), this returns the tag name without a prefix. But, if the [Node](#) is of type [Document](#) then null is returned.

parentNode

Returns the parent [Node](#) of this [Node](#).

ownerDocument

The [Document](#) object associated with this [Node](#).

textContent

This attribute returns the text content of this [Node](#) and its descendants. When it is defined to be null, setting it has no effect. On setting, any possible children this [Node](#) may have are removed and, if it the new string is not empty or null, replaced by a single text node containing the string this attribute is set to. On getting, no serialization is performed, the returned string does not contain any markup. No whitespace normalization is performed and the returned string does not contain the white spaces in element content. Similarly, on setting, no parsing is performed either, the input string is taken as pure textual content.

[textContent](#) must be supported on all non-svg elements. For elements defined within this specification (see [list of elements](#)) [textContent](#) must be supported on the [Text Content](#), ['desc'](#), ['title'](#) and ['metadata'](#) elements. [textContent](#) on elements that do not support [textContent](#) is null. Setting [textContent](#) on an element that do not support [textContent](#) has no effect.

[textContent](#) on the [Document](#) node is always null.

An alternate way of accessing text content on elements defined within the svg specification is via the [getTrait\("#text"\)](#) syntax.

For further details see [DOM3 textContent](#).

Methods

appendChild

Appends a child to this [Node](#).

Parameters

in [Node](#) newChild The [Node](#) to be appended to this [Node](#). This is equivalent to [insertBefore](#) ([newChild](#), null).

Return value

[Node](#) The [Node](#) added.

Exceptions

[DOMException](#) HIERARCHY_REQUEST_ERR: Raised if this [Node](#) is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors or this node itself, or if this [Node](#) is of type [Document](#) and the uDOM application attempts to append a second [Element](#) node.

[DOMException](#) WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this [Node](#).

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the `newChild` node is a child of the [Document](#) node or if the child is of a type that cannot be created with [createElementNS](#).

[DOMException](#) INVALID_STATE_ERR: Raised if the `newChild` node would cause the document to go into error.

insertBefore

Inserts `newChild` before `refChild` in the child list for this [Node](#). If `refChild` is null, `newChild` is inserted at the end of the list. If the `newChild` is already part of the tree, it is first removed.

Parameters

in [Node](#) `newChild` The child to add.

in [Node](#) `refChild` The child before which the new child is added.

Return value

[Node](#) The [Node](#) being inserted.

Exceptions

[DOMException](#) HIERARCHY_REQUEST_ERR: Raised if this [Node](#) is of a type that does not allow children of the type of the `newChild` node, or if the [Node](#) to append is one of this node's ancestors or this [Node](#) itself, or if this [Node](#) is of type [Document](#) and the uDOM application attempts to append a second [Element](#) node.

[DOMException](#) WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this [Node](#).

[DOMException](#) NOT_FOUND_ERR: Raised if `refChild` is not a child of this [Node](#).

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the `newChild` node is a child of the [Document](#) node or if the child is of a type that cannot be created with [createElementNS](#).

[DOMException](#) INVALID_STATE_ERR: if the `newChild` node would cause the document to go into error.

removeChild

Removes the specified child associated with this [Node](#).

Parameters

in [Node](#) `oldChild` The [Node](#) that is to be removed.

Return value

[Node](#) The [Node](#) removed.

Exceptions

[DOMException](#) NOT_FOUND_ERR: Raised if `oldChild` is not a child of this [Node](#).

[DOMException](#) NOT_SUPPORTED_ERR: Raised if this [Node](#) is of type [Document](#) or if the child, or any of its descendants, is of a type that cannot be created with the [Document](#) method [createElementNS](#).

cloneNode

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` is null).

Cloning an [Element](#) copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any children it contains unless it is a deep clone. This includes text contained in an the [Element](#).

Note: Cloning an element that has an ID will clone it as well, and thus the ID should be changed before the cloned element is inserted into the tree as duplicate IDs produce implementation-dependent results.

Note: Document nodes is implementation dependent.

Parameters

in `boolean` `feature` If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an [Element](#)).

Return value

[Node](#) The duplicate node.

No Exceptions

Represents an [Element](#) in the document. A user agent is allowed to store normalized attribute values internally as described in [DOM 3](#).

IDL Definition

```
interface Element : Node
{
    DOMString getAttributeNS(in DOMString namespaceURI, in DOMString localName) raises(DOMException);
    void setAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName, in DOMString value) raises(DOMException);
};
```

No defined constants

No defined attributes

Methods

[getAttributeNS](#)

Retrieves an attribute value by local name and namespace [IRI](#). Per [\[XML Namespaces\]](#), applications must use the value `null` as the [namespaceURI](#) parameter for methods if they wish to have no namespace.

An SVG1.2 user agent must support [getAttributeNS](#) for certain types of attributes and may support [getAttributeNS](#) for other types of attributes. An SVG1.2 user agent must support [getAttributeNS](#) for:

- All attributes on any element *not* listed under [List of SVG1.2 Elements](#).
- Any attribute on any element listed under [List of SVG1.2 Elements](#) which also is defined as being accessible by the [getTrait](#) method on [SVGElement](#) in the [SVTT1.2 Trait Table](#).

In order to allow for implementation efficiency on small devices, an SVG1.2 user agent is *not* required to support [getAttributeNS](#) for other types of attributes.

Note: Because it is not required that user agents support [getAttributeNS](#) on every possible type of attribute, conforming SVG1.2 content must use [getAttributeNS](#) only on those types of attributes which SVG1.2 user agents must support, as described above.

A viewer implementing the uDOM is allowed to return normalized values in [getAttributeNS](#) as defined in [Attribute Normalization](#).

Parameters

in [DOMString](#) [namespaceURI](#) The namespace URI of the attribute to retrieve.

in [DOMString](#) [localName](#) The local name of the attribute to retrieve.

Return value

[DOMString](#) The attribute value.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if [getAttributeNS](#) tries to get an unsupported attribute.

[setAttributeNS](#)

Adds/sets a new attribute. The value to set is a simple string; it is not parsed as it is being set. So any markup is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. Per [\[XML Namespaces\]](#), applications must use the value `null` as the [namespaceURI](#) parameter for methods if they wish to have no namespace.

A uDOM implementation must support [setAttributeNS](#) for all attributes.

Parameters

in [DOMString](#) [namespaceURI](#) The namespace URI of the attribute to create or alter.

in [DOMString](#) [qualifiedName](#) The local name of the attribute to create or alter.

in [DOMString](#) [value](#) The value to set in string form.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if [setAttributeNS](#) tries to set an unsupported attribute.

NO_MODIFICATION_ALLOWED_ERR: Raised if the attribute cannot be modified.

INVALID_CHARACTER_ERR: Raised if the specified local name is not an XML name.

NAMESPACE_ERR: Raised if the [qualifiedName](#) is malformed per the Namespaces in XML specification, if the [qualifiedName](#) has a prefix and the [namespaceURI](#) is `null`, if the [qualifiedName](#) has a prefix that is "xml" and the [namespaceURI](#) is different from "http://www.w3.org/XML/1998/

namespace", if the `qualifiedName` or its prefix is "xmlns" and the namespaceURI is different from "http://www.w3.org/2000/xmlns/", or if the namespaceURI is "http://www.w3.org/2000/xmlns/" and neither the `qualifiedName` nor its prefix is "xmlns".

A.3.4 Document

The [Document](#) interface represents an XML Document.

This interface is a subset of the Document interface defined in the [DOM Level 3 Core](#).

Note: The behavior of the following attributes and methods from the [Node](#) interface when called on a [Document](#) object:

- the [parentNode](#) attribute will be null
- [appendChild](#) throws HIERARCHY_REQUEST_ERR
- [insertBefore](#) throws HIERARCHY_REQUEST_ERR
- [removeChild](#) throws NOT_SUPPORTED_ERR

IDL Definition

```
interface Document : Node
{
    readonly attribute DOMImplementation implementation;
    Element createElementNS(in DOMString namespaceURI, in DOMString qualifiedName) raises(DOMException);
    readonly attribute Element documentElement;
    Element getElementById(in DOMString elementId);
};
```

No defined constants

Attributes

implementation

The [DOMImplementation](#) object that handles this document.

documentElement

Return a child element of this document node which corresponds to the root-most element in the XML document. For SVG documents it must be [SVGSVGElement](#), but return type is [Element](#) for DOM Core compatibility and to allow for future extensions.

Methods

createElementNS

Create a new [Element](#) based on the specified (qualified) tag name.

Parameters

in [DOMString](#) namespaceURI The namespace uri for the newly created element.

in [DOMString](#) qualifiedName The qualified name for the newly created element (For example: "rect", to create a **'rect'** element).

Return value

[Element](#) The newly created [Element](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the type of element is not supported by the implementation.

INVALID_CHARACTER_ERR: Raised if the specified local name is not an XML name.

NAMESPACE_ERR: Raised if the `qualifiedName` is malformed per the Namespaces in XML specification, if the `qualifiedName` has a prefix and the namespaceURI is null, if the `qualifiedName` has a prefix that is "xml" and the namespaceURI is different from "http://www.w3.org/XML/1998/namespace", if the `qualifiedName` or its prefix is "xmlns" and the namespaceURI is different from "http://www.w3.org/2000/xmlns/", or if the namespaceURI is "http://www.w3.org/2000/xmlns/" and neither the `qualifiedName` nor its prefix is "xmlns".

getElementById

Return the [Element](#) in the current document with the given unique ID.

Parameters

in [DOMString](#) elementId The ID of the object to be retrieved.

Return value

[Element](#) The [Element](#) in the current document with the given unique ID. If no such element exists, this returns null.

No Exceptions

A.3.5 DOMImplementation

The [DOMImplementation](#) interface provides methods for performing operations that are independent of any particular instance of the document object model.

This interface is a subset of the DOMImplementation interface defined in the [DOM Level 3 Core](#).

IDL Definition

```
interface DOMImplementation
{
    boolean hasFeature(in DOMString feature, in DOMString version);
};
```

No defined constants

No defined attributes

Methods

[hasFeature](#)

Test if the DOM implementation implements a specific feature and version. Supports both [DOM Features](#) and [SVG's feature strings](#).

Parameters

in [DOMString](#) feature The name of the feature to test.

in [DOMString](#) version This is the version number of the feature to test.

Return value

boolean Will be true if the feature is implemented in the specified version, false otherwise.

No Exceptions

A.4 Module: events

A.4.1 EventTarget

This interface represents an event target, and is a subset of the [EventTarget](#) interface defined in the [DOM Level 3 Event model](#).

This interface is implemented by an object ([SVGElement](#), [SVGDocument](#), [SVGElementInstance](#)) that can notify listeners about events and allows registration and removal of [EventListener](#) objects.

IDL Definition

```
interface EventTarget
{
    void addEventListener(in DOMString type, in EventListener listener, in boolean useCapture) raises(DOMException);
    void removeEventListener(in DOMString type, in EventListener listener, in boolean useCapture) raises(DOMException);
    void addEventListenerNS(in DOMString namespaceURI, in DOMString type, in EventListener listener, in boolean useCapture, in DOMObject
evtGroup) raises(DOMException);
    void removeEventListenerNS(in DOMString namespaceURI, in DOMString type, in EventListener listener, in boolean useCapture) raises
(DOMException);
};
```

No defined constants

No defined attributes

Methods

[addEventListener](#)

This method registers the specified listener with the event target. If an [EventListener](#) is added to an [EventTarget](#) while it is processing an event, it will not be triggered by the current actions. If multiple identical [EventListener](#)s are registered on the same [EventTarget](#) with the same parameters the duplicate instances are discarded. They do not cause the [EventListener](#) to be called twice and since they are discarded they do not need to be removed with the [removeEventListener](#) method.

Parameters

in [DOMString](#) type The type of event to listen to.

in [EventListener](#) listener Will be notified when an event of the desired type happens on this target or one of its descendant.

in boolean `useCapture` This must be 'false' since capture phase is not supported.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if `useCapture` is true since capture phase is not required in SVG Tiny and implementations may not support registering listeners for the capture phase.

removeEventListener

This method removes the specified listener from the event target. If an [EventListener](#) is removed from an [EventTarget](#) while it is processing an event, it will not be triggered by the current actions. Calling [removeEventListener](#) with arguments which do not identify any currently registered [EventListener](#) on the [EventTarget](#) has no effect.

Parameters

in [DOMString](#) type The type of event that was listened to.

in [EventListener](#) listener The listener that was previously registered.

in boolean `useCapture` This can only be 'false' since capture phase is not supported.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if `useCapture` is true since capture phase is not required in SVG Tiny and implementations may not support registering listeners for the capture phase.

addEventListenerNS

This method registers the specified listener with the event target. If an [EventListener](#) is added to an [EventTarget](#) while it is processing an event, it will not be triggered by the current actions. If multiple identical [EventListener](#)s are registered on the same [EventTarget](#) with the same parameters the duplicate instances are discarded. They do not cause the [EventListener](#) to be called twice and since they are discarded they do not need to be removed with the [removeEventListenerNS](#) method.

Parameters

in [DOMString](#) namespaceURI The namespace URI of the event.

in [DOMString](#) type The type of event to listen to.

in [EventListener](#) listener Will be notified when an event of the desired type happens on this target or one of its descendant.

in boolean `useCapture` This must be 'false' since capture phase is not supported.

in DOMObject `evtGroup` For compatibility with [DOM Level 3](#). In SVG1.2 this parameter must be `null`.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if `useCapture` is true since capture phase is not required in SVG Tiny and implementations may not support registering listeners for the capture phase.

removeEventListenerNS

This method removes the specified listener from the event target. If an [EventListener](#) is removed from an [EventTarget](#) while it is processing an event, it will not be triggered by the current actions. Calling [removeEventListenerNS](#) with arguments which do not identify any currently registered [EventListener](#) on the [EventTarget](#) has no effect.

Parameters

in [DOMString](#) namespaceURI The namespace URI of the event.

in [DOMString](#) type The type of event that was listened to.

in [EventListener](#) listener The listener that was previously registered.

in boolean `useCapture` This must be 'false' since capture phase is not supported.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if `useCapture` is true since capture phase is not required in SVG Tiny and implementations may not support registering listeners for the capture phase.

A.4.2 EventListener

This interface represents an event listener, and is a subset of the [EventListener](#) interface defined in the [DOM Level 3 Event model](#).

This interface must be implemented and registered on an [EventTarget](#) using the [addEventListenerNS](#) method to be notified about events that occur on or bubble through the event target.

IDL Definition

```
interface EventListener
{
    void handleEvent(in Event evt);
}
```

```
};
```

No defined constants

No defined attributes

Methods

handleEvent

This method is called whenever an event occurs of the type for which the [EventListener](#) interface was registered. The [Event](#) object contains the necessary information pertaining to the event, such as its target and type.

Parameters

in [Event](#) evt The [Event](#) object containing necessary event information.

No Return value

No Exceptions

A.4.3 Event

The [Event](#) interface is used to provide contextual information about an event to the handler processing the event. An object which implements the [Event](#) interface is passed as the first parameter to the [handleEvent](#) call. If an event target is an element instance (see [SVGElementInstance](#)), the [currentTarget](#) is an implementation of [EventTarget](#) that does not implement the [Node](#) interface. In profiles of SVG that support the [SVGElementInstance](#) interface, the [currentTarget](#) is an [SVGElementInstance](#).

IDL Definition

```
interface Event
{
    readonly attribute EventTarget target;
    readonly attribute EventTarget currentTarget;
    readonly attribute DOMString type;
    readonly attribute DOMString namespaceURI;
    readonly attribute boolean cancelable;
    boolean isDefaultPrevented();
    void stopPropagation();
    void preventDefault();
};
```

No defined constants

Attributes

target

Used to indicate the event target. This attribute contains the target node when used with the DOM event flow.

currentTarget

Used to indicate the [EventTarget](#) whose [EventListeners](#) are currently being processed. In SVG Tiny, this is always an object to which event listener was attached.

type

This method returns the event type information. The name of the event is case-sensitive. For a list of supported event types see the [Supported Events](#) chapter.

namespaceURI

The namespace associated with this event at creation time, or null if it is unspecified.

cancelable

Used to indicate whether or not an event can have its default action prevented (see also [Default actions and cancelable events](#) in [DOM Events](#)). If the default action can be prevented the value is true, otherwise the value is false.

Methods

isDefaultPrevented

This method will return true if the method [preventDefault\(\)](#) has been called for this event, false otherwise.

No Parameters

Return value

boolean true if [preventDefault\(\)](#) has been called for this event.

No Exceptions

stopPropagation

This method is used to prevent event listeners of the same group to be triggered but its effect is deferred until all event listeners attached on the [currentTarget](#) have been triggered (see [Event propagation and event groups](#) in [DOM Events](#)). Once it has been called, further calls to that method have no additional effect.

Note: This method does not prevent the default action from being invoked; use [preventDefault\(\)](#) for that effect.

No Parameters

No Return value

No Exceptions

preventDefault

If an event is [cancelable](#), the [preventDefault\(\)](#) method is used to signify that the event is to be canceled, meaning any default action normally taken by the implementation as a result of the event will not occur (see also [Default actions and cancelable events](#) in [DOM Events](#)), and thus independently of event groups. Calling this method for a non-cancelable event has no effect.

Note: This method does not stop the event propagation; use [stopPropagation\(\)](#) for that effect.

No Parameters

No Return value

No Exceptions

A.4.4 OriginalEvent

[OriginalEvent](#) is supported on events going through the following element types: ['animation'](#) and ['foreignObject'](#) (when ['foreignObject'](#) has an ['xlink:href'](#) attribute).

IDL Definition

```
interface OriginalEvent
{
    readonly attribute Event originalEvent;
};
```

No defined constants

Attributes

originalEvent

For the supported element (['animation'](#) and ['foreignObject'](#)), events can be dispatched to the [SVGElementInstance](#) objects corresponding to the content referenced by the supported element. UI events that are dispatched to these [SVGElementInstance](#) objects can bubble up to the supported element. Upon bubbling to the supported element, the User Agent must manufacture new [Event](#) and [OriginalEvent](#) objects. The new [Event](#) object must be of the same type and carry the same information as the original event object but which has the supported element as the "target". The new [OriginalEvent](#) object must set the [originalEvent](#) attribute to the original event object.

No defined methods

A.4.5 MouseEvent

[Event](#) that provides specific contextual information associated with pointing device events.

[Event](#) types that are [MouseEvent](#)s: [click](#), [mousedown](#), [mouseup](#), [mouseover](#), [mousemove](#), [mouseout](#).

IDL Definition

```
interface MouseEvent : UIEvent
{
    readonly attribute long screenX;
    readonly attribute long screenY;
    readonly attribute long clientX;
    readonly attribute long clientY;
    readonly attribute unsigned short button;
};
```

No defined constants

Attributes

screenX

The horizontal coordinate at which the event occurred relative to the origin of the screen coordinate system.

screenY

The vertical coordinate at which the event occurred relative to the origin of the screen coordinate system.

clientX

The horizontal coordinate at which the event occurred relative to the uDOM implementation's client area.

clientY

The vertical coordinate at which the event occurred relative to the uDOM implementation's client area.

button

During mouse events caused by the depression or release of a mouse button, this is used to indicate which mouse button changed state. The values for button range from zero to indicate the left button of the mouse, one to indicate the middle button if present, and two to indicate the right button. For mice configured for left handed use in which the button actions are reversed the values are instead read from right to left.

No defined methods

A.4.6 TextEvent

One or more characters have been entered.

[Event](#) type that is a [TextEvent](#): [textInput](#).

IDL Definition

```
interface TextEvent : UIEvent
{
    readonly attribute DOMString data;
};
```

```
};
```

No defined constants

Attributes

data

Holds the value of the characters generated by the character device. This may be a single Unicode character or a non-empty sequence of Unicode characters. This attribute must not be `null` or contain the empty string.

No defined methods

A.4.7 KeyboardEvent

Provides specific contextual information associated with keyboard devices. Each [KeyboardEvent](#) references a key using an identifier.

[Event](#) types that are [KeyboardEvents](#): [keydown](#), [keyup](#).

IDL Definition

```
interface KeyboardEvent : UIEvent
{
    readonly attribute DOMString keyIdentifier;
};
```

No defined constants

Attributes

keyIdentifier

Holds the identifier of the key.

No defined methods

A.4.8 TimeEvent

[Event](#) that is fired by all [Timed Elements](#).

[Event](#) types that are [TimeEvents](#): [beginEvent](#), [endEvent](#), [repeatEvent](#).

IDL Definition

```
interface TimeEvent : Event
{
    readonly attribute long detail;
};
```

No defined constants

Attributes

detail

Specifies detailed information about the [TimeEvent](#), the information depends on the type of event. For [beginEvent](#) and [endEvent](#) the [detail](#) field is not used. For [repeatEvent](#) the [detail](#) field contains the current repeat iteration.

No defined methods

A.4.9 UIEvent

The [UIEvent](#) interface provides specific contextual information associated with User Interface events.

[Event](#) types that are [UIEvents](#): [focusin](#), [focusout](#), [activate](#), [MouseEvent](#), [TextEvent](#), [KeyboardEvent](#), [WheelEvent](#).

IDL Definition

```
interface UIEvent : Event
{
    readonly attribute long detail;
};
```

No defined constants

Attributes

detail

Specifies detailed information about the [UIEvent](#), the information depends on the type of event. For [focusin](#) and [focusout](#) the [detail](#) field is not used. For [activate](#) the [detail](#) field indicates the type of activation that occurs: 1 for a simple activation (e.g. a simple click or Enter), 2 for hyperactivation (for instance a double click or Shift Enter).

No defined methods

A.4.10 WheelEvent

Many devices today have a rotational input method, such as the wheel on a mouse or the "jog dial" of a phone or PDA.

The "wheel" event is triggered when the user rotates the rotational input device. Listeners for this event may only be attached to the Element node that is the Document node's `documentElement`. If a listener for this event is attached to another node in the document, the user agent MUST NOT trigger it when the event occurs.

[Event](#) type that is a [WheelEvent](#): [wheel](#).

IDL Definition

```
interface WheelEvent : UIEvent
{
    readonly attribute unsigned long wheelDelta;
};
```

No defined constants

Attributes

wheelDelta

Indicates the number of "clicks" the wheel has been rotated. A positive value indicates that the wheel has been rotated away from the user (or in a right-hand manner on horizontally aligned devices) and a negative value indicates that the wheel has been rotated towards the user (or in a left-hand manner on horizontally aligned devices).

A "click" is defined to be a unit of rotation. On some devices this is a finite physical step. On devices with smooth rotation, a "click" becomes the smallest measurable amount of rotation.

No defined methods

A.4.11 ProgressEvent

Many resources, such as raster images, movies and complex SVG content can take a substantial amount of time to download. In some use cases the author would prefer to delay the display of content or the beginning of an animation until the entire content of a file has been downloaded. In other cases, the author may wish to give the viewer some feedback that a download is in progress (e.g. a loading progress screen).

The [ProgressEvent](#) occurs when the user agent makes progress loading a resource (local or external) referenced by an [xlink:href](#) attribute.

The user agent must dispatch a [ProgressEvent](#) at the beginning of a load operation (i.e. just before starting to access the resource). This event is of type [preload](#).

The user agent must dispatch a [ProgressEvent](#) at the end of a load operation (i.e. after load is complete and the user agent is ready to render the corresponding resource). This event is of type [postload](#).

The user agent may dispatch [ProgressEvent](#)s between the [preload](#) event and the [postload](#) events. Such events are of type [loadProgress](#).

[Event](#) types that are [ProgressEvents](#): [loadProgress](#), [preload](#), [postload](#).

IDL Definition

```
interface ProgressEvent : Event
{
    readonly attribute boolean lengthComputable;
    readonly attribute unsigned long loaded;
    readonly attribute unsigned long total;
};
```

No defined constants

Attributes

lengthComputable

If false the total number of bytes (total) cannot be computed and the value of total should be ignored. This might occur if the size of the downloaded resource is unknown or if the data has already arrived.

loaded

Specifies the number of bytes downloaded since the beginning of the download. This value is ignored for a 'preload' or 'postload' event.

total

Specifies the expected total number of bytes expected in a load operation. For a 'loadProgress' event, it should specify the total number of bytes expected.

No defined methods

Example: ProgressEvent

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
xmlns:xlink="http://www.w3.org/1999/xlink">

<script type="application/ecmascript"><![CDATA[
function showImage(imageHref) {
var image = document.getElementById('myImage');
image.setTraitNS("http://www.w3.org/1999/xlink", "href", imageHref);
}

function imageLoadStart (evt) {
var progressBar = document.getElementById('progressBar');
progressBar.setFloatTrait("width", 0);
var loadAnimation = document.getElementById('loadAnimation');
loadAnimation.beginElement();
}

function imageLoadProgress (evt) {
var progressBar = document.getElementById('progressBar');
progressBar.setFloatTrait("width", 100*(evt.loaded/evt.total));
}

function imageLoadComplete (evt) {
var progressBar = document.getElementById('progressBar');
progressBar.setFloatTrait("width", 100);
var loadAnimation = document.getElementById('loadAnimation');
loadAnimation.endElement();
}
]}></script>

<image id="myImage" xlink:href="imageA.png" width="300" height="400">
<handler type="application/ecmascript" ev:event="preload">
imageLoadStart(evt);
</handler>

<handler type="application/ecmascript" ev:event="loadProgress">
imageLoadProgress(evt);
</handler>

<handler type="application/ecmascript" ev:event="postload">
imageLoadComplete(evt);
</handler>
</image>

<rect width="120" height="30" y="400">
<handler type="application/ecmascript" ev:event="click">
showImage('imageB.png');
</handler>
</rect>

<animate id="loadAnimation" ... />

<rect id="progressBar" ... />
</svg>
```

A.4.12 ConnectionEvent

The [ConnectionEvent](#) is used to represent all events that may occur during the processing of connections, as described in the [Connection](#) interface. These events represent not only data received through a connection, but also the establishment and closing of a connection, as well as error conditions that may occur.

[Event](#) types that are [ConnectionEvents](#): [connectionConnected](#), [connectionDataSent](#), [connectionDataReceived](#), [connectionClosed](#), [connectionError](#).

IDL Definition

```
interface ConnectionEvent : Event
{
    const unsigned short NO_ERR = 0;
    const unsigned short NETWORK_ERR = 1;
    readonly attribute unsigned short errorCode;
    readonly attribute sequence<octet> receivedData;
};
```

Constants

NO_ERR

When a connection event is dispatched and does not represent an error, its [errorCode](#) field is set to this value so that it can be distinguished from error

related events.

NETWORK_ERR

When an error occurs during the lifetime of a connection, an event is dispatched with its [errorCode](#) field set to this value.

Attributes

errorCode

When an event indicating that an error has occurred is dispatched, this field contains the corresponding error constant. In all other situations this field is set to NO_ERR.

receivedData

When an event indicating that data has been successfully received is dispatched, this field contains the sequence of octets that represents that data. In all other situations this field is null.

No defined methods

A.5 Module: smil

A.5.1 ElementTimeControl

This interface defines common methods for elements which define animation behaviors compatible with SMIL Animation ([Timed Elements](#) and the [svg](#) element).

IDL Definition

```
interface ElementTimeControl
{
    void beginElementAt(in float offset);
    void beginElement();
    void endElementAt(in float offset);
    void endElement();
    void pauseElement();
    void resumeElement();
    readonly attribute boolean isPaused;
};
```

No defined constants

Attributes

isPaused

true if the animation is paused, false otherwise. See [Paused element and the active duration](#).

Note: An element that is stopped (has reached the end of its active duration) is not paused.

Methods

beginElementAt

Creates a begin instance time for the current time plus or minus the passed offset. The new instance time is added to the [begin instance times list](#).

Parameters

in float **offset** The offset in seconds at which to begin the element.

No Return value

No Exceptions

beginElement

Creates a begin instance time for the current time. The new instance time is added to the [begin instance times list](#). This is equivalent to beginElementAt(0).

No Parameters

No Return value

No Exceptions

endElementAt

Creates an end instance time for the current time plus or minus the passed offset. The new instance time is added to the [end instance times list](#).

Parameters

in float **offset** The offset in seconds at which to end the element.

No Return value

No Exceptions

endElement

Creates an end instance time for the current time. The new instance time is added to the [end instance times list](#). This is equivalent to endElementAt(0).

No Parameters

No Return value

No Exceptions

pauseElement

Pauses the timed element. See [Paused element and the active duration](#).

No Parameters

No Return value

No Exceptions

resumeElement

Resumes the timed element. See [Paused element and the active duration](#).

No Parameters

No Return value

No Exceptions

A.6 Module: global

A.6.1 Global

The [Global](#) interface is designed to be the parent interface for language specific window objects. It is currently defined to be empty.

IDL Definition

```
interface Global
{
};
```

No defined constants

No defined attributes

No defined methods

A.6.2 GlobalException

An exception thrown for problems occurring with methods in the global space.

IDL Definition

```
exception GlobalException
{
    unsigned short code;
};

// ExceptionCode
const unsigned short NOT_CONNECTED_ERR = 1;
const unsigned short ENCODING_ERR = 2;
```

Constants

NOT_CONNECTED_ERR

Occurs when one attempts to send data over a closed connection, or a connection fails to be established.

ENCODING_ERR

Occurs when encoding or decoding a string fails.

No defined attributes

No defined methods

A.6.3 Connection

The [Connection](#) interface provides an API for socket-level communication. The socket must be set up to use the TCP protocol. A [Connection](#) object is created using the [SVGGlobal](#) interface method [createConnection\(\)](#).

Since it inherits from the [EventTarget](#) interface, it is possible to register event listeners on the [Connection](#) interface in order to handle the various types of [ConnectionEvent](#) that are dispatched during the processing of connections.

After a connection has been established the [Connection](#) object must dispatch a [ConnectionEvent](#) with the type '[connectionConnected](#)' to event handlers registered on that [Connection](#) object.

After data has been successfully sent, the [Connection](#) object must dispatch a [ConnectionEvent](#) with the type '[connectionDataSent](#)' to event handlers registered on that [Connection](#) object. The event is dispatched exactly once for each call to [send\(\)](#).

When data has been received, the [Connection](#) object must dispatch a [ConnectionEvent](#) with the type '[connectionDataReceived](#)' to event handlers registered on that [Connection](#) object.

In the circumstance that the connection is closed by any peer, a [ConnectionEvent](#) with the type '[connectionClosed](#)' must be dispatched to event handlers registered on that [Connection](#) object.

If an error occurs, a [ConnectionEvent](#) with the type '[connectionError](#)' must be dispatched to event handlers registered on that [Connection](#) object. This must cause the connection to be closed.

Data must be sent and received over the connection as a binary stream of octets.

The [connect\(\)](#) and [send\(\)](#) methods work asynchronously (non-blocking). Multiple calls to [send\(\)](#) can be made in sequence without waiting for the [connectionDataSent](#) event to have been dispatched. Each set of data must be queued and sent, in FIFO order, by the UA. Each [send\(\)](#) will result in a [connectionDataSent](#) event except when an error occurs.

The `send()` method must raise an `GlobalException` of type `NOT_CONNECTED_ERR` exception if the connection is not open at the time the call is made. All other errors are treated asynchronously, in the form of `connectionError` events.

IDL Definition

```
interface Connection : EventTarget
{
    readonly attribute boolean connected;
    void connect(in DOMString host, in unsigned short port) raises(GlobalException);
    void send(in sequence<octet> data) raises(GlobalException);
    void close();
};
```

No defined constants

Attributes

`connected`

This boolean field will be `true` if a connection is currently open, and `false` otherwise.

Methods

`connect`

Cause a connection to be open to the provided host and port, where the host may be either a domain name or an IP address. If the connection fails to be established, an `GlobalException` of type `NOT_CONNECTED_ERR` MUST be raised. Once the connection is established, the `connected` field must be set to `true`.

Parameters

in `DOMString host` A domain name or IP address that the TCP connection is to be established to.

in `unsigned short port` The port to which the TCP connection is being established.

No Return value

Exceptions

`GlobalException` `NOT_CONNECTED_ERR`: Raised if the connection fails.

`send`

This method sends the data over the connection. It transmits a sequence of octets without modifying it in any way. A string can be transmitted after having been converted to a sequence of octets using the `stringToBinary` method on the `SVGGlobal` interface.

Parameters

in `sequence<octet> data` The data to send.

No Return value

Exceptions

`GlobalException` `NOT_CONNECTED_ERR`: Raised if the connection is not open when `send` is called.

`close`

This method causes the connection to be closed. It is asynchronous and any data queued for transmission using the `send()` method but not yet successfully transmitted must be sent before the connection is effectively closed. However, any call to `send()` occurring after `close()` has been called will raise an exception as if the connection had already been closed, and the `connected` field must be set to `false` immediately.

No Parameters

No Return value

No Exceptions

A.6.4 Timer

The `Timer` interface provides an API for scheduling execution. A `Timer` object is always either in the running (attribute `running` is `true`) or waiting (attribute `running` is `false`) state. After each iteration of the timer, an `Event` of type `'timer'` is triggered.

IDL Definition

```
interface Timer : events::EventTarget
{
    attribute long delay;
    attribute long interval;
    readonly attribute boolean running;
    void start();
    void stop();
};
```

No defined constants

Attributes

delay

Initial delay before the first execution is fired, 0 meaning fire as soon as possible.

interval

The period between each execution, if -1, no additional execution will be fired.

running

The current state of the timer: `true` if the timer is running, `false` if the timer isn't running.

Methods

start

Starts the timer.

No Parameters

No Return value

No Exceptions

stop

Stops the timer.

No Parameters

No Return value

No Exceptions

A.7 Module: svg

A.7.1 SVGException

An exception thrown for SVG-specific errors, such as noninvertable matrix in [inverse](#).

IDL Definition

```
exception SVGException
{
    unsigned short code;
};

// ExceptionCode
const unsigned short SVG_INVALID_VALUE_ERR = 1;
const unsigned short SVG_MATRIX_NOT_INVERTABLE = 2;
```

Constants

SVG_INVALID_VALUE_ERR

Value passed to an SVG-specific method is invalid, such as out of range color component in [createSVGRGBColor](#).

SVG_MATRIX_NOT_INVERTABLE

Matrix that has a determinant equal to zero, and therefore not invertable.

No defined attributes

No defined methods

A.7.2 SVGDocument

IDL Definition

```
interface SVGDocument : Document, EventTarget
{
    readonly attribute SVGGlobal global;
};
```

No defined constants

Attributes

global

Access to the [SVGGlobal](#) object.

No defined methods

A.7.3 SVGElementInstance

For each '[use](#)' element, the uDOM represents the referenced content with a shadow tree of [SVGElementInstance](#) objects (the "instance tree") of type [SVGElementInstance](#). An [SVGElementInstance](#) represents a single [Node](#) in the instance tree.

If the `'use'` element references a simple graphics element such as a `'rect'`, then there is only a single [SVGElementInstance](#) object, and the [correspondingElement](#) attribute on this [SVGElementInstance](#) object is the [SVGElement](#) that corresponds to the referenced `'rect'` element.

If the `'use'` element references a `'g'` which contains two `'rect'` elements, then the instance tree contains three [SVGElementInstance](#) objects, a root [SVGElementInstance](#) object whose [correspondingElement](#) is the [SVGElement](#) object for the `'g'`, and then two child [SVGElementInstance](#) objects, each of which has its [correspondingElement](#) that is an [SVGElement](#) object representing the `'rect'`. Note however that the uDOM does not provide methods to navigate between an [SVGElementInstance](#) and its children [SVGElementInstance](#)s.

IDL Definition

```
interface SVGElementInstance : EventTarget
{
    readonly attribute SVGElement correspondingElement;
    readonly attribute SVGElement correspondingUseElement;
};
```

No defined constants

Attributes

correspondingElement

The corresponding element to which this object is an instance. For example, if a `'use'` element references a `'rect'` element, then an [SVGElementInstance](#) is created, with its [correspondingElement](#) being the [SVGElement](#) object for the `'rect'` element.

correspondingUseElement

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

No defined methods

Example: Ecma script usage of the SVGElementInstance interface.

In this example, three `'use'` elements use the same `'rect'` element. Each `'use'` has different `'fill'` properties that are inherited down to the used `'rect'`. The result is three `'rect'` elements with different `'fill'` colors. Clicking one of these three `'rect'` elements will cause a fourth `'rect'` to change color to match the clicked one. Worth noticing is that if the original `'rect'` had not been in the `'defs'` element the script would go into error when the original `'rect'` is clicked. This is because the `'currentTarget'` attribute would return an `'SVGElement'` that doesn't have the `'correspondingUseElement'` attribute.

```
<svg version="1.2" baseProfile="tiny" width="640" height="480"
viewBox="0 0 640 480" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ev="http://www.w3.org/2001/xml-events">
<defs>
<rect id="r1" width="90" height="65"/>
</defs>

<use xlink:href="#r1" x="50" y="200" fill="red"/>
<use xlink:href="#r1" x="250" y="200" fill="blue"/>
<use xlink:href="#r1" x="450" y="200" fill="green"/>

<rect id="r2" x="250" y="50" width="90" height="65"/>

<ev:listener ev:observer="r1" ev:event="clickEvent" ev:handler="handler"/>
<handler id="handler" type="application/ecma">changeColor(evt);</handler>

<script type="text/ecmascript">
<![CDATA[
var target = document.getElementById("r2");

function changeColor(evt)
{
    var useElement = evt.currentTarget.correspondingUseElement;
    target.setRGBColorTrait("fill", useElement.getRGBColorTrait("fill"));
}
]]>
</script>
</svg>
```

A.7.4 SVGSVGElement

This interface represents the `'svg'` element in the (SVG) document tree.

User Agent Transforms

The uDOM attributes [currentScale](#), [currentRotate](#) and [currentTranslate](#) are combined to form a user agent transformation which is applied at the outermost level on the SVG document (i.e. outside the `'svg'` element) if "magnification" is enabled (i.e. [zoomAndPan](#) attribute is set to "magnify"). Their values can potentially be modified through user-agent specific UI. User agent transformation can be obtained by multiplying the matrix

$$\begin{bmatrix} \text{currentScale} & 0 & \text{currentTranslate.x} & [\cos(\text{currentRotate}) & -\sin(\text{currentRotate})] \\ [& 0 & \text{currentScale} & \text{currentTranslate.y} & \text{by} & [\sin(\text{currentRotate}) & \cos(\text{currentRotate})] \\ [& 0 & 0 & 1 &] & [& 0 & 0 & 1] \end{bmatrix}$$

i.e. (translate, then scale, then rotate the coordinate system). The reference point for scale and rotate operations is the origin (0, 0).

IDL Definition

```
interface SVGSVGElement : SVGLocatableElement, SVGAnimationElement
{
    const unsigned short NAV_AUTO           = 1;
    const unsigned short NAV_NEXT          = 2;
    const unsigned short NAV_PREV          = 3;
    const unsigned short NAV_UP            = 4;
    const unsigned short NAV_UP_RIGHT      = 5;
    const unsigned short NAV_RIGHT         = 6;
    const unsigned short NAV_DOWN_RIGHT    = 7;
    const unsigned short NAV_DOWN          = 8;
    const unsigned short NAV_DOWN_LEFT     = 9;
    const unsigned short NAV_LEFT          = 10;
    const unsigned short NAV_UP_LEFT       = 11;
    attribute float currentScale;
    attribute float currentRotate;
    readonly attribute SVGPoint currentTranslate;
    readonly attribute SVGRect viewport;
    attribute float currentTime;
    SVGMatrix createSVGMatrixComponents(in float a, in float b, in float c, in float d, in float e, in float f);
    SVGRect createSVGRect();
    SVGPath createSVGPath();
    SVGRGBColor createSVGRGBColor(in long red, in long green, in long blue) raises(SVGException);
    void moveFocus(in unsigned short motionType) raises(DOMException);
    void setFocus(in DOMObject object) raises(DOMException);
    DOMObject getCurrentFocusedObject();
};
```

Constants

NAV_AUTO

Indicates that focus must move to the next focusable object according to the User Agent's own algorithm.

NAV_NEXT

Indicates that focus must move to the next focusable object according to current [nav-next](#) value.

NAV_PREV

Indicates that focus must move to the previous focusable object according to current [nav-prev](#) value.

NAV_UP

Indicates a request that focus must move in the given direction.

NAV_UP_RIGHT

Indicates a request that focus must move in the given direction.

NAV_RIGHT

Indicates a request that focus must move in the given direction.

NAV_DOWN_RIGHT

Indicates a request that focus must move in the given direction.

NAV_DOWN

Indicates a request that focus must move in the given direction.

NAV_DOWN_LEFT

Indicates a request that focus must move in the given direction.

NAV_LEFT

Indicates a request that focus must move in the given direction.

NAV_UP_LEFT

Indicates a request that focus must move in the given direction.

Attributes

currentScale

On read : Returns the current user agent scale (zoom) coefficient. The initial value for currentScale is 1.

On write : Sets the current user agent scale (zoom) coefficient.

Exceptions

[DOMException](#) INVALID_ACCESS_ERR: Raised if the scale value is set to zero.

currentRotate

On read : Returns the current user agent rotation angle in degrees. The initial value for currentRotate is 0.

On write : Sets the current user agent rotate coefficient in degrees.

currentTranslate

Current user agent translation used for scrolling or panning (The returned [SVGPoint](#) object is "live" and setting its x and y components will change the user agent's translation). The initial values for [currentTranslate](#) is [SVGPoint\(0,0\)](#).

viewport

The position and size of the viewport (implicit or explicit) that corresponds to this ['svg'](#) element. When the user agent is actually rendering the content, then the position and size values represent the actual values when rendering.

If this SVG document is embedded as part of another document (e.g., via the HTML 'object' element), then the position and size are unitless values in the coordinate system of the parent document. (If the parent uses CSS or XSL layout, then unitless values represent pixel units for the current CSS or XSL viewport, as described in the [CSS2](#) specification.) If the parent element does not have a coordinate system, then the user agent should provide reasonable default values for this attribute.

For stand-alone SVG documents, both ['x'](#) and ['y'](#) must be zero, the ['width'](#) must be the width of the viewport which the host environment provides to the SVG user agent into which it can render its content, and the ['height'](#) must be the height of the viewport, with all values expressed in the pixel coordinate system from the host environment, preferably such that this pixel coordinate system matches the same pixel coordinate system presented to HTML and matches the model for pixel coordinates described in the [CSS2](#) specification.

Note that "pixel coordinate systems" are host-specific, two possible approaches that hosts might use for pixel coordinate systems are actual device pixels or (particularly for high-resolution devices) pseudo device pixels which either exactly match SVG and CSS's "px" coordinates or utilize a similar approach.

The object itself and its contents are both readonly. A [DOMException](#) with error code NO_MODIFICATION_ALLOWED_ERR is raised if attempt is made to modify it. The returned [SVGRect](#) object is "live", i.e. its [x](#), [y](#), [width](#), [height](#) is automatically updated if viewport size or position changes.

currentTime

On read : Returns the current animation timeline time in seconds.

On write : Sets the current animation timeline time (in seconds). This API is required to support seeking forwards and backwards in the timeline. After a seek, animation continues to play (forwards) from the new time.

Methods

createSVGMatrixComponents

Creates a new [SVGMatrix](#) object. This object can be used to modify the value of traits which are compatible with the [SVGMatrix](#) type using the [setMatrixTrait](#) method. The internal representation of the matrix is as follows:

```
[ a c e ]  
[ b d f ]  
[ 0 0 1 ]
```

Parameters

in float **a** The 'a' component of the matrix to be set.

in float **b** The 'b' component of the matrix to be set.

in float **c** The 'c' component of the matrix to be set.

in float **d** The 'd' component of the matrix to be set.

in float **e** The 'e' component of the matrix to be set.

in float **f** The 'f' component of the matrix to be set.

Return value

[SVGMatrix](#) The created [SVGMatrix](#) object.

No Exceptions

createSVGRect

Creates a new [SVGRect](#) object. This object can be used to modify the value of traits which are compatible with the [SVGRect](#) type using the [setRectTrait](#) method. The initial values for **x**, **y**, **width**, **height** of this new [SVGRect](#) are zero.

No Parameters

Return value

[SVGRect](#) The created [SVGRect](#).

No Exceptions

createSVGPath

Creates a new [SVGPath](#) object. This object can be used to modify the value of traits which are compatible with the [SVGPath](#) type using the [setPathTrait](#) method.

No Parameters

Return value

[SVGPath](#) The created [SVGPath](#).

No Exceptions

createSVGRGBColor

Creates a new [SVGRGBColor](#) object. This object can be used to modify the value of traits which are compatible with the [SVGRGBColor](#) type using the [setRGBColorTrait](#) method.

Parameters

in long **red** The red component of the [SVGRGBColor](#).

in long **green** The green component of the [SVGRGBColor](#).

in long **blue** The blue component of the [SVGRGBColor](#).

Return value

[SVGRGBColor](#) The created [SVGRGBColor](#).

Exceptions

[SVGException](#) **SVG_INVALID_VALUE_ERR**: Raised if any of the parameters are not in the 0 to 255 range.

moveFocus

Moves current focus to a different object based on the value of `motionType`. The User Agent must take into account the currently focused object in the document in order to find the new focused object.

If this method succeeds :

- A [DOMFocusOut](#) event must be dispatched which has the previously focused object as the event target.
- After that, a [DOMFocusIn](#) event must be dispatched which has the the new focused object as the event target.

A reference to the new focused object can be obtained using the [EventTarget](#) interface of the generated [DOMFocusIn](#) event.

Refer to the [navigation section](#) to see how navigation is managed. The behavior for this method must be the same as if an equivalent move was done by the end user (using joystick or TAB keys) and not by scripting.

Whenever the method fails (i.e. a [DOMException](#) is raised), focus must stay on the currently focused object and no [DOMFocusOut/DOMFocusIn](#) event is dispatched.

Note: For stand-alone SVG documents, the User Agent must always have a currently focused object. At the beginning, the [SVGDocument](#) has focus.

Parameters

in short **motionType** The type of motion.

No Return value

Exceptions

[DOMException](#) **NOT_SUPPORTED_ERROR**: Raised if the requested motion type is not supported (i.e. not one of the interface constants).

[DOMException](#) `INVALID_ACCESS_ERR`: Raised if the currently focused object doesn't have a `nav-*` value corresponding to the requested motion type. For instance, if a `moveFocus(NAV_UP)` is called on an element which has no `'nav-up'` attribute.

[DOMException](#) `INVALID_STATE_ERR`: Raised if the currently focused object has a `nav-*` value corresponding to the requested motion type but the target indicated in this attribute can not be found or is not a focusable object. For instance, if a `moveFocus(NAV_UP)` is called on an object which has a `'nav-up'` attribute but the value of this attribute references an element which is not focusable.

setFocus

A request to put the focus on the given object.

If this method succeeds:

- A [DOMFocusOut](#) event must be dispatched which has the previously focused object as the event target.
- After that, a [DOMFocusIn](#) event must be dispatched which has the the new focused object as the event target.

A reference to the new focused object can be obtained using the [EventTarget](#) interface of the generated [DOMFocusIn](#) event.

Whenever the method fails (i.e. a [DOMException](#) is raised), focus must stay on the currently focused object and no [DOMFocusOut](#)/[DOMFocusIn](#) event is dispatched.

Note: For stand-alone SVG documents, the User Agent must always have a currently focused object. At the beginning, the [SVGDocument](#) has focus.

Parameters

in [DOMObject](#) object The object which should receive focus.

No Return value

Exceptions

[DOMException](#) `NOT_SUPPORTED_ERROR`: Raised if the requested element is not focusable (i.e. its `'focusable'` attribute does not evaluate to `true`).

getCurrentFocusedObject

Returns a reference to the object which has the focus. This returns a [DOMObject](#) because the currently focused object may be an object outside the SVG namespace. If the currently focused object is in the SVG namespace, it should be cast to `events::EventTarget`.

In the case of standalone SVG documents, this method should never return `null` because for standalone SVG documents a User Agent must always have a currently focused object.

No Parameters

Return value

[DOMObject](#) The currently focused object.

No Exceptions

A.7.5 SVGRGBColor

This interface represents an [SVGRGBColor](#) datatype made up of red, green, and blue components. It can be used to read properties that store color values ([getRGBColorTrait](#)) such as `'fill'`, `'stroke'`, and `'color'`.

IDL Definition

```
interface SVGRGBColor
{
    readonly attribute unsigned long red;
    readonly attribute unsigned long green;
    readonly attribute unsigned long blue;
};
```

No defined constants

Attributes

- red** Returns the red component of the [SVGRGBColor](#).
- green** Returns the green component of the [SVGRGBColor](#).
- blue** Returns the blue component of the [SVGRGBColor](#).

No defined methods

A.7.6 SVGRect

This interface represents an [SVGRect](#) datatype, consisting of a minimum X, minimum Y, width and height values.

IDL Definition

```
interface SVGRect
{
    attribute float x;
    attribute float y;
    attribute float width;
    attribute float height;
};
```

No defined constants

Attributes

x

On read : Returns the minimum X value for this [SVGRect](#).

On write : Sets the minimum X value of this [SVGRect](#) to the specified value.

y

On read : Returns the minimum Y value for this [SVGRect](#).

On write : Sets the minimum Y value of this [SVGRect](#) to the specified value.

width

On read : Returns the width for this [SVGRect](#).

On write : Sets the width of this [SVGRect](#) to the specified value.

height

On read : Returns the height for this [SVGRect](#).

On write : Sets the height of this [SVGRect](#) to the specified value.

No defined methods

A.7.7 SVGPoint

This interface represents an [SVGPoint](#) datatype, identified by its x and y components.

IDL Definition

```
interface SVGPoint
{
    attribute float x;
    attribute float y;
};
```

No defined constants

Attributes

x

On read : Returns the x component of the [SVGPoint](#).

On write : Sets the x component of the [SVGPoint](#) to the specified float value.

y

On read : Returns the y component of the [SVGPoint](#).

On write : Sets the y component of the [SVGPoint](#) to the specified float value.

No defined methods

A.7.8 SVGPath

This interface represents an [SVGPath](#) datatype used to define the path geometry.

Path data passing through this interface must be normalized as per the rules given in [Path Normalization](#).

IDL Definition

```
interface SVGPath
{
    const unsigned short MOVE_TO = 77;
    const unsigned short LINE_TO = 76;
    const unsigned short CURVE_TO = 67;
    const unsigned short QUAD_TO = 81;
    const unsigned short CLOSE = 90;
    readonly attribute unsigned long numberOfSegments;
    unsigned short getSegment(in unsigned long cmdIndex) raises(DOMException);
    float getSegmentParam(in unsigned long cmdIndex, in unsigned long paramIndex) raises(DOMException);
    void moveTo(in float x, in float y);
    void lineTo(in float x, in float y);
};
```

```
void quadTo(in float x1, in float y1, in float x2, in float y2);
void curveTo(in float x1, in float y1, in float x2, in float y2, in float x3, in float y3);
void close();
};
```

Constants

- MOVE_TO**
Numeric value is ASCII code of the letter 'M'.
- LINE_TO**
Numeric value is ASCII code of the letter 'L'.
- CURVE_TO**
Numeric value is ASCII code of the letter 'C'.
- QUAD_TO**
Numeric value is ASCII code of the letter 'Q'.
- CLOSE**
Numeric value is ASCII code of the letter 'Z'.

Attributes

- numberOfSegments**
Return number of segments in this path.

Methods

getSegment

Returns segment command by zero-based command index.

Parameters

in unsigned long **cmdIndex** The command index for the segment command to retrieve.

Return value

unsigned short The segment command. One of [MOVE_TO](#), [LINE_TO](#), [CURVE_TO](#), [QUAD_TO](#) or [CLOSE](#)

Exceptions

[DOMException](#) **INDEX_SIZE_ERR**: Raised if segment index is out of bounds or param index is out of bounds for this segment's type.

getSegmentParam

Returns segment parameter by zero-based command index and zero-based parameter index.

Parameters

in float **cmdIndex** The command index for the segment command.

in float **paramIndex** The parameter index to retrieve.

Return value

float The segment parameter.

Exceptions

[DOMException](#) **INDEX_SIZE_ERR**: Raised if segment index out of bounds or param index out of bounds for this segment's type.

moveTo

Appends an 'M' (absolute move) segment to the path with the specified coordinates.

Parameters

in float **x** The x-axis coordinate for the specified point.

in float **y** The y-axis coordinate for the specified point.

No Return value

No Exceptions

lineTo

Appends an 'L' (absolute line) segment to the path with the specified coordinates.

Parameters

in float **x** The x-axis coordinate for the specified point.

in float **y** The y-axis coordinate for the specified point.

No Return value

No Exceptions

quadTo

Appends a 'Q' (absolute quadratic curve) segment to the path.

Parameters

in float **x1** The x-axis coordinate of the first control point.

in float **y1** The y-axis coordinate of the first control point.

in float **x2** The x-axis coordinate of the final end point.

in float **y2** The y-axis coordinate of the final end point.

No Return value

No Exceptions

curveTo

Appends a 'C' (absolute cubic curve) segment to the path.

Parameters

in float **x1** The x-axis coordinate of the first control point.

in float **y1** The y-axis coordinate of the first control point.

in float **x2** The x-axis coordinate of the second end point.

in float **y2** The y-axis coordinate of the second end point.

in float **x3** The x-axis coordinate of the final end point.

in float `y3` The y-axis coordinate of the final end point.

No Return value
No Exceptions

close

Numeric value is ASCII code of the letter 'Z'.

A.7.9 SVGMatrix

This interface represents an [SVGMatrix](#) datatype, identified by an affine transform. It can be used to read and modify the values of the `'transform'` attribute.

Note: The `mTranslate`, `inverse`, `mMultiply`, `mScale` and `mRotate` methods in this interface mutate the [SVGMatrix](#) object and return a reference to the [SVGMatrix](#) instance itself, after performing the necessary matrix operation.

This matrix transforms source coordinates (x, y) into destination coordinates (x', y') by considering them to be a column vector and multiplying the coordinate vector by the matrix according to the following process:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a.x + c.y + e \\ b.x + d.y + f \\ 1 \end{bmatrix}$$

IDL Definition

```
interface SVGMatrix
{
    float getComponent(in unsigned long index) raises(DOMException);
    SVGMatrix mMultiply(in SVGMatrix secondMatrix);
    SVGMatrix inverse() raises(SVGException);
    SVGMatrix mTranslate(in float x, in float y);
    SVGMatrix mScale(in float scaleFactor);
    SVGMatrix mRotate(in float angle);
};
```

No defined constants

No defined attributes

Methods

getComponent

Returns a component of the matrix by the component's zero-based index. `getComponent(0)` is a, `getComponent(1)` is b, etc.

Parameters

in unsigned long `index` The index of the matrix component to retrieve.

Return value

float The matrix component.

Exceptions

[DOMException](#) `INDEX_SIZE_ERR`: Raised if the `index` is invalid.

mMultiply

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

Parameters

in [SVGMatrix](#) `secondMatrix` The matrix to post-multiply with.

Return value

[SVGMatrix](#) The resulting current matrix.

No Exceptions

inverse

Returns a new instance of [SVGMatrix](#) containing the inverse of the current matrix.

No Parameters

Return value

[SVGMatrix](#) The inverse of the current matrix.

Exceptions

[SVGException](#) `SVG_MATRIX_NOT_INVERTABLE`: Raised when determinant of this matrix is zero.

mTranslate

Post-multiplies a translation transformation on the current matrix and returns the resulting current matrix. This is equivalent to calling `multiply(T)`, where `T` is an [SVGMatrix](#) object represented by the following matrix:

$$\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

Parameters

in float `x` The distance by which coordinates are translated in the X axis direction.

in float `y` The distance by which coordinates are translated in the Y axis direction.

Return value

[SVGMatrix](#) The resulting current matrix.

No Exceptions

mScale

Post-multiplies a uniform scale transformation on the current matrix and returns the resulting current matrix. This is equivalent to calling `multiply(S)`, where `S` is an [SVGMatrix](#) object represented by the following matrix:

```
[ scaleFactor  0      0 ]
[  0      scaleFactor  0 ]
[  0      0      1 ]
```

Parameters

in float `scaleFactor` The factor by which coordinates are scaled along the X and Y axis.

Return value

[SVGMatrix](#) The resulting current matrix.

No Exceptions

mRotate

Post-multiplies a rotation transformation on the current matrix and returns the resulting current matrix. This is equivalent to calling `multiply(R)`, where `R` is an [SVGMatrix](#) object represented by the following matrix:

```
[ cos(angle) -sin(angle) 0 ]
[ sin(angle)  cos(angle) 0 ]
[  0      0      1 ]
```

Parameters

in float `angle` The angle of rotation in degrees.

Return value

[SVGMatrix](#) The resulting current matrix.

No Exceptions

A.7.10 SVGLocatable

Interface for getting information about the location of elements. **Note:** Animations will have an effect on the values of bounding box.

IDL Definition

```
interface SVGLocatable
{
    SVGRect  getBBox();
    SVGMatrix getScreenCTM();
    SVGRect  getScreenBBox();
};
```

No defined constants

No defined attributes

Methods

getBBox

Returns the tight bounding box in current user coordinate space (i.e. after application of the `'transform'` attribute or the `viewbox`). Tight bounding box is the smallest possible rectangle that includes the geometry of all contained graphics elements excluding stroke. The calculation is done in the user coordinate space of the element. When the bounding box is calculated elements with the `'display'` property (trait) set to `none` are ignored.

No Parameters

Return value

[SVGRect](#) The tight bounding box in current user coordinate space. The returned object is a copy of the current bounding box value and will not change if the corresponding bounding box changes.

No Exceptions

getScreenCTM

Returns the transformation matrix from current user units to the [initial viewport](#) coordinate system. DOM [MouseEvent](#) `clientX` and `clientY` coordinates are in the initial viewport coordinate system. Note that null is returned if this element is not hooked into the document tree. This method would have been more aptly named as `getClientCTM`, but the name `getScreenCTM` is kept for historical reasons.

No Parameters

Return value

[SVGMatrix](#) The transformation matrix. The returned object is a copy of the current screen CTM value and will not change if the corresponding screen CTM changes.

No Exceptions

getScreenBBox

Returns the tight bounding box in screen coordinate space. Tight bounding box is the smallest possible rectangle that includes the geometry of all contained graphics elements excluding stroke. The box coordinates are in the [initial viewport](#) coordinate system, which is connected to the current user coordinate space by the matrix returned by the `SVGLocatable::getScreenCTM` method.

Note: null is returned if this element is not hooked into the document tree.

No Parameters

Return value

[SVGRect](#) The tight bounding box in screen coordinate space. The returned object is a copy of the current screen bounding box value and will not change if the corresponding screen bounding box changes.

No Exceptions

The following examples further clarify the behavior of the [getBBox\(\)](#) method. The examples have a short explanation, an SVG fragment and are followed by a set of bounding box values which have the following format:

```
[elementId] : {x, y, width, height} | {null}
```

where x, y, width and height define the values of the [SVGRect](#) object's returned from a [getBBox](#) call on the element with the specified id. There are a few cases where the bounding box may be null (see example 6).

Example #1: Simple groups and bounds

This first example shows the values returned by the [getBBox](#) method for various simple basic shapes and groups. In particular, it shows that the transform, on an element, does not change the value of its user space bounding box.

```
<g id="group1" transform="translate(10, 20)" fill="red">
  <rect id="rect1" transform="scale(2)" x="10" y="10" width="50" height="50"/>
  <rect id="rect2" x="10" y="10" width="100" height="100"/>
  <g id="group2" transform="translate(10, 20)">
    <rect id="rect3" x="0" y="10" width="150" height="50"/>
    <circle id="circle1" cx="20" cy="20" r="100" />
  </g>
</g>
```

Result:

```
[group1] : {-70.0, -60.0, 230.0, 200.0}
[rect1] : {10.0, 10.0, 50.0, 50.0}
[rect2] : {10.0, 10.0, 100.0, 100.0}
[group2] : {-80.0, -80.0, 230.0, 200.0}
[rect3] : {0.0, 10.0, 150.0, 50.0}
[circle1] : {-80.0, -80.0, 200.0, 200.0}
```

Example #2: Bounding box on zero width or height rectangle

This example illustrates that the bounding box on elements is based on the element's geometry coordinates. For example, the bounding box on a zero-width rectangle is defined (see below), even though the rectangle is not rendered.

```
<g id="group1" transform="translate(10, 20)" fill="red">
  <rect id="rect2" x="10" y="10" width="400" height="0"/>
  <g id="group2" transform="translate(10, 20)">
    <rect id="rect3" x="0" y="10" width="150" height="50"/>
  </g>
</g>
```

Result:

```
[group1] : {10.0, 10.0, 400.0, 70.0}
[rect2] : {10.0, 10.0, 400.0, 0.0}
[group2] : {0.0, 10.0, 150.0, 50.0}
[rect3] : {0.0, 10.0, 150.0, 50.0}
```

Example #3: Bounding Box on zero radius ellipses.

This is another example of how bounding boxes are based on the element's geometry. Here, the bounding box of an ellipse with a zero x-axis radius is still defined, even though the ellipse is not rendered.

```
<svg id="mySVG" version="1.2" baseProfile="tiny" width="10" height="20">
  <g id="group1" transform="translate(10, 20)" fill="red">
    <rect id="rect1" x="10" y="10" width="100" height="100"/>
    <ellipse id="ellipse1" cx="20" cy="20" rx="0" ry="70" />
  </g>
</svg>
```

Result:

```
[mySVG] : {20.0, -30.0, 100.0, 160.0}
[group1] : {10.0, -50.0, 100.0, 160.0}
[rect1] : {10.0, 10.0, 100.0, 100.0}
[ellipse1] : {20.0, -50.0, 0.0, 140.0}
```

Example #4: Viewports do not clip bounding boxes

This example shows that no matter what the viewport is on the root SVG element, the bounding boxes, based on the geometry, are still defined. Here, even though the root svg has a zero width, the bounding boxes for the root itself and its children is precisely defined.

```
<svg id="mySVG" version="1.2" baseProfile="tiny" width="0" height="50">
  <g id="group1" transform="translate(10, 20)" fill="red" >
    <rect id="rect1" x="10" y="10" width="50" height="50"/>
  <g id="group2" transform="translate(10, 20)">
    <rect id="rect2" x="0" y="10" width="150" height="0"/>
    <circle id="circle1" cx="20" cy="20" r="500"/>
  </g>
</svg>
```

Result:

```
[mySVG] : {-460.0, -440.0, 1000.0, 1000.0}
[group1] : {-470.0, -460.0, 1000.0, 1000.0}
[rect1] : {10.0, 10.0, 50.0, 50.0}
[group2] : {-480.0, -480.0, 1000.0, 1000.0}
[rect2] : {0.0, 10.0, 150.0, 0.0}
[circle1] : {-480.0, -480.0, 1000.0, 1000.0}
```

Example #5: getBBox on <use>

This example shows that the bounding box for a **'use'** element accounts for the x and y attributes defined on the element, just like the x and y attributes impact the bounding box computation on a **'rect'** or on an **'image'** element.

```
<svg version="1.2" baseProfile="tiny">
  <defs>
    <rect id="myRect" x="0" y="0" width="60" height="40"/>
  </defs>
  <use id="myUse" xlink:href="#myRect" x="-30" y="-20"/>
</svg>
```

Result:

```
[myRect] : {0.0, 0.0, 60.0, 40.0}
[myUse] : {-30.0, -20.0, 60.0, 40.0}
```

Example #6: Empty group

This example shows that the bounding box for an empty group is null. By the same token, the bounding box of a **'path'** with an empty **SVGPath** (i.e. one with no path commands, which may happen after creating a new **'path'** element with a **createElementNS** call) is also null.

```
<g id="emptyG"/>
```

Result:

```
[emptyG] : {null}
```

Example #7: Impact of display='none' and visibility='hidden'

This example shows how the bounding box of children with **display='none'** are not accounted for in the computation of their parent's bounding box. This reflects the definition of the **'display'** property and its impact on rendering and bounding box computation. The example also shows that elements with a **'hidden'** **visibility'** still contribute to their parent's bounding box computation.

```
<g id="g1">
  <g id="g1.1.display.none" display="none">
    <rect id="rect1" x="10" y="10" width="40" height="40"/>
  </g>
  <rect id="rect2.visibility.hidden" visibility="hidden" x="30" y="60" width="10" height="20"/>
</g>
```

Result:

```
[g1] : {30.0, 60.0, 10.0, 20.0}
[g1.1.display.none] : {10.0, 10.0, 40.0, 40.0}
[rect1] : {10.0, 10.0, 40.0, 40.0}
[rect2.visibility.hidden] : {30.0, 60.0, 10.0, 20.0}
```

Example #8: Concatenating bounding boxes in the container's user space

This example shows how the concatenation and computation of bounding boxes for container element happens in the container's user space.

```
<g id="g1">
  <line id="line1" x2="100" y2="100" transform="rotate(-45)"/>
</g>
```

Result:

```
[g1] : {0.0, 0.0, 141.42136, 0}
[line1] : {0.0, 0.0, 100.0, 100.0}
```

Example #9: No influence of stroke-width

This example illustrates that stroking has no impact on the computation of bounding boxes.

```
<g>
  <line id="thickLine" stroke-width="10" x2="100" y2="0"/>
</g>
```

Result:

```
[thickLine] : {0.0, 0.0, 100.0, 0.0}
```

Example #10: No influence of viewBox

This example illustrates that the viewBox has no impact on the computation of bounding boxes.

```
<svg id="rootSvg" version="1.2" baseProfile="tiny" width="500" height="300" viewBox="0 0 200 100">
  <rect x="-100" y="-200" width="500" height="100"/>
</svg>
```

Result:

```
[rootSVG] : {-100, -200, 500, 100}
```

A.7.11 SVGLocatableElement

This interface represents an element that has a physical location on the screen.

This interface is implemented by: ['rect'](#), ['circle'](#), ['ellipse'](#), ['line'](#), ['path'](#), ['use'](#), ['image'](#), ['text'](#), ['textArea'](#), ['tspan'](#), ['svg'](#), ['a'](#), ['video'](#), ['animation'](#), ['switch'](#), ['foreignObject'](#), ['polygon'](#), ['polyline'](#) and ['g'](#).

IDL Definition

```
interface SVGLocatableElement : SVGElement, SVGLocatable
{
};
```

No defined constants**No defined attributes****No defined methods**

A.7.12 TraitAccess

Trait manipulation interface. This interface is used read and manipulate the value of "traits" associated with an [SVGElement](#). Each *trait* corresponds to an attribute or property, which is parsed and understood by the element and in most cases animatable. Unlike attributes, each element has a well-defined set of traits and attempting to access an undefined trait may throw an exception. Also unlike attributes traits are typed and their values are normalized; for instance SVG ['path'](#) specification is parsed and all path commands are converted to their absolute variants, it is not possible to say through the value of the trait if a path command was absolute or relative. When getting and setting trait values, accessor of the correct type must be used or exception will be thrown.

Initial trait values come from parsing corresponding attributes. If a value is not specified, but a corresponding attribute (or property for environments where styling is supported) is inherited, inherited value is returned as a result of the trait query method. If it is not inherited, default value is returned. Default values are also returned in the case when there is no parent to inherit from, for ex: when you create a new element, set a trait value to ['inherit'](#), but there is no parent for inheritance. The different [getTrait](#) methods ([getTrait](#), [getTraitNS](#), [getFloatTrait](#), ...) always returns a base value (i.e. before animation is applied), and this is true for both static and animated content. The different [getPresentationTrait](#) methods ([getPresentationTrait](#), [getPresentationTraitNS](#), [getFloatPresentationTrait](#), ...) return either the current animated value if the given trait is currently being animated (per the [SMIL specification](#)) or the base value if the given trait is not currently being animated. Not all attributes are accessible by traits, see the [table of supported attributes](#).

Setting a trait value has the same effect as changing a corresponding attribute, but trait setters can operate on typed values. The value which is modified is always a base value. For inheritable traits the trait value can always be set to ['inherit'](#) (but querying the value will always return the actual inherited value as explained above).

Note about invalid/unsupported trait values: There are two situations where the various trait setter methods (such as [setTrait](#), [setFloatTrait](#) or [setPathTrait](#) methods) consider a value invalid and throw a [DOMException](#) with the INVALID_ACCESS_ERR code.

The trait methods will consider the value to be invalid if its ['in error'](#).

The second situation is when the trait value is invalid with regards to animations currently applied to the trait. The value is considered invalid because it would put the animation, and therefore the document, in an error state. For example, if a `'path'` element has animations on its `'d'` attribute, trying to change the `'d'` attribute to a value incompatible with the animations will cause the exception to happen.

The trait setter methods will consider a value unsupported when it complies with the definition for an `'unsupported value'`. This will result in a `DOMException` thrown with the `NOT_SUPPORTED_ERR` code. For example, trying to set the `'stroke-linejoin'` trait to `'foo'` would cause this exception.

IDL Definition

```
interface TraitAccess
{
    DOMString getTrait(in DOMString name) raises(DOMException);
    DOMString getTraitNS(in DOMString namespaceURI, in DOMString name) raises(DOMException);
    float getFloatTrait(in DOMString name) raises(DOMException);
    SVGMatrix getMatrixTrait(in DOMString name) raises(DOMException);
    SVGRect getRectTrait(in DOMString name) raises(DOMException);
    SVGPath getPathTrait(in DOMString name) raises(DOMException);
    SVGRGBColor getRGBColorTrait(in DOMString name) raises(DOMException);
    DOMString getPresentationTrait(in DOMString name) raises(DOMException);
    DOMString getPresentationTraitNS(in DOMString namespaceURI, in DOMString name) raises(DOMException);
    float getFloatPresentationTrait(in DOMString name) raises(DOMException);
    SVGMatrix getMatrixPresentationTrait(in DOMString name) raises(DOMException);
    SVGRect getRectPresentationTrait(in DOMString name) raises(DOMException);
    SVGPath getPathPresentationTrait(in DOMString name) raises(DOMException);
    SVGRGBColor getRGBColorPresentationTrait(in DOMString name) raises(DOMException);
    void setTrait(in DOMString name, in DOMString value) raises(DOMException);
    void setTraitNS(in DOMString namespaceURI, in DOMString name, in DOMString value) raises(DOMException);
    void setFloatTrait(in DOMString name, in float value) raises(DOMException);
    void setMatrixTrait(in DOMString name, in SVGMatrix matrix) raises(DOMException);
    void setRectTrait(in DOMString name, in SVGRect rect) raises(DOMException);
    void setPathTrait(in DOMString name, in SVGPath path) raises(DOMException);
    void setRGBColorTrait(in DOMString name, in SVGRGBColor color) raises(DOMException);
};
```

No defined constants

No defined attributes

Methods

`getTrait`

Returns the trait value (possibly [normalized](#)) as a [DOMString](#). In SVG Tiny only certain traits can be obtained as a [DOMString](#) value. Syntax of the returned [DOMString](#) matches the syntax of the corresponding attribute. This element is exactly equivalent to `getTraitNS` with `namespaceURI` set to `null`.

Parameters

in [DOMString](#) name The name of the trait to retrieve.

Return value

[DOMString](#) The trait value.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to a [DOMString](#) (SVG Tiny only).

`getTraitNS`

Same as `getTrait`, but for namespaced traits. Parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters

in [DOMString](#) namespaceURI The namespace of the trait to retrieve.

in [DOMString](#) name The name of the trait to retrieve.

Return value

[DOMString](#) The trait value.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to a [DOMString](#) (SVG Tiny only).

`getFloatTrait`

Get the trait value as a float.

Parameters

in [DOMString](#) name The name of the trait to retrieve.

Return value

float The trait value as a float.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to a float.

`getMatrixTrait`

Returns the trait value as an [SVGMatrix](#). The returned object is a copy of the actual trait value and will not change if the corresponding trait changes.

Parameters

in [DOMString](#) **name** The name of the trait to retrieve.

Return value

[SVGMatrix](#) The trait value as an [SVGMatrix](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGMatrix](#).

getRectTrait

Returns the trait value as an [SVGRect](#). The returned object is a copy of the actual trait value and will not change if the corresponding trait changes.

Parameters

in [DOMString](#) **name** The name of the trait to retrieve.

Return value

[SVGRect](#) The trait value as an [SVGRect](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGRect](#).

getPathTrait

Returns the trait value as an [SVGPath](#). The returned object is a copy of the actual trait value and will not change if the corresponding trait changes.

Parameters

in [DOMString](#) **name** The name of the trait to retrieve.

Return value

[SVGPath](#) The trait value as an [SVGPath](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGPath](#).

getRGBColorTrait

Returns the trait value as an [SVGRGBColor](#). The returned object is a copy of the trait value and will not change if the corresponding trait changes. If the actual trait value is not an [SVGRGBColor](#), i.e. 'none' or a link to a paint server (e.g. to a gradient or a solid-color), this method must raise a [DOMException](#) with error code TYPE_MISMATCH_ERR.

Parameters

in [DOMString](#) **name** The name of the trait to retrieve.

Return value

[SVGRGBColor](#) The trait value as an [SVGRGBColor](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGRGBColor](#).

getPresentationTrait

Returns the trait presentation value as a [DOMString](#). In SVG Tiny only certain traits can be obtained as a [DOMString](#) value. Syntax of the returned [DOMString](#) matches the syntax of the corresponding attribute. This element is exactly equivalent to [getPresentationTraitNS](#) with `namespaceURI` set to `null`.

Parameters

in [DOMString](#) **name** The name of the trait to retrieve.

Return value

[DOMString](#) The trait presentation value.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to a [DOMString](#) (SVG Tiny only).

getPresentationTraitNS

Same as [getPresentationTrait](#), but for namespaced traits. The parameter `name` must be a non-qualified trait name, i.e. without prefix.

Parameters

in [DOMString](#) **namespaceURI** The namespace of the trait to retrieve.

in [DOMString](#) **name** The name of the trait to retrieve.

Return value

[DOMString](#) The trait presentation value.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to a [DOMString](#) (SVG Tiny only).

getFloatPresentationTrait

Get the trait presentation value as a `float`.

Parameters

in [DOMString](#) `name` The name of the trait to retrieve.

Return value

`float` The trait presentation value as a `float`.

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to a `float`.

getMatrixPresentationTrait

Returns the trait presentation value as an [SVGMatrix](#). The returned object is a copy of the actual trait value and will not change if the corresponding trait changes or as animation continue to affect the trait presentation value.

Parameters

in [DOMString](#) `name` The name of the trait to retrieve.

Return value

[SVGMatrix](#) The trait presentation value as an [SVGMatrix](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGMatrix](#).

getRectPresentationTrait

Returns the trait presentation value as an [SVGRect](#). The returned object is a copy of the actual trait value and will not change if the corresponding trait changes or as animation continue to affect the trait presentation value.

Parameters

in [DOMString](#) `name` The name of the trait to retrieve.

Return value

[SVGRect](#) The trait presentation value as an [SVGRect](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGRect](#).

getPathPresentationTrait

Returns the trait presentation value as an [SVGPath](#). The returned object is a copy of the actual trait value and will not change if the corresponding trait changes or as animation continue to affect the trait presentation value.

Parameters

in [DOMString](#) `name` The name of the trait to retrieve.

Return value

[SVGPath](#) The trait presentation value as an [SVGPath](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGPath](#).

getRGBColorPresentationTrait

Returns the trait presentation value as an [SVGRGBColor](#). The returned object is a copy of the trait value and will not change if the corresponding trait changes or as animation continue to affect the trait presentation value. If the actual trait presentation value is not an `RGBColor` (i.e. "none"), this method will return `null`.

Parameters

in [DOMString](#) `name` The name of the trait to retrieve.

Return value

[SVGRGBColor](#) The trait presentation value as an [SVGRGBColor](#).

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or `null`.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if requested trait's computed value cannot be converted to an [SVGRGBColor](#).

setTrait

Set the trait value as a [DOMString](#). In SVG Tiny only certain traits can be set through a [DOMString](#) value. The syntax of the [DOMString](#) that should be given as a value must be the same as syntax of the corresponding XML attribute value. Exactly equivalent to [setTraitNS](#) with the `namespaceURI` attribute set to `null`.

Parameters

in [DOMString](#) **name** The name of the trait to be set.

in [DOMString](#) **value** The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or an attempt has been made to set an unsupported value.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as a [DOMString](#).

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is an invalid value for the given trait or `null` is specified.

[DOMException](#) NO_MODIFICATION_ALLOWED_ERR: Raised if attempt is made to change a readonly trait.

setTraitNS

Same as [setTrait](#), but for namespaced traits. The parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters

in [DOMString](#) **namespaceURI** The namespace of the trait to be set.

in [DOMString](#) **name** The name of the trait to be set.

in [DOMString](#) **value** The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element or an attempt has been made to set an unsupported value.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as a [DOMString](#).

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is an invalid value for the given trait or `null` is specified. This error is also thrown when the **'use'** element is hooked into the document tree and the the value of **'xlink:href'** is set invalid.

[DOMException](#) NO_MODIFICATION_ALLOWED_ERR: Raised if attempt is made to change a readonly trait.

setFloatTrait

Set the trait value as a `float`.

Parameters

in [DOMString](#) **name** The name of the trait to be set.

in `float` **value** The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as a `float`.

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is an invalid value for the given trait or `null` is specified.

setMatrixTrait

Set the trait value as an [SVGMatrix](#). Values in [SVGMatrix](#) are copied in the trait so subsequent changes to the given [SVGMatrix](#) have no effect on the value of the trait.

Parameters

in [DOMString](#) **name** The name of the trait to be set.

in [SVGMatrix](#) **value** The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as an [SVGMatrix](#).

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is `null`.

setRectTrait

Set the trait value as an [SVGRect](#). Values in [SVGRect](#) are copied in the trait so subsequent changes to the given [SVGRect](#) have no effect on the value of the trait.

Parameters

in [DOMString](#) **name** The name of the trait to be set.

in [SVGRect](#) **value** The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as an [SVGRect](#).

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is an invalid value for the given trait or null is specified. An [SVGRect](#) is invalid if the width or height values are set to negative..

setPathTrait

Set the trait value as an [SVGPath](#). Values in [SVGPath](#) are copied in the trait so subsequent changes to the given [SVGPath](#) have no effect on the value of the trait.

Parameters

in [DOMString](#) name The name of the trait to be set.

in [SVGPath](#) value The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as an [SVGPath](#).

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is an invalid value for the given trait or null is specified. An [SVGPath](#) is invalid if it begins with any segment other than MOVE_TO segment.

Note: An empty [SVGPath](#) is still a valid value.

setRGBColorTrait

Set the trait value as an [SVGRGBColor](#). Values in [SVGRGBColor](#) are copied in the trait so subsequent changes to the given [SVGRGBColor](#) have no effect on the value of the trait.

Parameters

in [DOMString](#) name The name of the trait to be set.

in [SVGRGBColor](#) value The value of the trait.

No Return value

Exceptions

[DOMException](#) NOT_SUPPORTED_ERR: Raised if the requested trait is not supported on this element.

[DOMException](#) TYPE_MISMATCH_ERR: Raised if the requested trait's value cannot be specified as an [SVGRGBColor](#).

[DOMException](#) INVALID_ACCESS_ERR: Raised if the input value is null.

Traits supported in this specification, SVG Tiny 1.2 uDOM

Trait access is not required on all of the [SVG Tiny attributes and properties](#). The table below shows the attributes and properties that SVG Tiny DOM 1.2 implementations must support trait access to. Each attribute row lists the allowed getter and setter(s). The 'Default Values' column specifies the default values that must be used for each attribute or property. If a 'Default Values' column entry is empty, there is no default value. Unless explicitly stated in the 'Comments' column, a supported attribute is accessible on all elements it can belong to. See the [attribute section](#) for a list of attributes and which elements they belong to.

Implementations that support multiple versions of svg must allow trait access to the most extensive set and support the types supported by each trait in the most extensive set. However, content relying on traits or trait types available in future versions may not work in all conformant SVG Tiny 1.2 uDOM implementations.

For 'REQUIRED' attributes, there are two cases:

- i) The document is in error, if this attribute was not present at the time of loading.
- ii) When using the uDOM API, the specified default value (in parenthesis) must be used.

The User Agent must raise a NOT_SUPPORTED_ERR whenever there is an attempt to use trait methods for traits which are not supported by the UA.

Note: For some of the attributes and data types [additional rules](#) apply. These rules are defined below the table.

Note: In the table below:

- Wherever the table indicates that a user agent must support the [getTrait](#) method, it must also support the [getTraitNS](#), [getPresentationTrait](#), and [getPresentationTraitNS](#) methods.
- Wherever the table indicates that a user agent must support one of the typed (i.e., non-string) trait getter methods (e.g., [getFloatTrait](#)), it must also support the corresponding [get***PresentationTrait](#) method (e.g., [getFloatPresentationTrait](#)).
- Wherever the table indicates that a user agent must support the [setTrait](#) method, it must also support the [setTraitNS](#) method.
- Traits can only be used for accessing attribute and property values, traits *cannot* be used for accessing font descriptors.

Attribute	Trait Getter	Trait Setter	Default Values	Comments
accumulate	getTrait [none sum]	setTrait [none sum]	none	

additive	getTrait [replace sum]	setTrait [replace sum]	replace	
attributeName	getTrait	setTrait		
audio-level	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait [inherit]	1.0f	
baseProfile	getTrait	Not available (readonly)	"tiny"	
begin	N/A	setTrait		
calcMode	getTrait [discrete linear paced spline]	setTrait [discrete linear paced spline]	linear (except for animateMotion) paced (for animateMotion)	
color	getRGBColorTrait [SVGRGBColor] getTrait [SVGRGBColor]	setRGBColorTrait [SVGRGBColor] setTrait [inherit SVGRGBColor]	rgb(0,0,0)	-See RGB access rule .
cx	getFloatTrait	setFloatTrait	0.0f	
cy	getFloatTrait	setFloatTrait	0.0f	
d	getPathTrait [SVGPath]	setPathTrait [SVGPath]	REQUIRED(Empty SVGPath)	
display	getTrait [inline none]	setTrait [inline none inherit]	"inline"	-See Display access rule .
dur	N/A	setTrait		
editable	getTrait [true false]	setTrait [true false]	"false"	
fill	getRGBColorTrait [SVGRGBColor] getTrait [none SVGRGBColor]	setRGBColorTrait [SVGRGBColor] setTrait [none currentColor inherit SVGRGBColor]	rgb(0,0,0)	-See RGB access rule . -Attribute defining the fill-color.
fill	getTrait[freeze remove]	setTrait[freeze remove]	"remove"	-Attribute defining the fill behavior of a smil animation element.
fill-opacity	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait[inherit]	1.0f	
fill-rule	getTrait [nonzero evenodd]	setTrait [nonzero evenodd inherit]	"nonzero"	
focusable	getTrait [true false]	setTrait [true false auto]		
nav-right	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-next	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-up	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-up-right	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-up-left	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-prev	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	

nav-down	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-down-right	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-down-left	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
nav-left	getTrait [auto url (<Local IRI Reference>) self]	setTrait [url(<Local IRI Reference>) auto self]	"auto"	
focusHighlight	getTrait [auto none]	setTrait [auto none]	"auto"	
font-family	getTrait [the font-family computed value listed in the same syntax as the font-family property]	setTrait [same syntax as font-family attribute]	User-Agent	
font-size	getFloatTrait [value >= 0]	setFloatTrait [value >= 0] setTrait [inherit]	User-Agent	
font-style	getTrait [normal italic oblique]	setTrait [normal italic oblique inherit]	"normal"	
font-weight	getTrait [100 200 300 400 500 600 700 800 900]	setTrait [normal bold bolder lighter 100 200 300 400 500 600 700 800 900 inherit]	"normal"	
gradientUnits	getTrait [userSpaceOnUse objectBoundingBox]	setTrait [userSpaceOnUse objectBoundingBox]	"objectBoundingBox"	
height	getFloatTrait [value >= 0]	setFloatTrait [value >= 0]	REQUIRED(0.0f)	If the height is specified as a percentage or unit value (only possible for the ' svg ' ' height '), a DOMException with error code <code>TYPE_MISMATCH_ERR</code> is raised since the requested trait's computed value cannot be converted to a float.
id	getTrait	setTrait		
keyPoints	N/A	setTrait		
keySplines	N/A	setTrait		
keyTimes	N/A	setTrait		
max	N/A	setTrait		
min	N/A	setTrait		
offset	getFloatTrait[value >= 0 && value <= 1]	setFloatTrait[value >= 0 && value <= 1]		
opacity	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait[inherit]		
path	getPathTrait [SVGPath]	setPathTrait [SVGPath]	REQUIRED(Empty SVGPath)	
r	getFloatTrait [value >= 0]	setFloatTrait [value >= 0]	REQUIRED (0.0f)	
repeatCount	N/A	setTrait		
repeatDur	N/A	setTrait		

restart	getTrait [always whenNotActive never]	setTrait [always whenNotActive never]	always	
rx	getFloatTrait [value >= 0]	setFloatTrait [value >= 0]	0.0f	
ry	getFloatTrait [value >= 0]	setFloatTrait [value >= 0]	0.0f	
snapshotTime	getFloatTrait [value >= 0]	setFloatTrait [value >= 0]	0.0f	
solid-color	getRGBColorTrait [SVGRGBColor] getTrait [SVGRGBColor]	setRGBColorTrait [SVGRGBColor] setTrait [SVGRGBColor inherit]	rgb(0,0,0)	-See RGB access rule .
solid-opacity	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait [inherit]	1.0f	
stop-color	getRGBColorTrait [SVGRGBColor] getTrait [none SVGRGBColor]	setRGBColorTrait [SVGRGBColor] setTrait(none currentColor inherit SVGRGBColor]	rgb(0,0,0)	-See RGB access rule .
stop-opacity	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait[inherit]	1.0f	
stroke	getRGBColorTrait [SVGRGBColor] getTrait [none SVGRGBColor]	setRGBColorTrait [SVGRGBColor] setTrait [none currentColor inherit SVGRGBColor]	"none"	-See RGB access rule .
stroke-dashoffset	getFloatTrait	setTrait [inherit] setFloatTrait	0.0f	
stroke-linecap	getTrait [butt round square]	setTrait [butt round square inherit]	"butt"	
stroke-linejoin	getTrait [miter round bevel]	setTrait [miter round bevel inherit]	"miter"	
stroke-miterlimit	getFloatTrait [value >= 1]	setTrait [inherit] setFloatTrait [value >= 1]	4.0f	
stroke-opacity	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait[inherit]	1.0f	
stroke-width	getFloatTrait [value >= 0]	setTrait [inherit] setFloatTrait [value >= 0]	1.0f	
target	getTrait	setTrait	"_self"	
"text"	getTrait("#text") ["text content"]	setTrait("#text", "text content")		-See Text Node Access for details.
text-anchor	getTrait [start middle end]	setTrait [start middle end inherit]	"start"	
transform	getMatrixTrait [SVGMatrix] getTrait	setMatrixTrait [SVGMatrix] setTrait	Identity matrix (1,0,0,1,0,0)	-See Transform access rule .

type	For animateTransform: getTrait [translate scale rotate skewX skewY]	For animateTransform: setTrait [translate scale rotate skewX skewY]		
values	N/A	setTrait		
vector-effect	getTrait [none non-scaling-stroke]	setTrait [none non-scaling-stroke inherit]	"none"	
version	getTrait	Not available (readonly)	"1.2"	
viewBox	getRectTrait [null, SVGRect]	setRectTrait [SVGRect] setTrait [none]	null	If the actual trait value is not an SVGRect (i.e. "none"), the getRectTrait method will return null.
viewport-fill	getRGBColorTrait [SVGRGBColor] getTrait [none SVGRGBColor]	setRGBColorTrait [SVGRGBColor] setTrait [none currentColor SVGRGBColor inherit]	"none"	-See RGB access rule .
viewport-fill-opacity	getFloatTrait [value >= 0 && value <= 1]	setFloatTrait [value >= 0 && value <= 1] setTrait [inherit]	1.0f	
visibility	getTrait [visible hidden]	setTrait [visible hidden inherit]	"visible"	
width	getFloatTrait [value >= 0]	setFloatTrait [value >= 0]	REQUIRED (0.0f)	If the width is specified as a percentage or unit value (only possible for the ' svg ' ' width '), a DOMException with error code TYPE_MISMATCH_ERR is raised since the requested trait's computed value cannot be converted to a float.
x	getFloatTrait (array of x-values not supported)	setFloatTrait (array of x-values not supported)	0.0f	
x1	getFloatTrait	setFloatTrait	0.0f	
x2	getFloatTrait	setFloatTrait	0.0f	
xlink:href	getTraitNS [absolute IRI or ""]	setTraitNS	"".	
y	getFloatTrait (array of y-values not supported)	setFloatTrait (array of y-values not supported)	0.0f	
y1	getFloatTrait	setFloatTrait	0.0f	
y2	getFloatTrait	setFloatTrait	0.0f	
zoomAndPan	getTrait [disable magnify]	setTrait [disable magnify]	"magnify"	

A.7.13 Additional accessing rules

Accessing rules for RGBColorTrait

The [getRGBColorTrait](#) is used to get the [SVGRGBColor](#) color. If the actual trait value is not an [SVGRGBColor](#), i.e. 'none' or a link to a paint server (e.g. to a gradient or a solid-color), this method must raise a [DOMException](#) with error code TYPE_MISMATCH_ERR and [getTrait](#) should be used instead. [setTrait](#) must be used to set a color type that is not an [SVGRGBColor](#).

Accessing rules for 'transform' attribute

The '[transform](#)' attribute in SVGT1.2 can have three types of values. The "normal" transformation list (e.g. scale, translate, rotate, matrix etc), the newly introduced '[ref\(svg\)](#)' type or 'none'. [getMatrixTrait](#) returns the current evaluated matrix in all cases. If the user needs to know that the '[transform](#)' attribute value was a '[ref](#)' or a '[none](#)' he must call

`getTrait`. By using `setTrait` he can set the `'transform'` attribute to `'ref(svg)'` or `'none'`.

Accessing rules for `'display'` property

Due to backward compatibility reasons the `'display'` values accessible via the trait mechanism are limited to `'none'` and `'inline'`, all other values are translated into `'none'` or `'inline'`. (For a list of all possible `'display'` values, see [Controlling visibility](#).) If other `'display'` values are of interest, e.g. the user want to set display to `'block'`, the more generic `getAttributeNS/setAttributeNS` must be used. Note however that an SVGT1.2 user agent is allowed to normalize its attribute data as described in [Display Normalization](#).

Accessing rules for animation and font related elements

The following rule applies to Smil-animation elements (`'animate'`, `'animateTransform'`, `'animateColor'`, `'animateMotion'`, `'set'`) and font element with its children (`'font'`, `'font-face'`, `'missing-glyph'`, `'glyph'`, `'hkern'`, `'font-face-src'`, `'font-face-uri'`, `'font-face-name'`).

These elements can be inserted and removed from the tree but they cannot be modified once inserted into the tree. Manipulating animations while they are in the uDOM tree and possibly active would result in undefined behaviour. This may also have an effect on past resolved times. This restriction means that if the author wishes to add animations via script, the element must be modified when it is not attached to the tree. A similar reasoning applies to the different font elements, modifying them while attached to the tree might lead to unpredictable result. Following is an example of adding animation to the tree including setting the properties prior to insertion.

Example: Animating via the uDOM

```
<svg version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg" id="svg-root"
width="100%" height="100%" viewBox="0 0 480 360">
<rect id="myRect" fill="green" x="10" y="10" width="200" height="100" stroke="black" stroke-width="2"/>
</svg>
```

A script (java) such as the following might be used to add an animation to the rectangle:

```
SVGElement newAnimate = (SVGElement)document.createElementNS(svgNS, "animate");
newAnimate.setTrait("attributeName", "fill");
newAnimate.setTrait("from", "red");
newAnimate.setTrait("to", "blue");
newAnimate.setTrait("dur", "5");
newAnimate.setTrait("repeatCount", "10");
Element myRect = document.getElementById("myRect");
myRect.appendChild(newAnimate);
```

A.7.14 ElementTraversal

This interface provides a way to traverse elements in the uDOM tree. It is needed mainly because SVG Tiny uDOM does not expose character data nodes. Each element in the SVG Tiny document tree implements this interface. This applies to elements in the foreign namespaces as well.

IDL Definition

```
interface ElementTraversal
{
    readonly attribute Element firstElementChild;
    readonly attribute Element lastElementChild;
    readonly attribute Element nextElementSibling;
    readonly attribute Element previousElementSibling;
};
```

No defined constants

Attributes

firstElementChild

Returns the first child element node of this element. null if this element has no child elements.

lastElementChild

last child element node of this element. null if this element has no child elements.

nextElementSibling

Returns the next sibling element node of this element. null if this element has no element sibling nodes that come after this one in the document tree.

previousElementSibling

previous sibling element node of this element. null if this element has no element sibling nodes that come before this one in the document tree.

No defined methods

A.7.15 SVGElement

This interface represents an SVG element in the document tree. It provides methods to traverse elements in the uDOM tree and allows setting and getting the `id` of an element.

Note: See the [definition of id and xml:id](#) for the rules of how to treat `id` and `xml:id`.

IDL Definition

```
interface SVGElement : Element, EventTarget, TraitAccess, ElementTraversal
{
    attribute DOMString id;
};
```

No defined constants

Attributes

id

On read : Returns the Element's id, null if no id specified.

On write : Sets the Element's id attribute. See the description of how to handle multiple ID attribute in the [Structure chapter](#).

No defined methods

A.7.16 SVGAnimationElement

This interface represents an [SVGAnimationElement](#) which is implemented by [Timed Elements](#) and the [svg](#) element.

IDL Definition

```
interface SVGAnimationElement : SVGElement, smil::ElementTimeControl
{
};
```

No defined constants

No defined attributes

No defined methods

A.7.17 SVGVisualMediaElement

This interface represents a media element that is visual, i.e. has a physical location on the screen.

This interface is implemented by: ['animation'](#) and ['video'](#).

IDL Definition

```
interface SVGVisualMediaElement : SVGLocatableElement, SVGAnimationElement
{
};
```

No defined constants

No defined attributes

No defined methods

A.7.18 EventListenerInitializer2

[EventListenerInitializer2](#) allows event listeners to be initialized. In typical usage with Java, a ['script'](#) element references a JAR file which contains a manifest entry (SVG-Handler-Class) which identifies the class responsible for creating the event listeners. Further information about [EventListenerInitializer2](#) can be found in the [Scripting chapter](#).

Note: Script code must not invoke the [EventListenerInitializer2](#) methods directly (they are only called by the User Agent implementation during the parsing or execution of ['script'](#), ['listener'](#) or ['handler'](#) elements). User Agents must prevent user scripts from invoking the methods.

IDL Definition

```
interface EventListenerInitializer2
{
    void initializeEventListeners( in Element scriptElement );
    EventListener createEventListener( in Element handlerElement );
};
```


No defined constants

No defined attributes

Methods

[initializeEventListeners](#)

For each **'script'** element, at load time for that element, the user agent finds the appropriate object which implements the [EventListenerInitializer2](#) interface. (For Java, it is the class identified by the SVG-Handler-Class manifest entry). The user agent then invokes the [initializeEventListeners](#) method, which allows the scripting code to register event listeners.

Parameters

in [Element](#) `scriptElement` The **'script'** element which has been loaded.

No Return value

No Exceptions

[createEventListener](#)

For each `<handler>` element, at load time for that element, the user agent finds the appropriate object which implements the [EventListenerInitializer2](#) interface. (For Java, it is the class identified by the SVG-Handler-Class manifest entry). The user agent then invokes the [createEventListener](#) method, which allows the scripting code to register an appropriate event listener.

This method returns the event listener that will handle events. The user agent must register this event listener with the event target identified by the **'handler'** element.

Parameters

in [Element](#) `handlerElement` The **'handler'** element which has been loaded.

Return value

[EventListener](#) The created [EventListener](#).

No Exceptions

Example #1: The usage of java together with the **'script'** element

The example rely on the fact the 'myclasses.jar' JAR file contains a MANIFEST file with the following entry:

```
SVG-Handler-Class: org.example.SVGHandler
```

Given the following SVG document:

```
<svg xmlns:svg="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 0 500 500">
<script type="application/java-archive" xlink:href="myclasses.jar"/>
<rect id="therect" x="0" y="0" width="100" height="100"/>
</svg>
```

Given that the SVGHandler implementation available in the 'myclasses.jar' JAR file is the following:

```
package org.example;

import org.w3c.dom.*;
import org.w3c.dom.svg.*;

public class SVGHandler implements EventListenerInitializer2 {
    public void initializeEventListeners (Element scriptElement) {
        Document document = scriptElement.getOwnerDocument();
        EventTarget rect = (EventTarget) document.getElementById("therect");
        rect.addEventListenerNS("http://www.w3.org/2001/xml-events", "click", new MyListener(), null, false, null);
    }

    public EventListener createEventListener (Element handlerElement) {}
}
```

An instance of "MyListener" listener will be called when a **'click'** event will occur on the **'rect'** element.

Example #2: The usage of java together with the **'handler'** element

The example rely on the fact the 'myclasses.jar' JAR file contains a MANIFEST file with the following entry:

```
SVG-Handler-Class: org.example.SVGHandler
```

Given the following SVG document:

```
<svg xmlns:svg="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 0 500 500">
xmlns:ev="http://www.w3.org/2001/xml-events" xmlns:myns="http://example.org/myNS">
<script id="init" type="application/java-archive" xlink:href="myclasses.jar"/>
<rect id="therect" x="0" y="0" width="100" height="100">
<handler type="application/java" ev:event="click"
xlink:href="#init" myns:listenerClass="MyListener"/>
</rect>
</svg>
```

Given that the SVGHandler implementation available in the 'myclasses.jar' JAR file is the following:

```
package org.example;

import org.w3c.dom.*;
import org.w3c.dom.svg.*;

public class SVGHandler implements EventListenerInitializer2 {
    public void initializeEventListeners (Element scriptElement) {}
}
```

```

public EventListener createEventListener (Element handlerElement) {
    EventListener listenerInstance = null;
    try {
        String listenerClass = handlerElement.getAttributeNS("http://example.org/myNS", "listenerClass");
        listenerInstance = Class.forName(listenerClass).newInstance();
    }
    catch (Exception e) {}

    return listenerInstance;
}
}
}

```

An instance of "MyListener" listener will be called when a `'click'` event will occur on the `'rect'` element. What `createEventListener` does is totally in the hand of the developer, the listener instance can be hardcoded or fully configured from information put on the `'handler'` element as shown here.

A.7.19 SVGGlobal

The majority of scripted SVG documents in existence make use of the browser specific Window interface. SVG 1.2 specifies an `SVGGlobal` interface, taking into account the de-facto standard that already exists, as well as adding the new features present in SVG 1.2. `SVGGlobal` inherits from the `Global` interface, which is currently defined to be empty. The `Global` interface is designed to be the parent interface for language specific window objects. In scripting implementations, the methods and attributes defined by the `Global` object are normally part of the global execution context.

Interface `SVGGlobal` provides a global object for scripts embedded in an SVG document.

IDL Definition

```

interface SVGGlobal : Global, EventListenerInitializer2
{
    Connection createConnection();
    Timer createTimer();
    void gotoLocation(in DOMString newIRI);
    readonly attribute Document document;
    readonly attribute Global parent;
    DOMString binaryToString(in sequence<octet>, in DOMString encoding) raises(GlobalException);
    sequence<octet> stringToBinary(in DOMString data, in DOMString encoding) raises(GlobalException);
    DOMString getURL(in DOMString iri, in AsyncStatusCallback callback);
    DOMString postURL(in DOMString iri, in DOMString data, in AsyncStatusCallback callback, in DOMString type, in DOMString encoding);
    Node parseXML(in DOMString data, in Document contextDoc);
};

```

No defined constants

Attributes

document

The `Document` that this `SVGGlobal` operates on.

parent

The `Global` context of this document's parent. If the document has no notion of parent (e.g. when the document is displayed in a stand-alone viewer) the parent is `null`.

Methods

createConnection

Creates a `Connection` object.

No Parameters

Return value

`Connection` The created `Connection`.

No Exceptions

createTimer

Creates a new `Timer` instance.

Return value

`Timer` The created `Timer`.

No Exceptions

gotoLocation

Request that the user agent traverse a link to the given `IRI`. If the `IRI` is invalid or unresolvable, the current document remains loaded. "" means that no link traversal is specified. (See the attribute definition for `xlink:href`).

Parameters

in `DOMString newIRI` The `IRI` to be traversed.

No Return value

No Exceptions

binaryToString

Given a sequence of octets such as can be obtained through a `Connection` object and a character encoding identifier, this method returns the string produced by interpreting the sequence of octets using that character encoding. Both the UTF-8 and UTF-16 (BE and LE) encodings must be supported, implementations may support other encodings as well.

Parameters

in `sequence<octet> data` A character encoding identifier. The values "UTF-8", "UTF-16", "ISO-10646-UCS-2", and "ISO-10646-UCS-4" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646. It is recommended that character encodings registered (as charsets) with the Internet Assigned Numbers Authority, other than those just listed, be referred to using their registered names; other encodings should use names starting with an "x-" prefix.

in `DOMString encoding` A character encoding identifier.

Return value

`DOMString` The string resulting from decoding the sequence of octets.

Exceptions

`GlobalException` `ENCODING_ERR`: Raised if it is impossible to decode the given sequence of octets with the provided encoding (e.g. because of poorly formed data).

stringToBinary

Given a string and a character encoding identifier, this method returns a sequence of octets representing the string encoded using the specified encoding as can be used to transmit data using the `Connection` interface. Both the UTF-8 and UTF-16 (BE and LE) encodings must be supported, implementations may support other encodings as well.

Parameters

in `DOMString data` A string to be encoded using the specified encoding.

in `DOMString encoding` A character encoding identifier. The values "UTF-8", "UTF-16", "ISO-10646-UCS-2", and "ISO-10646-UCS-4" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646. It is recommended that character encodings registered (as charsets) with the Internet Assigned Numbers Authority, other than those just listed, be referred to using their registered names; other encodings should use names starting with an "x-" prefix.

Return value

`sequence<octet>` The sequence of octets resulting from encoding the string.

Exceptions

`GlobalException` `ENCODING_ERR`: Raised if it is impossible to encode the given string with the provided encoding (e.g. because the target encoding cannot capture all characters in the string).

getURL

Given an `IRI` and an `AsyncStatusCallback` object on which to operate a callback, this method will attempt to fetch the resource at that `IRI` using the HTTP GET method. Once the request has been completed the callback is called as described in the `AsyncStatusCallback` interface.

Processing requirements

This method call must take place asynchronously. When called, control returns immediately to the calling context, and once the request is completed the callback is called. Multiple calls to this method must be executed in FIFO order.

User Agents are required to support the gzip content coding and must decode it before passing it on to the callback. User Agents are not required to support encoding content that they send, though they are encouraged to. Cookies should be supported so that state can be maintained across requests. User Agents may provide the user with means to interact with the request (e.g. to enter authentication information) but is not required to. For security reasons, User Agents are encouraged to restrict the domains to which one may make such requests. When enforcing such restrictions, the callback is called immediately with its `AsyncURLStatus` object's success field set to false and other fields set to `null`. Redirection responses (3xx HTTP codes) must not be exposed through the API but rather they must be processed internally according to the HTTP specification.

Parameters

in `DOMString iri` The `IRI` of the resource that is being requested.

in `AsyncStatusCallback callback` The object on which the callback will be called upon completion of the request.

No return value
No exceptions

postURL

Given an `IRI`, data to be transmitted, an `AsyncStatusCallback` object on which to operate a callback, a media type, and a content coding, this method will send the data to the specified `IRI` as the body of an HTTP POST request using the requested media type and content coding. Once the request has been completed the callback is called as described in the `AsyncStatusCallback` interface.

Processing requirements are the same as for `getURL`, with the following notes and additions.

- The data passed in does not get any HTML form encoding applied to it, so that applications that wish to transmit content corresponding to what an HTML form would must produce the encoding themselves
- When the content type parameter is set then the Content-Type header of the request must be set accordingly. If the syntax of the content type parameter does not match that of a media type, it must be ignored. If this parameter is not specified, then it must default to `text/plain`.
- When the encoding parameter is set then the User Agent must encode the submitted data with that HTTP content coding and set the Content-Encoding header accordingly, if it supports it. If it does not support it, then it must ignore it, must not set the Content-Encoding header, and must

transmit the data with no encoding. The only required content coding is `identity`.

Parameters

in DOMString <code>iri</code>	The IRI of the resource that is being requested.
in DOMString <code>data</code>	The data that will be the body of the POST request.
in AsyncStatusCallback <code>callback</code>	The object on which the callback will be called upon completion of the request.
in DOMString <code>type</code>	The content type of the POST request.
in DOMString <code>encoding</code>	The encoding of the POST request.

No return value

No exceptions

`parseXML`

Given a string and a [Document](#) object, parse the string as an XML document and return a [Node](#) representing it. If the XML in the string is not well-formed according to either [XML 1.0](#) or [XML 1.1](#) or not namespace-well-formed according to [Namespaces in XML 1.1](#), this method must return a `null` value.

The purpose of the `document` parameter is to provide the context into which the XML is parsed. If it is `null`, this method must return a [Document](#) object representing the parsed XML. If it is defined, this method returns an [Element](#) object the `ownerDocument` field of which must be set to be the provided [Document](#) object. In effect when the `document` parameter is specified the processing must be equivalent to applying the following steps:

1. parsing the XML into a [Document](#)
2. retrieving its [documentElement](#) element
3. calling `importNode` on the [Document](#) object passed to `parseXML`, with the [Element](#) from the previous step as its first parameter and the `deep` parameter set to `true`. (Please note that `importNode` is part of DOM 3 Core but not of the uDOM. It is mentioned here to indicate that the effect must be as if `importNode` had been used, but not to require that it be supported in implementations)
4. return the result of the last step, cast to [Element](#)

Parameters

in DOMString <code>data</code>	The data that is to be parsed as XML.
in Document <code>contextDoc</code>	The Document object in the context of which to produce the parsing.

Return value

[Node](#) A [Node](#) (either a [Document](#) or an [Element](#)) representing the content that was parsed.

No exceptions

A.7.20 AsyncStatusCallback

This interface is implemented by code that intends to process content retrieved through `getURL` or `postURL`, both of which take an instance of this interface as a parameter, and call its `operationComplete` method upon completion of the request.

IDL Definition

```
interface AsyncStatusCallback
{
    void operationComplete(in AsyncURLStatus status);
};
```

No defined constants

No defined attributes

Methods

`operationComplete`

This method is implemented by code in order to be notified of the result of HTTP requests. Upon request completion, it receives an [AsyncURLStatus](#) object that captures the details of what was returned.

Parameters

in [AsyncURLStatus](#) `status` An object representing the HTTP response.

No return value

No exceptions

A.7.21 AsyncURLStatus

This interface captures several aspects of an HTTP response in order to be passed to the `operationComplete` method upon completion of an HTTP request.

IDL Definition

```
interface AsyncURLStatus
{
    readonly attribute boolean success;
    readonly attribute DOMString contentType;
    readonly attribute DOMString content;
};
```

No defined constants

Attributes

success

A boolean field indicating whether the request succeeded or not. For response codes in the 200 range, it must be set to true, and for responses in the 400 and 500 ranges it must be set to false. Responses in the 100 range must be ignored and responses in the 300 range must be processed as indicated in [getURL](#)'s processing requirements.

contentType

A string containing the media type of the response, which must be that set in the Content-Type HTTP header. If there was no Content-Type header, its value is null.

content

A string containing the body of the HTTP response. Binary HTTP bodies must not be used and implementations must set this field to null when they receive one. If the body was in the `text/*` hierarchy and specified a `charset` parameter, then the text must be converted into the host programming language's native form if the encoding is supported. If the encoding is not supported, the value of this field must be null. The only required encodings are UTF-8 and UTF-16 (BE and LE). If the HTTP body had one or more content codings applied to it then it must be fully decoded before setting this field.

No defined methods

B IDL Definitions

This appendix contains the complete OMG IDL for the SVG UDOM.

The SVG UDOM IDL defines the model for the SVG UDOM. Note that the SVG UDOM IDL is defined such that some interfaces have more than one base class. The different standard language bindings for the SVG UDOM are responsible for defining how to map all aspects of the SVG UDOM into the given language, including how the language should implement interfaces with more than one base class.

```

pragmas
{
  java.jni.api.name="org.w3c.dom";
  core.package.vendor="W3C";
  core.package.name="SVG Tiny";
  core.package.id="svgt";
};

[
  comment="subsetted Core DOM";
  java.jni.api.name="org.w3c.dom";
]

module dom
{
  valuetype DOMString sequence<unsigned short>;
  typedef Object DOMObject;
  interface Node;
  interface Element;
  interface Document;
  interface DOMImplementation;

  exception DOMException
  {
    unsigned short code;
  };

  const unsigned short INDEX_SIZE_ERR = 1;
  const unsigned short HIERARCHY_REQUEST_ERR = 3;
  const unsigned short WRONG_DOCUMENT_ERR = 4;
  const unsigned short INVALID_CHARACTER_ERR = 5;
  const unsigned short NO_MODIFICATION_ALLOWED_ERR = 7;
  const unsigned short NOT_FOUND_ERR = 8;
  const unsigned short NOT_SUPPORTED_ERR = 9;
  const unsigned short INVALID_STATE_ERR = 11;
  const unsigned short INVALID_MODIFICATION_ERR = 13;
  const unsigned short NAMESPACE_ERR = 14;
  const unsigned short INVALID_ACCESS_ERR = 15;
  const unsigned short TYPE_MISMATCH_ERR = 17;

  interface Node
  {
    readonly attribute DOMString namespaceURI;
    readonly attribute DOMString localName;
    readonly attribute Node parentNode;
    readonly attribute Document ownerDocument;
    attribute DOMString textContent;
    Node appendChild(in Node newChild) raises(DOMException);
    Node insertBefore(in Node newChild, in Node refChild) raises(DOMException);
    Node removeChild(in Node oldChild) raises(DOMException);
    Node cloneNode(in boolean deep);
  };

  interface Element : Node
  {
    DOMString getAttributeNS(in DOMString namespaceURI, in DOMString localName) raises(DOMException);
    void setAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName, in DOMString value) raises(DOMException);
  };

  interface Document : Node
  {
    readonly attribute DOMImplementation implementation;
    Element createElementNS(in DOMString namespaceURI, in DOMString qualifiedName) raises(DOMException);
    readonly attribute Element documentElement;
    Element getElementById(in DOMString elementId);
  };

  interface DOMImplementation
  {
    boolean hasFeature(in DOMString feature, in DOMString version);
  };
};

module events
{
  typedef dom::DOMString DOMString;
  typedef dom::DOMException DOMException;
  typedef dom::Document Document;
  typedef dom::Element Element;
  interface EventTarget;
  interface EventListener;
  interface Event;

  interface EventTarget
  {
    void addEventListener(in DOMString type, in EventListener listener, in boolean useCapture) raises(DOMException);
    void removeEventListener(in DOMString type, in EventListener listener, in boolean useCapture) raises(DOMException);
    void addEventListenerNS(in DOMString namespaceURI, in DOMString type, in EventListener listener, in boolean useCapture, in DOMObject

```

```

evtGroup) raises(DOMException);
void removeEventListenerNS(in DOMString namespaceURI, in DOMString type, in EventListener listener, in boolean useCapture) raises
(DOMException);
};

interface EventListener
{
void handleEvent(in Event evt);
};

interface Event
{
readonly attribute EventTarget target;
readonly attribute EventTarget currentTarget;
readonly attribute DOMString type;
readonly attribute DOMString namespaceURI;
readonly attribute boolean cancelable;
boolean isDefaultPrevented();
void stopPropagation();
void preventDefault();
};

interface OriginalEvent
{
readonly attribute Event originalEvent;
};

interface MouseEvent : UIEvent
{
readonly attribute long screenX;
readonly attribute long screenY;
readonly attribute long clientX;
readonly attribute long clientY;
readonly attribute unsigned short button;
};

interface TextEvent : UIEvent
{
readonly attribute DOMString data;
};

interface KeyboardEvent : UIEvent
{
readonly attribute DOMString keyIdentifier;
};

interface TimeEvent : Event
{
readonly attribute long detail;
};

interface UIEvent : Event
{
readonly attribute long detail;
};

interface WheelEvent : UIEvent
{
readonly attribute unsigned long wheelDelta;
};

interface ProgressEvent : Event
{
readonly attribute boolean lengthComputable;
readonly attribute unsigned long loaded;
readonly attribute unsigned long total;
};

interface ConnectionEvent : Event
{
const unsigned short NO_ERR = 0;
const unsigned short NETWORK_ERR = 1;
readonly attribute unsigned short errorCode;
readonly attribute sequence<octet> receivedData;
};
};

module smil
{
interface ElementTimeControl
{
void beginElementAt(in float offset);
void beginElement();
void endElementAt(in float offset);
void endElement();
void pauseElement();
void resumeElement();
readonly attribute boolean isPaused;
};
};

module global
{
typedef dom::DOMString DOMString;
typedef dom::DOMException DOMException;
typedef events::EventTarget EventTarget;
interface Connection;
interface Timer;

interface Global
{
};

exception GlobalException
{
unsigned short code;
};

const unsigned short NOT_CONNECTED_ERR = 1;

```

```

const unsigned short ENCODING_ERR = 2;

interface Connection : EventTarget
{
    readonly attribute boolean connected;
    void connect(in DOMString host, in unsigned short port) raises(GlobalException);
    void send(in sequence<octet> data) raises(GlobalException);
    void close();
};

interface Timer : events::EventTarget
{
    attribute long delay;
    attribute long interval;
    readonly attribute boolean running;
    void start();
    void stop();
};

};

module svg
{
    typedef dom::DOMString DOMString;
    typedef dom::DOMException DOMException;
    typedef dom::Document Document;
    typedef dom::Element Element;
    typedef dom::DOMObject DOMObject;
    typedef global::Connection Connection;
    typedef global::Timer Timer;
    typedef global::Global Global;
    typedef events::EventListener EventListener;
    typedef events::EventTarget EventTarget;
    interface SVGSVGElement;
    interface SVGRGBColor;
    interface SVGRect;
    interface SVGPoint;
    interface SVGPath;
    interface SVGMatrix;
    interface SVGLocatableElement;
    interface SVGElement;
    interface SVGAnimationElement;
    interface SVGDocument;
    interface SVGGlobal;

    exception SVGException
    {
        unsigned short code;
    };

    const unsigned short SVG_INVALID_VALUE_ERR = 1;
    const unsigned short SVG_MATRIX_NOT_INVERTABLE = 2;

    interface SVGDocument : Document, EventTarget
    {
        readonly attribute SVGGlobal global;
    };

    interface SVGElementInstance : EventTarget
    {
        readonly attribute SVGElement correspondingElement;
        readonly attribute SVGElement correspondingUseElement;
    };

    interface SVGSVGElement : SVGLocatableElement, SVGAnimationElement
    {
        const unsigned short NAV_AUTO           = 1;
        const unsigned short NAV_NEXT           = 2;
        const unsigned short NAV_PREV           = 3;
        const unsigned short NAV_UP             = 4;
        const unsigned short NAV_UP_RIGHT       = 5;
        const unsigned short NAV_RIGHT          = 6;
        const unsigned short NAV_DOWN_RIGHT     = 7;
        const unsigned short NAV_DOWN           = 8;
        const unsigned short NAV_DOWN_LEFT     = 9;
        const unsigned short NAV_LEFT           = 10;
        const unsigned short NAV_UP_LEFT        = 11;
        attribute float currentScale;
        attribute float currentRotate;
        readonly attribute SVGPoint currentTranslate;
        readonly attribute SVGRect viewport;
        attribute float currentTime;
        SVGMatrix createSVGMatrixComponents(in float a, in float b, in float c, in float d, in float e, in float f);
        SVGRect createSVGRect();
        SVGPath createSVGPath();
        SVGRGBColor createSVGRGBColor(in long red, in long green, in long blue) raises(SVGException);
        void moveFocus(in unsigned short motionType) raises(DOMException);
        void setFocus(in DOMObject object) raises(DOMException);
        DOMObject getCurrentFocusedObject();
    };

    interface SVGRGBColor
    {
        readonly attribute unsigned long red;
        readonly attribute unsigned long green;
        readonly attribute unsigned long blue;
    };

    interface SVGRect
    {
        attribute float x;
        attribute float y;
        attribute float width;
        attribute float height;
    };

    interface SVGPoint
    {
        attribute float x;
        attribute float y;
    };
};

```



```

interface SVGPath
{
    const unsigned short MOVE_TO = 77;
    const unsigned short LINE_TO = 76;
    const unsigned short CURVE_TO = 67;
    const unsigned short QUAD_TO = 81;
    const unsigned short CLOSE = 90;
    readonly attribute unsigned long numberOfSegments;
    unsigned short getSegment(in unsigned long cmdIndex) raises(DOMException);
    float getSegmentParam(in unsigned long cmdIndex, in unsigned long paramIndex) raises(DOMException);
    void moveTo(in float x, in float y);
    void lineTo(in float x, in float y);
    void quadTo(in float x1, in float y1, in float x2, in float y2);
    void curveTo(in float x1, in float y1, in float x2, in float y2, in float x3, in float y3);
    void close();
};

interface SVGMatrix
{
    float getComponent(in unsigned long index) raises(DOMException);
    SVGMatrix mMultiply(in SVGMatrix secondMatrix);
    SVGMatrix inverse() raises(SVGException);
    SVGMatrix mTranslate(in float x, in float y);
    SVGMatrix mScale(in float scaleFactor);
    SVGMatrix mRotate(in float angle);
};

interface SVGLocatable
{
    SVGRect getBBox();
    SVGMatrix getScreenCTM();
    SVGRect getScreenBBox();
};

interface SVGLocatableElement : SVGElement, SVGLocatable
{
};

interface TraitAccess
{
    DOMString getTrait(in DOMString name) raises(DOMException);
    DOMString getTraitNS(in DOMString namespaceURI, in DOMString name) raises(DOMException);
    float getFloatTrait(in DOMString name) raises(DOMException);
    SVGMatrix getMatrixTrait(in DOMString name) raises(DOMException);
    SVGRect getRectTrait(in DOMString name) raises(DOMException);
    SVGPath getPathTrait(in DOMString name) raises(DOMException);
    SVGRGBColor getRGBColorTrait(in DOMString name) raises(DOMException);
    DOMString getPresentationTrait(in DOMString name) raises(DOMException);
    DOMString getPresentationTraitNS(in DOMString namespaceURI, in DOMString name) raises(DOMException);
    float getFloatPresentationTrait(in DOMString name) raises(DOMException);
    SVGMatrix getMatrixPresentationTrait(in DOMString name) raises(DOMException);
    SVGRect getRectPresentationTrait(in DOMString name) raises(DOMException);
    SVGPath getPathPresentationTrait(in DOMString name) raises(DOMException);
    SVGRGBColor getRGBColorPresentationTrait(in DOMString name) raises(DOMException);
    void setTrait(in DOMString name, in DOMString value) raises(DOMException);
    void setTraitNS(in DOMString namespaceURI, in DOMString name, in DOMString value) raises(DOMException);
    void setFloatTrait(in DOMString name, in float value) raises(DOMException);
    void setMatrixTrait(in DOMString name, in SVGMatrix matrix) raises(DOMException);
    void setRectTrait(in DOMString name, in SVGRect rect) raises(DOMException);
    void setPathTrait(in DOMString name, in SVGPath path) raises(DOMException);
    void setRGBColorTrait(in DOMString name, in SVGRGBColor color) raises(DOMException);
};

interface ElementTraversal
{
    readonly attribute Element firstElementChild;
    readonly attribute Element lastElementChild;
    readonly attribute Element nextElementSibling;
    readonly attribute Element previousElementSibling;
};

interface SVGElement : Element, EventTarget, TraitAccess, ElementTraversal
{
    attribute DOMString id;
};

interface SVGAnimationElement : SVGElement, smil::ElementTimeControl
{
};

interface SVGVisualMediaElement : SVGLocatableElement, SVGAnimationElement
{
};

interface EventListenerInitializer2
{
    void initializeEventListeners( in Element scriptElement );
    EventListener createEventListener( in Element handlerElement );
};

interface SVGGlobal : Global, EventListenerInitializer2
{
    Connection createConnection();
    Timer createTimer();
    void gotoLocation(in DOMString newIRI);
    readonly attribute Document document;
    readonly attribute Global parent;
    DOMString binaryToString(in sequence<octet>, in DOMString encoding) raises(GlobalException);
    sequence<octet> stringToBinary(in DOMString data, in DOMString encoding) raises(GlobalException);
    DOMString getURL(in DOMString iri, in AsyncStatusCallback callback);
    DOMString postURL(in DOMString iri, in DOMString data, in AsyncStatusCallback callback, in DOMString type, in DOMString encoding);
    Node parseXML(in DOMString data, in Document contextDoc);
};

interface AsyncStatusCallback
{
    void operationComplete(in AsyncURLStatus status);
};

interface AsyncURLStatus
{
    readonly attribute boolean success;
};

```

```
    readonly attribute DOMString contentType;  
    readonly attribute DOMString content;  
};  
};
```

C Implementation Requirements

Contents

- C.1 [Introduction](#)
- C.2 [Unsupported elements, attributes, properties, attribute values and property values](#)
- C.3 [Error processing](#)
- C.4 [Namespace, version, baseProfile, requiredFeatures and requiredExtensions](#)
- C.5 [Clamping of Colour and Opacity Values](#)
- C.6 ['path' element implementation notes](#)
- C.7 [Text selection implementation notes](#)
- C.8 [Printing implementation notes](#)

This appendix is normative.

C.1 Introduction

The following are notes about implementation requirements corresponding to various features in the SVG language.

C.2 Unsupported elements, attributes, properties, attribute values and property values

Conforming SVG User Agents must ignore unknown attributes, attribute values, styling properties, styling property values, and descendant elements as follows:

- Within an SVG document fragment, unknown elements, including unknown elements in the SVG or XML Events namespaces as well as known elements in the SVG or XML Events namespaces occurring in unexpected locations, and their descendant elements do not participate in SVG rendering and are not processed beyond their contribution to the construction of the DOM. The processing model for unknown elements is equivalent to an `<svg:g>` element with the 'display' property set to 'none'.
- For known elements in the SVG or XML Events namespaces, unknown attributes that have no namespace and known attributes with unsupported values are treated as if they hadn't been specified when rendering. Unsupported values are defined in the [definitions section](#).
- Attributes put in the SVG namespace on any element are processed as unknown attributes.
- For unknown attributes in the XLink or XML Events namespaces, or known attributes in the XLink or XML Events namespaces with unsupported values, the user agent must process the element with regards to, respectively, linking or event handling, as if the attributes had not been specified.

Since fill is an inherited property, and since the fill attribute on the circle element is ignored due to the above rules, the circle's fill is indeed green, and not black.

Example: ignored-fill.svg

```
<g fill="green">
  <circle fill="hey baby, like wow" r="50"/>
</g>
```

C.3 Error processing

A conforming SVG User Agent must process errors in the following manner.

There are various scenarios where an SVG document fragment must be considered technically **in error**:

- When the content is not well-formed according to the XML 1.0 or XML 1.1 specifications [\[XML11\]](#)
- When the content is not namespace-well-formed according to the Namespaces in XML 1.1 specification [\[XML-NS\]](#)
- Other situations that are described as being *in error* in this specification

A document can go in and out of error over time. For example, document changes from the [SVG DOM](#) or from [animation](#) can cause a document to become *in error* and a further change can cause the document to become correct again.

When a document is in error the User Agent shall provide a highly perceivable indication of error.

Because of situations where a block of scripting changes might cause a given SVG document fragment to go into and out of error, error processing shall occur only at times when document presentation (e.g., rendering to the display device) is updated.

C.4 Namespace, version, baseProfile, requiredFeatures and requiredExtensions

User agents must only consider elements explicitly placed in the SVG namespace by XML Namespace declarations in the document (e.g., `<svg xmlns="http://www.w3.org/2000/svg">`) as being SVG elements.

SVG content can use attributes 'requiredFeatures' and 'requiredExtensions' to provide explicit indication of the minimal features that must be supported by the UA in order to render the SVG content correctly. SVG content can also use attributes 'version' in conjunction with 'baseProfile' to provide explicit indication of the minimal features that must be supported. For example, if 'version' is '1.2' and 'baseProfile' is 'tiny', then these attributes indicate that the content requires a UA that minimally supports the SVG Tiny 1.2 specification. If an SVG user agent does not support the minimal required feature set, then the user agent should alert or otherwise provide a highly visible notification to the user that it may not be able to render the content correctly.

However, SVG content that provides a specified value for 'version' but does not provide a specified value for 'baseProfile' simply indicates to the user agent the specification level (1.0, 1.1, 1.2) to which the content conforms. If 'version' is specified but not 'baseProfile', the SVG content does not provide sufficient information to the user agent to determine the minimum feature set that is required to render the content; the user agent can only know that the author might be using SVG language features that were not defined in earlier versions of the language. Therefore, if the SVG content specifies a version of the SVG language unknown to the user agent, then the user agent should alert or otherwise provide a highly perceivable notification to the user that it may not be able to render the content correctly.

When SVG content specifies SVG language versions, profiles, features or extensions not supported by the currently installed user agent, the user agent may notify the user how to update the user agent should a newer version be available which does support these features.

C.5 Clamping of Colour and Opacity Values

This section describes the behaviour of SVG User Agents.

Some numeric attribute and property values have restricted ranges, such as color component values. When out-of-range color or opacity values are provided, the user agent should defer any error checking until after presentation time, as composited actions might produce intermediate values which are out-of-range but final values which are within range.

Color values are not in error if they are out-of-range, even if final computations produce an out-of-range color value at presentation time. User agents should clamp color values to the nearest color value (possibly determined by simple clipping) which the system should process as late as possible (e.g., presentation time), although it is acceptable for user agents to clamp color values as early as parse time. Thus, implementation dependencies might preclude consistent behavior across different systems when out-of-range color values are used.

Opacity values out-of-range are not in error and should be clamped to the range 0 to 1 at the time which opacity values

have to be processed (e.g., at presentation time or when it is necessary to perform intermediate filter effect calculations).

C.6 'path' element implementation notes

A conforming SVG user agent must implement path rendering as follows:

- Directionality and zero-length path segments:
 - Certain line-capping and line-joining situations 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 must 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 uses the start point's directionality. If the end point has directionality but the start point doesn't, then the start point uses the end point's directionality. Otherwise, set the directionality for the path segment's start and end points to align with the positive x-axis in user space.
 - If **'stroke-linecap'** is set to **butt** and the given path segment has zero length, the linecap for that segment must not be drawn; however, the linecap for zero-length path segments when **'stroke-linecap'** is set to either **round** or **square** must be drawn. (This allows round and square dots to be drawn on the canvas.)
- The S/s commands indicate that the first control point of the given cubic Bézier segment is calculated by reflecting the previous path segments second control point relative to the current point. The exact math must be 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 must be considered an error.
- Unrecognized contents within a path data stream (i.e., contents that are not part of the path data grammar) must be considered an error.

C.7 Text selection implementation notes

This specification does not mandate any particular scheme for visual feedback for text selection.

The following implementation notes describe the algorithm that must be used for deciding which characters are selected during a [text selection](#) operation. A conforming SVG User Agent must implement the behaviour specified by this algorithm.

As the text selection operation occurs (e.g., while the user clicks and drags the mouse to identify the selection), the user agent determines a *start selection position* and an *end selection position*, each of which represents a position in the text string between two characters. After determining start selection position and end selection position, the user agent selects the appropriate characters, where the resulting text selection must consist of either:

- no selection or
- a *start character*, an *end character* (possibly the same character), and all of the characters within the same **'text'** element whose position in the DOM is logically between the start character and end character.

On systems with pointer devices, to determine the *start selection position*, the SVG user agent should determine which boundary between characters corresponding to rendered glyphs is the best target (e.g., closest) based on the current pointer location at the time of the event that initiates the selection operation (e.g., the mouse down event). The user agent should then track the completion of the selection operation (e.g., the mouse drag, followed ultimately by the mouse up). At the end of the selection operation, the user agent should determine which boundary between characters is the best target (e.g., closest) for the *end selection position*.

If no character reordering has occurred due to bidirectionality, then the selection must consist of all characters between the *start selection position* and *end selection position*. For example, if a **text** element contains the string "abcdef" and the start selection position and end selection positions are 0 and 3 respectively (assuming the left side of the "a" is position zero), then the selection shall consist of "abc".

When the user agent is implementing selection of bidirectional text, and when the selection starts (or ends) between characters which are not contiguous in logical order, then there may be multiple potential combinations of characters that can be considered part of the selection. The algorithms to choose among the combinations of potential selection options shall choose the selection option which most closely matches the text string's visual rendering order.

When multiple characters map inseparably to a given set of one or more glyphs, the user agent may either disallow the selection to start in the middle of the glyph set or may attempt to allocate portions of the area taken up by the glyph set to the characters that correspond to the glyph.

For systems which support pointer devices such as a mouse, the user agent is required to provide a mechanism for selecting text even when the given text has associated event handlers or links, which might block text selection due to event processing precedence rules (see [Pointer events](#)). One implementation option: For platforms which support a pointer device such as a mouse, the user agent may provide for a small additional region around character cells which initiates text selection operations but does not initiate event handlers or links.

C.8 Printing implementation notes

Conforming SVG User Agents which support both zooming on display devices and printing, the default printing option should produce printed output that reflects the display device's current view of the current SVG document fragment (assuming there is no media-specific styling), taking into account any zooming and panning done by the user, the current state of animation, and any document changes due to DOM and scripting . Thus, if the user zooms into a particular area of a map on the display device and then requests a hardcopy, the hardcopy should show the same view of the map as appears on the display device. If a user pauses an animation and prints, the hardcopy should show the same graphics as the currently paused picture on the display device. If scripting has added or removed elements from the document, then the hardcopy should reflect the same changes that would be reflected on the display.

When an SVG document is rendered on a static-only device such as a printer which does not support SVG's animation and scripting and facilities, then the user agent shall ignore any animation and scripting elements in the document and render the remaining graphics elements according to the rules in this specification.

D Conformance Criteria

Contents

- D.1 [Introduction](#)
- D.2 [Terminology](#)
- D.3 [SVG Content Conformance](#)
 - D.3.1 [Conforming SVG Document Fragments](#)
 - D.3.2 [Conforming SVG Stand-Alone Documents](#)
 - D.3.3 [Conforming SVG Included Document Fragments](#)
 - D.3.4 [Conditionally Conforming SVG Tiny 1.2 Document Fragments](#)
- D.4 [SVG Writer Conformance](#)
 - D.4.1 [Conforming SVG Generators](#)
 - D.4.2 [Conforming SVG Authoring Tools](#)
 - D.4.3 [Conforming SVG Servers](#)
- D.5 [Conforming SVG Readers](#)
 - D.5.1 [Conforming SVG Interpreters](#)
 - D.5.2 [Conforming SVG Viewers](#)

This appendix is normative

D.1 Introduction

This specification defines conformance for several classes of products:

- Content (both complete SVG files, and those portions of XML files that are in the SVG namespace)
- Software that *reads* SVG (with further conformance requirements for software that displays SVG after reading it)
- Software that *writes* SVG (including authoring tools and servers)

Some SVG displaying software will not display animations or other run-time modifications - for example, server side rasterizers or SVG-enabled printers. Therefore, this specification has different conformance levels for static and dynamic SVG viewers.

D.2 Terminology

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

All examples are informative, not normative. All chapters are normative except for specific sections marked as being informative. All appendices state whether the appendix is normative or informative. In the case of a conflict between the prose of this specification and the RelaxNG schema, the prose is authoritative (for example, the prose description of some attributes has an [EBNF](#) grammar for allowed values, which the RelaxNG is not able to express). Similarly in the case of a conflict between a DTD or W3C XML Schema and the RelaxNG schema, the RelaxNG is authoritative (RelaxNG can express some constraints on content models that are problematic to express in W3C XML schema, and expresses namespaces in a natural and more general way than a DTD is able to).

D.3 SVG Content Conformance

D.3.1 Conforming SVG Document Fragments

An SVG document fragment is a *Conforming SVG Document Fragment* if it adheres to the specification described in this document ([Scalable Vector Graphics \(SVG\) Specification](#)) including SVG's schema (see [Relax NG schema](#)) and also:

- Is well-formed as defined in either [XML 1.1](#) or [XML 1.0](#) and conforms to [Namespaces in XML 1.1](#);
- Matches the NVDL script below, or alternatively if after having removed all elements not in the SVG namespace, and all attributes on elements in the SVG namespace that are in a namespace that isn't that of XLink, XML Events, or XML attributes, it validates against the Relax NG schema;
- Where the specification provides further constraints not expressed in the schema (such as for instance EBNF grammars for attribute values), it complies to them.

NVDL script:

```
<rules xmlns='http://purl.oclc.org/dsdl/nvd1/ns/structure/1.0'>
  <namespace ns='http://www.w3.org/2000/svg'>
    <validate schema='Tiny-1.2.rng'>
      <mode>
        <namespace ns='http://www.w3.org/2000/svg' match='attributes'>
          <reject/>
        </namespace>
        <namespace ns='' match='attributes'>
          <attach/>
        </namespace>
        <namespace ns='http://www.w3.org/XML/1998/namespace' match='attributes'>
          <attach/>
        </namespace>
        <namespace ns='http://www.w3.org/1999/xlink' match='attributes'>
          <attach/>
        </namespace>
        <namespace ns='http://www.w3.org/2001/xml-events' match='elements attributes'>
          <attach/>
        </namespace>
        <anyNamespace match='elements attributes'>
          <mode>
            <anyNamespace>
              <allow/>
            </anyNamespace>
          </mode>
        </anyNamespace>
      </mode>
    </validate>
  </namespace>
</rules>
```

The SVG language or these conformance criteria provide no designated size limits on any aspect of SVG content. There are no maximum values on the number of elements, the amount of character data, or the number of characters in attribute values.

D.3.2 Conforming SVG Stand-Alone Documents

A document is a *Conforming SVG Stand-Alone Document* if:

- it conforms to the criteria for Conforming SVG Document Fragment; and
- its root element is an **'svg'** element in the SVG namespace.

D.3.3 Conforming SVG Included Document Fragments

SVG document fragments can be included within parent XML documents using the XML namespace facilities described in [Namespaces in XML 1.1](#).

An SVG document fragment that is included within a parent XML document is a *Conforming Included SVG Document Fragment* if the SVG document fragment, when taken out of the parent XML document, conforms to the criteria for *Conforming SVG Document Fragments*.

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:

```
<ParentXML xmlns="http://ns.example/">
  <!-- Elements from ParentXML go here -->
  <!-- The following is not conforming -->
  <z:rect xmlns:z="http://www.w3.org/2000/svg"
    x="0" y="0" width="10" height="10"/>
  <!-- More elements from ParentXML go here -->
</ParentXML>
```

Instead, for the SVG part to become a Conforming Included SVG Document Fragment, the document could be modified as follows:

```
<ParentXML xmlns="http://ns.example/">
  <!-- Elements from ParentXML go here -->
  <!-- The following is conforming -->
  <z:svg xmlns:z="http://www.w3.org/2000/svg"
    width="100px" height="100px" >
    <z:rect x="0" y="0" width="10" height="10" />
  </z:svg>
  <!-- More elements from ParentXML go here -->
</ParentXML>
```

D.3.4 Conditionally Conforming SVG Tiny 1.2 Document Fragments

It is sometimes desirable to create content conforming to a larger profile, and have fallback to a lower profile. For example, some SVG content might have a switch element where one branch, protected by a test attribute that will evaluate to false in a conformant SVG 1.2 Tiny viewer, uses features not available in SVG Tiny 1.2 (pattern fills, clipped images, filter effects) and another branch has SVG Tiny 1.2 content (such as a gradient fill). The parts of the content that will be rendered by an SVG Tiny 1.2 viewer are all SVG Tiny 1.2, yet the content as a whole does not conform to SVG Tiny 1.2.

An SVG document fragment is a *Conditionally Conforming SVG Tiny 1.2 Document Fragment* if:

1. All elements with a requiredFeature string not defined in this specification are marked as false
2. All elements with a requiredExtension are marked as false
3. The document fragment is transformed to remove all elements (and their children) marked false
4. The transformed document fragment is a *Conforming SVG Document Fragment*

D.4 SVG Writer Conformance

D.4.1 Conforming SVG Generators

A *Conforming SVG Generator* is a program which:

- always creates at least one of [Conforming SVG Document Fragments](#), [Conforming SVG Stand-Alone Files](#) or [Conforming SVG Included Documents](#).
- does not create non-conforming SVG document fragments of any of the above types.

When writing elements that have an ID defined, SVG Generator SHOULD prefer the '[xml:id](#)' attribute [[XMLID](#)] over the '[id](#)' attribute for content that is known to target SVG versions 1.2 and above, and SHOULD include both if the content is intended to work for versions prior to 1.2. If both '[xml:id](#)' and '[id](#)' are included on the same element, their value MUST be the same.

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.4.2 Conforming SVG Authoring Tools

Conforming SVG Authoring Tools must meet all the requirements of a Conforming SVG Generator. Additionally, a Conforming SVG Authoring Tool must conform to all of the Priority 1 accessibility guidelines from the document "Authoring Tool Accessibility Guidelines 1.0" [ATAG] that are relevant to generators of SVG content. (Priorities 2 and 3 are encouraged, but not required for conformance.)

D.4.3 Conforming SVG Servers

Conforming SVG Servers must meet all the requirements of a Conforming SVG Generator. In addition, Conforming SVG Servers using HTTP or other protocols that use Internet Media types must serve SVG stand-alone files with the media type `image/svg+xml`.

Also, if the SVG content is compressed with gzip or deflate (i.e. an svgz file), Conforming SVG Servers must indicate this with the "Content-Encoding: gzip" or "Content-Encoding: deflate" headers, as appropriate, if the protocol supports them.

Note: This compression of *content* (the entity, in HTTP terms) is distinct from automatic compression of the *message*, as defined in HTTP/1.1 [TE/Transfer Encoding](#).

D.5 Conforming SVG Readers

D.5.1 Conforming SVG Interpreters

An SVG interpreter is a program which can parse and process SVG document fragments. Examples of SVG interpreters are server-side transcoding tools (e.g., a tool which converts SVG content into modified SVG content) or analysis tools (e.g., a tool which extracts the text content from SVG content). An [SVG viewer](#) also satisfies the requirements of an SVG interpreter in that it can parse and process SVG document fragments, where processing consists of rendering the SVG content to the target medium.

In a *Conforming SVG Interpreter*, the XML parser must be able to parse and process all XML constructs defined within [\[XML11\]](#) and [\[XML-NS\]](#).

A *Conforming SVG Interpreter* must be able to parse and process a conforming SVG Tiny 1.1 document fragment [\[SVG11\]](#).

There are two sub-categories of *Conforming SVG Interpreters*:

- *Conforming Static SVG Interpreters* must be able to parse and process the static language features of SVG that correspond to the feature string "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static" (see [Feature strings](#)).
- In addition to the requirements for the static category, *Conforming Dynamic SVG Interpreters* must be able to parse and process the language features of SVG that correspond to the feature string "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all" (see [Feature strings](#)) and which support all of the required features in the [SVG DOM](#) described in this specification.

A Conforming SVG Interpreter must parse any SVG document correctly. It is not required to interpret the semantics of all features correctly. It needs only check the syntax of attribute values on elements in the SVG namespace, and element content models in the SVG namespace that it knows about from the profile it implements (SVG Tiny 1.2).

Note: A transcoder from SVG into another graphics representation, such as an SVG-to-raster transcoder, represents a viewer, and thus viewer conformance criteria apply. (See [Conforming SVG Viewers](#).)

D.5.2 Conforming SVG Viewers

An SVG viewer is a program which can parse and process an SVG document fragment and render the contents of the

document onto some sort of output medium such as a display or printer; thus, an *SVG Viewer* is also an *SVG Interpreter*.

There are two sub-categories of *Conforming SVG Viewers*:

- *Conforming Static SVG Viewers* support the static language features of SVG that correspond to the feature string "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static" (see [Feature strings](#)). This category often corresponds to platforms and environments which only render static documents, such as printers.
- *Conforming Dynamic SVG Viewers* support the language features of SVG that correspond to the feature string "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all" (see [Feature strings](#)). This category often applies to platforms and environments such as common Web browsers which support user interaction and dynamic document content (i.e., documents whose content can change over time). (User interaction includes support for hyperlinking, events [e.g., mouse clicks], text selection, zooming and panning [see [Interactivity](#)]. Dynamic document content can be achieved via [declarative animation](#) or by scripts modifying the [SVG DOM](#).)

Specific criteria that apply to both *Conforming Static SVG Viewers* and *Conforming Dynamic SVG Viewers*:

- The program must also be a [Conforming SVG Interpreter](#),
- For interactive user environments, facilities must exist for zooming and panning of stand-alone SVG documents or SVG document fragments embedded within parent XML documents.
- In environments that have appropriate user interaction facilities, the viewer must support the ability to activate hyperlinks.
- If printing devices are supported, SVG content must be printable at printer resolutions with the same graphics features available as required for display (e.g., the specified colors must be rendered on color printers).
- On systems where this information is available, the parent environment must provide the viewer with information about physical device resolution. In situations where this information is impossible to determine, the parent environment shall pass a reasonable value for device resolution which tends to approximate most common target devices.
- The viewer must support JPEG/JFIF [[JPEG](#)] [[JFIF](#)] and PNG [[PNG](#)] image formats. Other image formats may be supported in addition.
- Resampling of image data must be consistent with the specification of property '[image-rendering](#)'.
- The viewer must support alpha channel blending of the image of the SVG content onto the target canvas.
- SVG implementations must correctly support [gzip](#)-encoded [[RFC1952](#)] and [deflate](#)-encoded [[RFC1951](#)] data streams, for any content type (including SVG, script files, images). SVG implementations that support HTTP must support these encodings according to the [HTTP 1.1](#) specification [[RFC2616](#)]; in particular, the client must specify with an "Accept-Encoding:" request header [[HTTP-ACCEPT-ENCODING](#)] those encodings that it accepts, including at minimum [gzip](#) and [deflate](#), and then decompress any [gzip](#)-encoded and [deflate](#)-encoded data streams that are downloaded from the server. Implementations must also support progressive rendering of compressed data streams.
- The viewer must support content using the [data: protocol](#) [[RFC2397](#)] wherever [IRI referencing](#) of whole documents (such as raster images, SVG documents, fonts and color profiles) is permitted within SVG content. This support must include use of base64 encoded content. (Note: fragments of SVG content which do not constitute an entire SVG document are not available using the "data:" protocol.)
- The viewer must support the following W3C Recommendations with regard to SVG content:
 - complete support for the XML 1.1 specification [[XML11](#)].
 - complete support for "Namespaces in XML 1.1" [[XML-NS](#)], including inclusion of non-SVG namespaces within SVG content. (Note that data from non-SVG namespaces are included in the DOM but are otherwise ignored from the point of view of rendering and interaction.)
- All visual rendering should be accurate to within one px unit to the mathematically correct result at the initial 1:1 zoom ratio. It is suggested that viewers attempt to keep a high degree of accuracy when zooming.
- On systems which support accurate sRGB [[SRGB](#)] color, all sRGB color computations and all resulting color values must be accurate to within one sRGB color component value, where sRGB color component values range from 0 to 255.

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

Specific criteria that apply to only *Conforming Dynamic SVG Viewers*:

- In Web browser environments, the viewer must have the ability to search and select text strings within SVG content.
- In interactive environments, the viewer must have the ability to select and copy text from SVG content to the system clipboard.

The Web Accessibility Initiative [\[WAI\]](#) has defined "User Agent Accessibility Guidelines 1.0" [\[UAAG\]](#). A Conforming SVG Viewer must conform to the Priority 1 accessibility guidelines defined in [\[UAAG\]](#), and should conform also to Priorities 2 and 3.

A *Conforming SVG Viewer* must be able to apply styling properties to SVG content using [presentation attributes](#) (see [properties shared with CSS and XSL](#)).

If the user agent includes an XHTML or SMIL viewing capability, then a *Conforming SVG Viewer* must support resources of MIME type "image/svg+xml" wherever raster image external resources can be used, such as in the XHTML **'img'** element.

If the user agent can apply CSS/XSL styling properties to XML documents (including XHTML, SMIL, or XForms documents), then a *Conforming SVG Viewer* must support resources of MIME type "image/svg+xml" in CSS/XSL properties that can refer to raster image resources (e.g., **'background-image'**).

E Conformance to QA Framework Specification Guidelines

Contents

- E.1 [Introduction](#)
- E.2 [Checklist table](#)

E.1 Introduction

This table is derived from the checklist of all defined requirements and good practices from the [QA Framework: Specification Guidelines \[QAF-SPEC\]](#) For each requirement and good practice, the table links to the part of the SVG Tiny 1.2 specification that satisfied the requirement or best practice.

E.2 Checklist table

Guidelines	YES	NO	N/A	Comments
2.1 Specifying Conformance				
Requirement 01: Include a conformance clause.	YES			
Good Practice 01: Define the specification's conformance model in the conformance clause.	YES			
Good Practice 02: Specify in the conformance clause how to distinguish normative from informative content.	YES			
Good Practice 03: Provide the wording for conformance claims.		no		
Good Practice 04: Provide an Implementation Conformance Statement Pro Forma.		no		

Good Practice 05:
Require an
Implementation
Conformance Statement
as part of valid
conformance claims.

no

2.2 Setting up ground rules

Requirement 02: Define
the scope.

YES

Good Practice 06:
Provide examples, use
cases, and graphics.

YES

Good Practice 07: Write
sample code or tests.

YES

Requirement 03:
Identify who or what
will implement the
specification.

YES

Requirement 04: Make
a list of normative
references.

YES

Good Practice 08: When
imposing requirements
by normative references,
address conformance
dependencies.

YES

2.3 Defining and using terminology

Requirement 05: Define
the terms used in the
normative parts of the
specification.

YES

Requirement 06:
Create conformance
labels for each part of
the conformance
model.

YES

<u>Good Practice 09: Define unfamiliar terms in-line and consolidate the definitions in a glossary section.</u>	<u>YES</u>			
<u>Good Practice 10: Use terms already defined without changing their definition.</u>	<u>YES</u>			
<u>Requirement 07: Use a consistent style for conformance requirements and explain how to distinguish them.</u>	<u>YES</u>			
<u>Requirement 08: Indicate which conformance requirements are mandatory, which are recommended, and which are optional.</u>	<u>YES</u>			
<u>Good Practice 11: Use formal languages when possible.</u>	<u>yes</u>			
<u>Good Practice 12: Write Test Assertions.</u>		no		
2.4 Managing Variability				
<u>Good Practice 13: Create subdivisions of the technology when warranted.</u>	<u>YES</u>			

<p><u>Requirement 09: If the technology is subdivided, then indicate which subdivisions are mandatory for conformance.</u></p>	<p><u>YES</u></p>	
<p><u>Requirement 10: If the technology is subdivided, then address subdivision constraints.</u></p>	<p><u>YES</u></p>	
<p><u>Good Practice 14: If the technology is profiled, define rules for creating new profiles.</u></p>	<p><u>YES</u></p>	
<p><u>Good Practice 15: Use optional features as warranted.</u></p>	<p>YES</p>	
<p><u>Good Practice 16: Clearly identify optional features.</u></p>	<p>YES</p>	
<p><u>Good Practice 17: Indicate any limitations or constraints on optional features.</u></p>	<p>YES</p>	
<p><u>Requirement 11: Address Extensibility.</u></p>	<p><u>YES</u></p>	
<p><u>Good Practice 18: If extensibility is allowed, define an extension mechanism.</u></p>	<p><u>YES</u></p>	
<p><u>Good Practice 19: Warn extension creators to create extensions that do not interfere with conformance.</u></p>	<p><u>YES</u></p>	

Good Practice 20: Define error-handling for unknown extensions.	YES				
Requirement 12: Identify deprecated features.				n/a	None
Requirement 13: Define how each class of product handles each deprecated feature.				n/a	
Good Practice 21: Explain how to avoid using a deprecated feature.				n/a	
Good Practice 22: Identify obsolete features.				n/a	None
Good Practice 23: Define an error handling mechanism.	YES				

F Accessibility Support

Contents

- F.1 [WAI Accessibility Guidelines](#)
- F.2 [SVG Content Accessibility Guidelines](#)
- F.3 [SVG User Agent Accessibility Guidelines](#)

This appendix is informative, not normative.

F.1 WAI Accessibility Guidelines

This appendix explains how accessibility guidelines published by W3C's Web Accessibility Initiative (WAI) apply to SVG.

1. The "Web Content Accessibility Guidelines 1.0" [[WCAG](#)] explains how authors can create Web content that is accessible to people with disabilities.
2. The "Authoring Tool Accessibility Guidelines 1.0" [[ATAG](#)] explains how developers can design accessible authoring tools such as SVG authoring tools. [To conform to the SVG specification](#), an SVG authoring tool must conform to ATAG (priority 1). SVG support for element [grouping](#), [reuse](#) and [navigation](#) is relevant to designing accessible SVG authoring tools.
3. The "User Agent Accessibility Guidelines 1.0" [[UAAG](#)] explains how developers can design accessible user agents such as SVG-enabled browsers. [To conform to the SVG specification](#), an SVG user agent must conform to the Priority 1 accessibility guidelines defined in [[UAAG](#)], and should conform also to Priorities 2 and 3.UAAG. SVG support for scaling, the DOM, and metadata are all relevant to designing accessible SVG user agents.

The W3C Note "Accessibility Features of SVG" [[SVG-ACCESS](#)] explains in detail how the requirements of the three guidelines apply to SVG.

F.2 SVG Content Accessibility Guidelines

This section explains briefly how authors can create accessible SVG documents; it summarizes and builds upon "Accessibility Features of SVG" [[SVG-ACCESS](#)].

Provide [text equivalents](#) for graphics.

- When the text content of a graphic (e.g., in a **'text'** element) explains its function, no text equivalent is required. Use the **'title'** child element to explain the function **'text'** elements whose meaning is not clear from their text content.
- When a graphic does not include explanatory text content, it requires a text equivalent. If the equivalent is complex, use the **'desc'** element, otherwise use the **'title'** child element.
- If a graphic is built from meaningful parts, build the description from meaningful parts.

Do not rely on color alone.

- Do not use color alone to convey semantic information.
- Ensure adequate color contrast.

Use markup correctly.

- Separate structure from presentation.
- Use the **'g'** element and rich descriptions to structure SVG documents. Reuse named objects.
- Publish highly-structured documents, not just graphical representations. Documents that are rich in structure may be rendered graphically, as speech, or as Braille. For example, express mathematical relationships in MathML [[MATHML](#)] and use SVG for explanatory graphics.
- Author documents that validate to the SVG RelaxNG grammar.

Use text for text, and graphics for graphics

- Represent text as character data, not as glyphs, images or curves.

- Style text with fonts. Authors may describe their own fonts in SVG.
- Do not use 'pi' fonts or picture fonts to represent small graphics. The resulting garbage text does not conform to [CHARMOD] and confuses text to speech engines.

Provide a default reading order

- Use 'textArea' elements to provide text that wraps in a box, rather than using script to wrap the text or using a sequence of unrelated 'text' elements.

Clarify natural language usage.

- Use [xml:lang](#) to identify the natural language of content and changes in natural language. This ensures that textual content can be spell-checked, or converted to speech.
- Use the 'systemLanguage' test attribute to provide language-specific alternative text and/or graphics.

Allow for rich navigation

- Do not assume that all devices have a pointer device. Allow for keyboard navigation as well.
- Provide links for 8-way directional navigation.

Ensure that dynamic content is accessible.

- Ensure that text equivalents for dynamic content are updated when the dynamic content changes.
- Avoid storing dynamic text in ecmaScript arrays or in XSLT stylesheets. This makes it less accessible, and also increases the difficulty of localising the text or providing multilingual alternatives.
- Ensure that SVG documents are still usable when scripts or other programmatic objects are turned off or not supported.

F.3 SVG User Agent Accessibility Guidelines

This section explains how implementors of SVG User Agents can make their software more accessible.

Provide access to color information

- SVG User Agents should implement and document APIs so that assistive technologies can have access to color information on individual elements.
- SVG User Agents should provide an optional high contrast view.

Provide a text-only view

- SVG user Agents should provide mechanisms that allow assistive technologies to achieve a useful text-only view. Examples include a DOM explorer, a synchronized text only view, or an XSLT style sheet to convert the textual content to XHTML.

Allow for rich navigation

- Provide links for 8-way directional navigation.

Allow multimodal navigation and interaction

- SVG user Agents should provide access to zooming and panning through modalities other than a pointer device, if available
- SVG user Agents should allow animations, audio, and video to be started, stopped and paused using modalities other than a pointer device, if available.

Allow keyboard navigation

Where a keyboard is supported, it should be possible to use it to zoom and pan, as well as start, stop, and pause animations, audio, and video.

G Internationalization Support

Contents

- G.1 [Introduction](#)
- G.2 [Internationalization and SVG](#)
- G.3 [SVG Internationalization Guidelines](#)

This appendix is informative, not normative.

G.1 Introduction

This appendix provides a brief summary of SVG's support for internationalization. The appendix is hyperlinked to the sections of the specification which elaborate on particular topics.

G.2 Internationalization and SVG

SVG is an application of XML [[XML11](#)] and thus supports Unicode [[UNICODE](#)], which defines the universal character set.

Additionally, SVG provides a mechanism for precise control of the glyphs used to draw text strings, which is described in [SVG Fonts](#). This allows content to supply, or link to, SVG Fonts to display international text, even if no fonts for that language are installed on the client machine. This facility provides:

- the ability to specify glyphs for any character
- the ability to specify ligatures for arbitrary strings of characters
- the ability to follow the guidelines for normalizing character data for the purposes of enhanced interoperability (see [[CHARMOD](#)]), while still having precise control over the glyphs that are drawn.

SVG Tiny 1.2 supports:

- Horizontal, left-to-right text, for example as found in Roman scripts
- Bidirectional text (for languages such as Arabic and Hebrew).

SVG Tiny 1.2 does not support vertical text, but SVG Full 1.2 does.

[SVG fonts](#) support contextual glyph selection, for example for [Arabic](#) contextual forms, and language-dependent [Han glyphs](#).

Multi-language SVG documents are possible by utilizing the [systemLanguage](#) attribute to have different text strings or graphics appear based on the client machine's language setting.

G.3 SVG Internationalization Guidelines

SVG generators are encouraged to follow W3C guidelines for normalizing character data [[CHARMOD-norm](#)].

H JPEG Support

Contents

- H.1 [Introduction](#)
- H.2 [Required Support](#)

H Introduction

This appendix specifies the JPEG support required by SVG Tiny 1.2 implementations. The required support is targeted at specifying a level of functionality known to be compatibly supported within the industry *and without licensing issues*.

In general when people refer to JPEG [[JPEG](#)], they actually mean JPEG compressed images within the JFIF [[JFIF](#)] file format. JFIF was created by the Independent JPEG Group (IJG) for storing a single JPEG-compressed image in a file.

H Required Support

SVG Viewers are required to support JPEG images stored in a JFIF file [[JFIF](#)]. Other transport or storage mechanisms may be supported.

The following coding processes defined by the JPEG specification [[JPEG](#)], in Table 1, section 4.11, must be supported:

- Baseline process
- Extended DCT-based processes (with the exception of arithmetic coding).

The following statements also apply:

- 8-bit samples must be supported. 12-bit samples may be supported, (Section 4.7 [[JPEG](#)]).
- Support for the DNL Marker (Section 3.2 [[JPEG](#)]) is not required.
- Support for non-integer sampling ratios is not required. (A.1.1 [[JPEG](#)]).

The following encoding processes are not required, but may be supported

- Complete Extended DCT-based processes.
- Lossless Processes
- Hierarchical Processes

SVG Tiny 1.2 UA's should convert Y,Cb,Cr values compressed in the JPEG image to RGB as defined in the JFIF specification [[JFIF](#)] and may assume that the RGB values are sRGB.

I Minimizing SVG File Sizes

This appendix is informative.

Considerable effort has been made to make SVG file sizes as small as possible while still retaining the benefits of XML and achieving compatibility and leverage with other W3C specifications.

Here are some of the features in SVG that promote small file sizes:

- SVG's path data definition was defined to produce a compact data stream for vector graphics data: all commands are one character in length; relative coordinates are available; separator characters do not have to be supplied when tokens can be identified implicitly; smooth curve formulations are available (cubic Bziers, quadratic Bziers and elliptical arcs) to prevent the need to tessellate into polylines; and shortcut formulations exist for common forms of cubic Bzier segments, quadratic Bzier 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, different sizing and positioning.

Additionally, HTTP/1.1 allows for compressed data to be passed from server to client, which can result in significant file size reduction. Here are some sample compression results using [gzip](#) compression on SVG documents:

Uncompressed SVG	With gzip compression	Compression ratio
12,912	2,463	81%
12,164	2,553	79%
11,613	2,617	77%
18,689	4,077	78%
13,024	2,041	84%

A related issue is progressive rendering. Some SVG viewers will support:

- the ability to display the first parts of an SVG document fragments as the remainder of the document is downloaded from the server; thus, the user will see part of the SVG drawing right away and interact with it, even if the SVG file size is large.
- delayed downloading of images and fonts. Just like some HTML browsers, some SVG viewers will download images and [WebFonts](#) last, substituting a temporary image and system fonts, respectively, until the given image and/or font is available.

Here are techniques for minimizing SVG file sizes and minimizing the time before the user is able to start interacting with the SVG document fragments:

- Construct the SVG file such that any links which the user might want to click on are included at the beginning of the SVG file
- Use default values whenever possible rather than defining all attributes and properties explicitly.
- Take advantage of the [path data](#) data compaction facilities: use relative coordinates; use *h* and *v* for horizontal and vertical lines; use *s* or *t* for cubic and quadratic Bzier segments whenever possible; eliminate extraneous white space and separators.
- Utilize symbols if the same graphic appears multiple times in the document

- For user agents that support styling with CSS, utilize CSS property inheritance and selectors to consolidate commonly used properties into named styles or to assign the properties to a parent **'g'** element.

J Feature strings

This appendix is normative.

The following are the feature strings for the [requiredFeatures](#) attribute. These same feature strings apply to the [SVG uDOM's hasFeature](#) method. In some cases the feature strings map directly to SVG modules, in others they represent some functionality of the User Agent (that it is a dynamic viewer for example).

Feature String:

`http://www.w3.org/Graphics/SVG/feature/1.2/#SVG`

User Agent Must Support:

At least one of the following (all of which are described subsequently):

- `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static-DOM"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-animated"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all"`

(Because the feature string `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG"` can be ambiguous in some circumstances, it is recommended that more specific feature strings be used.)

Feature String:

`http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static`

User Agent Must Support:

All of the following features (described below):

- `"http://www.w3.org/Graphics/SVG/feature/1.2/#CoreAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Structure"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessing"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessingAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Image"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Prefetch"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Shape"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Text"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#PaintAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#OpacityAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#GraphicsAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Gradient"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#SolidColor"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#XlinkAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#ExternalResourcesRequiredAttribute"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Font"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Hyperlinking"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Extensibility"`

For SVG viewers, `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static"` indicates that the viewer can process and render successfully all of the language features in the modules corresponding to the features listed above.

Feature String:

`http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static-DOM`

User Agent Must Support:

All of the DOM interfaces and methods that correspond to the language features for `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static"` as well as the following features:

- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Scripting"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Handler"`
- `"http://www.w3.org/Graphics/SVG/feature/1.2/#Listener"`

Feature String:

`http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-animated`

User Agent Must Support:

All of the language features from `"http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static"` plus the following feature:

- `http://www.w3.org/Graphics/SVG/feature/1.2/#TimedAnimation`

For SVG viewers running on media capable of rendering time-based material, such as displays, "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-animated" indicates that the viewer can process and render successfully all of the corresponding language features.

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all

User Agent Must Support:

All of the language features from:

- o "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static-DOM"
- o "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-animated"

Plus the following features:

- o "http://www.w3.org/Graphics/SVG/feature/1.2/#Audio"
- o "http://www.w3.org/Graphics/SVG/feature/1.2/#Video"
- o "http://www.w3.org/Graphics/SVG/feature/1.2/#Animation"
- o "http://www.w3.org/Graphics/SVG/feature/1.2/#Discard"

For SVG viewers running on media capable of rendering time-based material, such as displays, "http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all" indicates that the viewer can process and render successfully all of the corresponding language features.

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#CoreAttribute

User Agent Must Support:

[Core Attribute Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#Structure

User Agent Must Support:

[Structure Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessing

User Agent Must Support:

[Conditional Processing Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessingAttribute

User Agent Must Support:

[Conditional Processing Attribute Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#Image

User Agent Must Support:

[Image Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#Prefetch

User Agent Must Support:

[Prefetch Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#Discard

User Agent Must Support:

[Discard Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#Shape

User Agent Must Support:

[Shape Module](#)

Feature String:

http://www.w3.org/Graphics/SVG/feature/1.2/#Text

User Agent Must Support:

[Text Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#PaintAttribute>

User Agent Must Support:

[Paint Attribute Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#OpacityAttribute>

User Agent Must Support:

[Opacity Attribute Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#GraphicsAttribute>

User Agent Must Support:

[Graphics Attribute Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Gradient>

User Agent Must Support:

[Gradient Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#SolidColor>

User Agent Must Support:

[Solid Color Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Hyperlinking>

User Agent Must Support:

[Hyperlinking Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#XlinkAttribute>

User Agent Must Support:

[Xlink Attribute Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#ExternalResourcesRequired>

User Agent Must Support:

[ExternalResourcesRequired Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Scripting>

User Agent Must Support:

[Scripting Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Handler>

User Agent Must Support:

[Handler Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Listener>

User Agent Must Support:

[Listener Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#TimedAnimation>

User Agent Must Support:

[Timed Animation Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Animation>

User Agent Must Support:

[Animation Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Audio>

User Agent Must Support:

[Audio Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Video>

User Agent Must Support:

[Video Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Font>

User Agent Must Support:

[Font Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#Extensibility>

User Agent Must Support:

[Extensibility Module](#)

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#TransformedVideo>

User Agent Must Support:

The ability to perform any [transformation \(including scaling\) on video content](#).

Feature String:

<http://www.w3.org/Graphics/SVG/feature/1.2/#ComposedVideo>

User Agent Must Support:

The ability to [compose video content](#) with other content.

K Element Table

This appendix lists the elements that are defined in this specification. The following legend details how it should be read.

Elements

Column containing the element names. All elements are in the SVG namespace, except for **listener** which is in the XML Events namespace.

Attributes

For each element, this lists the attributes which may occur on it. Note that these are only the attributes and not the properties.

prop.

This column indicates whether or not the given element may receive properties. There are several options:



All [properties](#) defined in this specification can be set as attributes on the given element.



None of the properties defined in this specification can be set as attributes on the given element.



Only the properties in the media group can be set on this given element. The media group is a subset of the properties that is meaningful on media elements and does not cause the 'fill' property to clash with the 'fill' SMIL attribute. The media group comprises the following properties: audio-level, display, image-rendering, pointer-events, shape-rendering, text-rendering, viewport-fill, viewport-fill-opacity, and visibility.

Possible Children

Lists the elements that can occur as children of the given element. Note that this is not the real content of the element as defined in the [RelaxNG schema](#).

<text>

Indicates that character data can occur as the child of a given element.

Elements

[a](#)

Attributes

[class](#),
[externalResourcesRequired](#),
[focusHighlight](#), [focusable](#), [id](#),
[nav-down](#), [nav-down-left](#), [nav-down-right](#), [nav-left](#), [nav-next](#),
[nav-prev](#), [nav-right](#), [nav-up](#),
[nav-up-left](#), [nav-up-right](#),
[requiredExtensions](#),
[requiredFeatures](#),
[requiredFonts](#),
[requiredFormats](#),
[systemLanguage](#), [target](#),
[transform](#), [xlink:actuate](#), [xlink:arcrole](#), [xlink:href](#), [xlink:role](#),
[xlink:show](#), [xlink:title](#), [xlink:type](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#)

prop. Possible Children



The 'a' element may contain any element that its parent may contain, except itself.

<u>animate</u>	accumulate , additive , attributeName , attributeType , begin , by , calcMode , class , dur , end , fill , from , id , keySplines , keyTimes , max , min , repeatCount , repeatDur , restart , to , values , xlink: actuate , xlink:arcrole , xlink: href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	x	desc , handler , metadata , switch , title
<u>animateColor</u>	accumulate , additive , attributeName , attributeType , begin , by , calcMode , class , dur , end , fill , from , id , keySplines , keyTimes , max , min , repeatCount , repeatDur , restart , to , values , xlink: actuate , xlink:arcrole , xlink: href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	x	desc , handler , metadata , switch , title
<u>animateMotion</u>	accumulate , additive , begin , by , calcMode , class , dur , end , fill , from , id , keyPoints , keySplines , keyTimes , max , min , origin , path , repeatCount , repeatDur , restart , rotate , to , values , xlink:actuate , xlink: arcrole , xlink:href , xlink:role , xlink:show , xlink:title , xlink: type , xml:base , xml:id , xml: lang , xml:space	x	desc , handler , metadata , mpath , switch , title

animateTransform [accumulate](#), [additive](#), [attributeName](#), [attributeType](#), [begin](#), [by](#), [calcMode](#), [class](#), [dur](#), [end](#), [fill](#), [from](#), [id](#), [keySplines](#), [keyTimes](#), [max](#), [min](#), [repeatCount](#), [repeatDur](#), [restart](#), [to](#), [type](#), [values](#), [xlink:actuate](#), [xlink:arcrole](#), [xlink:href](#), [xlink:role](#), [xlink:show](#), [xlink:title](#), [xlink:type](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#)

x

[desc](#), [handler](#), [metadata](#), [switch](#), [title](#)

animation

[begin](#), [class](#), [dur](#), [end](#), [externalResourcesRequired](#), [fill](#), [focusHighlight](#), [focusable](#), [height](#), [id](#), [initialVisibility](#), [max](#), [min](#), [nav-down](#), [nav-down-left](#), [nav-down-right](#), [nav-left](#), [nav-next](#), [nav-prev](#), [nav-right](#), [nav-up](#), [nav-up-left](#), [nav-up-right](#), [preserveAspectRatio](#), [repeatCount](#), [repeatDur](#), [requiredExtensions](#), [requiredFeatures](#), [requiredFonts](#), [requiredFormats](#), [restart](#), [syncBehavior](#), [syncMaster](#), [syncTolerance](#), [systemLanguage](#), [transform](#), [width](#), [x](#), [xlink:actuate](#), [xlink:arcrole](#), [xlink:href](#), [xlink:role](#), [xlink:show](#), [xlink:title](#), [xlink:type](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#), [y](#)

M

[animate](#), [animateColor](#), [animateMotion](#), [animateTransform](#), [desc](#), [discard](#), [handler](#), [metadata](#), [set](#), [switch](#), [title](#)

<u>audio</u>	begin , class , dur , end , externalResourcesRequired , fill , id , max , min , repeatCount , repeatDur , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , restart , syncBehavior , syncMaster , syncTolerance , systemLanguage , type , xlink: actuate , xlink:arcrole , xlink: href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	M animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>circle</u>	class , cx , cy , focusHighlight , focusable , id , nav-down , nav- down-left , nav-down-right , nav- left , nav-next , nav-prev , nav- right , nav-up , nav-up-left , nav- up-right , r , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , xml:base , xml:id , xml:lang , xml: space	✓ animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>defs</u>	class , id , xml:base , xml:id , xml: ✓ lang , xml:space	a , animate , animateColor , animateMotion , animateTransform , animation , audio , circle , defs , desc , discard , ellipse , font , font-face , foreignObject , g , handler , image , line , linearGradient , listener , metadata ,

			path , polygon , polyline , prefetch , radialGradient , rect , script , set , solidColor , switch , text , textArea , title , use , video
<u>desc</u>	class , id , xml:base , xml:id , xml:lang , xml:space	✗	<text>
<u>discard</u>	begin , class , id , xlink:actuate , xlink:arcrole , xlink:href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	✗	desc , handler , metadata , switch , title
<u>ellipse</u>	class , cx , cy , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , rx , ry , systemLanguage , transform , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>font</u>	class , externalResourcesRequired , horiz-adv-x , horiz-origin-x , id , xml:base , xml:id , xml:lang , xml:space	✗	desc , font-face , glyph , hkern , metadata , missing-glyph , switch , title

<u>font-face</u>	accent-height , alphabetic , ascent , bbox , cap-height , class , descent , externalResourcesRequired , font-family , font-stretch , font- style , font-variant , font-weight , hanging , id , ideographic , mathematical , overline- position , overline-thickness , panose-1 , slope , stemh , stemv , strikethrough-position , strikethrough-thickness , underline-position , underline- thickness , unicode-range , units-per-em , widths , x-height , xml:base , xml:id , xml:lang , xml: space	X	desc , font-face-src , metadata , switch , title
<u>font-face-name</u>	class , id , name , xml:base , xml: id , xml:lang , xml:space	X	desc , metadata , switch , title
<u>font-face-src</u>	class , id , xml:base , xml:id , xml: lang , xml:space	X	desc , font-face- name , font-face-uri , metadata , switch , title
<u>font-face-uri</u>	class , externalResourcesRequired , id , xlink:actuate , xlink:arcrole , xlink:href , xlink:role , xlink: show , xlink:title , xlink:type , xml: base , xml:id , xml:lang , xml: space	X	desc , metadata , switch , title

foreignObject

[class](#),
[externalResourcesRequired](#),
[focusHighlight](#), [focusable](#),
[height](#), [id](#), [nav-down](#), [nav-down-left](#), [nav-down-right](#), [nav-left](#), [nav-next](#), [nav-prev](#), [nav-right](#), [nav-up](#), [nav-up-left](#), [nav-up-right](#), [requiredExtensions](#),
[requiredFeatures](#),
[requiredFonts](#),
[requiredFormats](#),
[systemLanguage](#), [transform](#),
[width](#), [x](#), [xlink:actuate](#), [xlink:arcrole](#), [xlink:href](#), [xlink:role](#),
[xlink:show](#), [xlink:title](#), [xlink:type](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#), [y](#)



[desc](#), [metadata](#),
[switch](#), [title](#)

g

[class](#),
[externalResourcesRequired](#),
[focusHighlight](#), [focusable](#), [id](#),
[nav-down](#), [nav-down-left](#), [nav-down-right](#), [nav-left](#), [nav-next](#),
[nav-prev](#), [nav-right](#), [nav-up](#),
[nav-up-left](#), [nav-up-right](#),
[requiredExtensions](#),
[requiredFeatures](#),
[requiredFonts](#),
[requiredFormats](#),
[systemLanguage](#), [transform](#),
[xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#)



[a](#), [animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[animation](#), [audio](#),
[circle](#), [defs](#), [desc](#),
[discard](#), [ellipse](#),
[font](#), [font-face](#),
[foreignObject](#), [g](#),
[handler](#), [image](#),
[line](#), [linearGradient](#),
[listener](#), [metadata](#),
[path](#), [polygon](#),
[polyline](#), [prefetch](#),
[radialGradient](#), [rect](#),
[script](#), [set](#),
[solidColor](#), [switch](#),
[text](#), [textArea](#), [title](#),
[use](#), [video](#)

glyph

[arabic-form](#), [class](#), [d](#), [glyph-name](#), [horiz-adv-x](#), [id](#), [lang](#),
[unicode](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#)



[desc](#), [metadata](#),
[switch](#), [title](#)

<u>handler</u>	class , ev:event , externalResourcesRequired , id , type , xml:base , xml:id , xml:lang , xml:space	X	<text>
<u>hkern</u>	class , g1 , g2 , id , k , u1 , u2 , xml:base , xml:id , xml:lang , xml:space	X	desc , metadata , switch , title
<u>image</u>	class , externalResourcesRequired , focusHighlight , focusable , height , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , opacity , preserveAspectRatio , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , type , width , x , xlink:actuate , xlink:arcrole , xlink:href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space , y	M	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>line</u>	class , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , x1 , x2 , xml:base , xml:id , xml:lang , xml:space , y1 , y2	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title

<u>linearGradient</u>	class , gradientUnits , id , x1 , x2 , xml:base , xml:id , xml:lang , xml:space , y1 , y2	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , metadata , set , stop , switch , title
<u>listener</u>	class , defaultAction , event , handler , id , observer , phase , propagate , target , xml:base , xml:id , xml:lang , xml:space	✗	
<u>metadata</u>	class , id , xml:base , xml:id , xml:lang , xml:space	✗	<text>
<u>missing-glyph</u>	class , d , horiz-adv-x , id , xml:base , xml:id , xml:lang , xml:space	✗	desc , metadata , switch , title
<u>mpath</u>	class , id , xlink:actuate , xlink:arcrole , xlink:href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	✗	desc , metadata , switch , title
<u>path</u>	class , d , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , pathLength , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title

<u>polygon</u>	class , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , points , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>polyline</u>	class , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , points , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>prefetch</u>	bandwidth , class , id , mediaCharacterEncoding , mediaContentEncodings , mediaSize , mediaTime , xlink:actuate , xlink:arcrole , xlink:href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	✗	desc , metadata , switch , title

<u>radialGradient</u>	class , cx , cy , gradientUnits , id , r , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , metadata , set , stop , switch , title
<u>rect</u>	class , focusHighlight , focusable , height , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , rx , ry , systemLanguage , transform , width , x , xml:base , xml:id , xml:lang , xml:space , y	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title
<u>script</u>	class , externalResourcesRequired , id , type , xml:base , xml:id , xml:lang , xml:space	✗	<text>
<u>set</u>	attributeName , attributeType , begin , class , dur , end , fill , id , max , min , repeatCount , repeatDur , restart , to , xlink:actuate , xlink:arcrole , xlink:href , xlink:role , xlink:show , xlink:title , xlink:type , xml:base , xml:id , xml:lang , xml:space	✗	desc , handler , metadata , switch , title
<u>solidColor</u>	class , id , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , title

<u>stop</u>	class , id , offset , xml:base , xml:id , xml:lang , xml:space	✓	animate , animateColor , animateMotion , animateTransform , desc , discard , metadata , set , switch , title
<u>svg</u>	baseProfile , class , contentScriptType , externalResourcesRequired , focusHighlight , focusable , height , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , playbackOrder , preserveAspectRatio , snapshotTime , syncBehaviorDefault , syncToleranceDefault , timelineBegin , version , viewBox , width , xml:base , xml:id , xml:lang , xml:space , zoomAndPan	✓	a , animate , animateColor , animateMotion , animateTransform , animation , audio , circle , defs , desc , discard , ellipse , font , font-face , foreignObject , g , handler , image , line , linearGradient , listener , metadata , path , polygon , polyline , prefetch , radialGradient , rect , script , set , solidColor , switch , text , textArea , title , use , video
<u>switch</u>	class , externalResourcesRequired , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , xml:base , xml:id , xml:lang , xml:space	✓	The 'switch' element may contain any element that its parent may contain.

<u>tbreak</u>	class , id , xml:base , xml:id , xml:lang , xml:space	✗	
<u>text</u>	class , editable , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , rotate , systemLanguage , transform , x , xml:base , xml:id , xml:lang , xml:space , y	✓	<code><text></code> , a , animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , tbreak , title , tspan
<u>textArea</u>	class , editable , focusHighlight , focusable , height , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , transform , width , x , xml:base , xml:id , xml:lang , xml:space , y	✓	<code><text></code> , a , animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , tbreak , title , tspan
<u>title</u>	class , id , xml:base , xml:id , xml:lang , xml:space	✗	<code><text></code>
<u>tspan</u>	class , focusHighlight , focusable , id , nav-down , nav-down-left , nav-down-right , nav-left , nav-next , nav-prev , nav-right , nav-up , nav-up-left , nav-up-right , requiredExtensions , requiredFeatures , requiredFonts , requiredFormats , systemLanguage , xml:base , xml:id , xml:lang , xml:space	✓	<code><text></code> , a , animate , animateColor , animateMotion , animateTransform , desc , discard , handler , metadata , set , switch , tbreak , title , tspan

use

[class](#),
[externalResourcesRequired](#),
[focusHighlight](#), [focusable](#), [id](#),
[nav-down](#), [nav-down-left](#), [nav-down-right](#), [nav-left](#), [nav-next](#),
[nav-prev](#), [nav-right](#), [nav-up](#),
[nav-up-left](#), [nav-up-right](#),
[requiredExtensions](#),
[requiredFeatures](#),
[requiredFonts](#),
[requiredFormats](#),
[systemLanguage](#), [transform](#),
[x](#), [xlink:actuate](#), [xlink:arcrole](#),
[xlink:href](#), [xlink:role](#), [xlink:show](#), [xlink:title](#), [xlink:type](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#), [y](#)



[animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[desc](#), [discard](#),
[handler](#), [metadata](#),
[set](#), [switch](#), [title](#)

video

[begin](#), [class](#), [dur](#), [end](#),
[externalResourcesRequired](#),
[fill](#), [focusHighlight](#), [focusable](#),
[height](#), [id](#), [initialVisibility](#), [max](#),
[min](#), [nav-down](#), [nav-down-left](#),
[nav-down-right](#), [nav-left](#), [nav-next](#), [nav-prev](#), [nav-right](#), [nav-up](#), [nav-up-left](#), [nav-up-right](#),
[overlay](#), [preserveAspectRatio](#),
[repeatCount](#), [repeatDur](#),
[requiredExtensions](#),
[requiredFeatures](#),
[requiredFonts](#),
[requiredFormats](#), [restart](#),
[syncBehavior](#), [syncMaster](#),
[syncTolerance](#),
[systemLanguage](#), [transform](#),
[transformBehavior](#), [type](#),
[width](#), [x](#), [xlink:actuate](#), [xlink:arcrole](#), [xlink:href](#), [xlink:role](#),
[xlink:show](#), [xlink:title](#), [xlink:type](#), [xml:base](#), [xml:id](#), [xml:lang](#), [xml:space](#), [y](#)

M

[animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[desc](#), [discard](#),
[handler](#), [metadata](#),
[set](#), [switch](#), [title](#)

L Attribute and Property Tables

Contents

- L.1 [Property Table](#)
- L.2 [Attribute Table](#)

This appendix lists the attributes and properties defined in this specification. While both attributes and properties are set on elements using the XML attributes syntax, they are conceptually different and therefore listed separately. Attributes are straightforward XML attributes that generally capture information specific to a given element, such as for instance the radius of a circle or the height of a text area. Properties add to that the fact that they capture CSS information, and therefore behave as is the equivalent CSS property had been set on the same element using a style sheet. SVG Tiny 1.2 does not require support for a style sheet language, but should the user agent use one these properties are also usable for use with it.

The following legend details how these two tables should be read.

Property or Attribute

The name of the property or attribute being described.

Value

For each property or attribute, the values that it accepts.

Elements

For each attribute, the elements that it may occur on.

ani.

This column indicates whether or not the given attribute or property may be animated. There are two options:



This attribute or property may be animated.



This attribute or property may not be animated.

inh.

This column indicates whether or not the given attribute or property is inheritable. There are two options:



This attribute or property may be inherited.



This attribute or property may not be inherited.

<text>

Indicates that character data can occur as the value of a given attribute.

L.1 Property Table

Property	ani.	inh.	Value
audio-level	✓	✗	'inherit' <number>
color	✓	✓	'inherit' <color>
color-rendering	✓	✓	'auto' 'optimizeSpeed' 'optimizeQuality' 'inherit'

display	✓	✗	'inline' 'block' 'list-item' 'run-in' 'compact' 'marker' 'table' 'inline-table' 'table-row-group' 'table-header-group' 'table-footer-group' 'table-row' 'table-column-group' 'table-column' 'table-cell' 'table-caption' 'none' 'inherit'
display-align	✓	✓	'auto' 'before' 'center' 'after' 'inherit'
fill	✓	✓	'inherit' <paint>
fill-opacity	✓	✓	'inherit' <number>
fill-rule	✓	✓	'inherit' 'nonzero' 'evenodd'
font-family	✓	✓	'inherit' <FontFamilyValue>
font-size	✓	✓	'inherit' <FontSizeValue>
font-style	✓	✓	'normal' 'italic' 'oblique' 'inherit'
font-variant	✓	✓	'normal' 'small-caps' 'inherit'
font-weight	✓	✓	'normal' 'bold' 'bolder' 'lighter' '100' '200' '300' '400' '500' '600' '700' '800' '900' 'inherit'
image-rendering	✓	✓	'auto' 'optimizeSpeed' 'optimizeQuality' 'inherit'
line-increment	✓	✓	'auto' 'inherit' <number>
opacity	✓	✗	'inherit' <number>
pointer-events	✓	✓	'visiblePainted' 'visibleFill' 'visibleStroke' 'visible' 'painted' 'fill' 'stroke' 'all' 'none' 'inherit'
shape-rendering	✓	✓	'auto' 'optimizeSpeed' 'crispEdges' 'geometricPrecision' 'inherit'
solid-color	✓	✗	'inherit' <color>
solid-opacity	✓	✗	'inherit' <number>
stop-color	✓	✗	'inherit' <color>
stop-opacity	✓	✗	'inherit' <number>
stroke	✓	✓	'inherit' <paint>
stroke-dasharray	✓	✓	'inherit' 'none' '<string>'
stroke-dashoffset	✓	✓	'inherit' <length>
stroke-linecap	✓	✓	'butt' 'round' 'square' 'inherit'
stroke-linejoin	✓	✓	'miter' 'round' 'bevel' 'inherit'
stroke-miterlimit	✓	✓	'inherit'

stroke-opacity	✓	✓	'inherit' <number>
stroke-width	✓	✓	'inherit'
text-anchor	✓	✓	'start' 'middle' 'end' 'inherit'
text-rendering	✓	✓	'auto' 'optimizeSpeed' 'optimizeLegibility' 'geometricPrecision' 'inherit'
vector-effect	✓	✗	'none' 'non-scaling-stroke' 'inherit'
viewport-fill	✓	✗	'inherit' <paint>
viewport-fill-opacity	✓	✗	'inherit' <number>
visibility	✓	✓	'visible' 'hidden' 'inherit'

L.2 Attribute Table

Attribute	ani.	inh.	Value	Elements
accent-height	✗	✗	<number>	font-face
accumulate	✗	✗	'none' 'sum'	animate , animateColor , animateMotion , animateTransform
additive	✗	✗	'replace' 'sum'	animate , animateColor , animateMotion , animateTransform
alphabetic	✗	✗	<number>	font-face
arabic-form	✗	✗	<text>	glyph
ascent	✗	✗	<number>	font-face
attributeName	✗	✗	<text>	animate , animateColor , animateTransform , set
attributeType	✗	✗	'XML' 'CSS' 'auto'	animate , animateColor , animateTransform , set
bandwidth	✗	✗	<number> 'auto'	prefetch
baseProfile	✗	✗	'none' 'tiny' 'basic' 'full'	svg

bbox	x	x	<text>	font-face
begin	x	x	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , discard , set , video
by	x	x	<text>	animate , animateColor , animateMotion , animateTransform
calcMode	x	x	'discrete' 'linear' 'paced' 'spline'	animate , animateColor , animateMotion , animateTransform
cap-height	x	x	<number>	font-face
class	x	x	<text>	a , animate , animateColor , animateMotion , animateTransform , animation , audio , circle , defs , desc , discard , ellipse , font , font-face , font-face-name , font-face-src , font- face-uri , foreignObject , g , glyph , handler , hkern , image , line , linearGradient , listener , metadata , missing-glyph , mpath , path , polygon , polyline , prefetch , radialGradient , rect , script , set ,

				solidColor , stop , svg , switch , tbreak , text , textArea , title , tspan , use , video
contentScriptType	✗	✗	<ContentType>	svg
cx	✓	✗	<coordinate>	circle , ellipse , radialGradient
cy	✓	✗	<coordinate>	circle , ellipse , radialGradient
d	✓	✗	<PathData>	glyph , missing-glyph , path
defaultAction	✗	✗	'perform' 'cancel'	listener
descent	✗	✗	<number>	font-face
dur	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
editable	✗	✗	'none' 'simple'	text , textArea
end	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
ev:event	✗	✗	<XML-Name>	handler
event	✗	✗	<XML-Name>	listener
externalResourcesRequired	✗	✗	<Boolean>	a , animation , audio , font , font-face , font-face-uri , foreignObject , g , handler , image , script , svg , switch , use , video

fill	✗	✗	'remove' 'freeze'	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
focusHighlight	✓	✗	'auto' 'none'	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
focusable	✓	✗	'auto' <Boolean>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
font-family	✗	✗	<text>	font-face
font-stretch	✗	✗	<text>	font-face
font-style	✗	✗	<text>	font-face
font-variant	✗	✗	<text>	font-face
font-weight	✗	✗	<text>	font-face
from	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform
g1	✗	✗	<text>	hkern
g2	✗	✗	<text>	hkern
glyph-name	✗	✗	<text>	glyph
gradientUnits	✓	✗	'userSpaceOnUse' 'objectBoundingBox'	linearGradient , radialGradient

handler	✗	✗	<iri>	listener
hanging	✗	✗	<number>	font-face
height	✓	✗	<length>	animation , foreignObject , image , rect , svg , video
height	✓	✗	<length> 'auto'	textArea
horiz-adv-x	✗	✗	<number>	font , glyph , missing-glyph
horiz-origin-x	✗	✗	<number>	font
id	✗	✗	<ID>	a , animate , animateColor , animateMotion , animateTransform , animation , audio , circle , defs , desc , discard , ellipse , font , font-face , font-face-name , font-face-src , font- face-uri , foreignObject , g , glyph , handler , hkern , image , line , linearGradient , listener , metadata , missing-glyph , mpath , path , polygon , polyline , prefetch , radialGradient , rect , script , set , solidColor , stop , svg , switch , tbreak , text , textArea , title , tspan , use , video

ideographic	✗	✗	<number>	font-face
initialVisibility	✗	✗	'whenStarted' 'always'	animation , video
k	✗	✗	<number>	hkern
keyPoints	✗	✗	<text>	animateMotion
keySplines	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform
keyTimes	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform
lang	✗	✗	<LanguageIDs>	glyph
mathematical	✗	✗	<number>	font-face
max	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
mediaCharacterEncoding	✗	✗	<text>	prefetch
mediaContentEncodings	✗	✗	<text>	prefetch
mediaSize	✗	✗	<number>	prefetch
mediaTime	✗	✗	<text>	prefetch
min	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
name	✗	✗	<text>	font-face-name

nav-down	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-down-left	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-down-right	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-left	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-next	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video

nav-prev	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-right	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-up	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-up-left	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video
nav-up-right	✓	✗	<Focus>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , svg , switch , text , textArea , tspan , use , video

observer	✗	✗	<iri>	listener
offset	✓	✗	<number>	stop
opacity	✓	✗	'inherit' <number>	image
origin	✗	✗	<text>	animateMotion
overlay	✗	✗	'none' 'top'	video
overline-position	✗	✗	<number>	font-face
overline-thickness	✗	✗	<number>	font-face
panose-1	✗	✗	<text>	font-face
path	✗	✗	<text>	animateMotion
pathLength	✓	✗	<number>	path
phase	✗	✗	'default' 'capture'	listener
playbackOrder	✗	✗	'all' 'forwardOnly'	svg
points	✓	✗	<Points>	polygon , polyline
preserveAspectRatio	✓	✗	<text>	animation , image , svg , video
propagate	✗	✗	'continue' 'stop'	listener
r	✓	✗	<length>	circle , radialGradient
repeatCount	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
repeatDur	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video

requiredExtensions	x	x	<list of iris>	a , animation , audio , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , switch , text , textArea , tspan , use , video
requiredFeatures	x	x	<list of iris>	a , animation , audio , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , switch , text , textArea , tspan , use , video
requiredFonts	x	x	<list of FontFamilyValues>	a , animation , audio , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , switch , text , textArea , tspan , use , video
requiredFormats	x	x	<FormatList>	a , animation , audio , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , switch , text , textArea , tspan , use , video

restart	✗	✗	'always' 'never' 'whenNotActive'	animate , animateColor , animateMotion , animateTransform , animation , audio , set , video
rotate	✗	✗	<text>	animateMotion
rotate	✓	✗	<list of Numbers>	text
rx	✓	✗	<length>	ellipse , rect
ry	✓	✗	<length>	ellipse , rect
slope	✗	✗	<number>	font-face
snapshotTime	✗	✗	'none' <Clock-value>	svg
stemh	✗	✗	<number>	font-face
stemv	✗	✗	<number>	font-face
strikethrough-position	✗	✗	<number>	font-face
strikethrough-thickness	✗	✗	<number>	font-face
syncBehavior	✗	✗	'canSlip' 'locked' 'independent' 'default'	animation , audio , video
syncBehaviorDefault	✗	✗	'canSlip' 'locked' 'independent' 'inherit'	svg
syncMaster	✗	✗	<Boolean>	animation , audio , video
syncTolerance	✗	✗	<Clock-value> 'default'	animation , audio , video
syncToleranceDefault	✗	✗	<Clock-value> 'inherit'	svg

systemLanguage	✗	✗	<LanguageIDs>	a , animation , audio , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , switch , text , textArea , tspan , use , video
target	✓	✗	'_replace' '_self' '_parent' '_top' '_blank' <XML- Name>	a
target	✗	✗	<iri>	listener
timelineBegin	✗	✗	'onLoad' 'onStart'	svg
to	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform , set
transform	✓	✗	<transform-list>	a , animation , circle , ellipse , foreignObject , g , image , line , path , polygon , polyline , rect , switch , text , textArea , use , video
transformBehavior	✗	✗	'geometric' 'pinned' 'pinned90' 'pinned180' 'pinned270'	video
type	✗	✗	<ContentType>	audio , handler , image , script , video
type	✗	✗	'translate' 'scale' 'rotate' 'skewX' 'skewY'	animateTransform
u1	✗	✗	<text>	hkern

u2	✗	✗	<text>	hkern
underline-position	✗	✗	<number>	font-face
underline-thickness	✗	✗	<number>	font-face
unicode	✗	✗	<text>	glyph
unicode-range	✗	✗	<text>	font-face
units-per-em	✗	✗	<number>	font-face
values	✗	✗	<text>	animate , animateColor , animateMotion , animateTransform
version	✗	✗	'1.0' '1.1' '1.2'	svg
viewBox	✓	✗	<text>	svg
width	✓	✗	<length>	animation , foreignObject , image , rect , svg , video
width	✓	✗	<length> 'auto'	textArea
widths	✗	✗	<text>	font-face
x	✓	✗	<coordinate>	animation , foreignObject , image , rect , textArea , use , video
x-height	✗	✗	<number>	font-face
x1	✓	✗	<coordinate>	line , linearGradient
x2	✓	✗	<coordinate>	line , linearGradient
x	✓	✗	<list of Coordinates >	text

xlink:actuate	✗	✗	'onLoad'	animate , animateColor , animateMotion , animateTransform , animation , audio , discard , font-face- uri , foreignObject , handler , image , mpath , prefetch , script , set , use , video
xlink:actuate	✗	✗	'onRequest'	a
xlink:arcrole	✗	✗	<iri>	a , animate , animateColor , animateMotion , animateTransform , animation , audio , discard , font-face- uri , foreignObject , handler , image , mpath , prefetch , script , set , use , video
xlink:href	✓	✗	<iri>	a , animate , animateColor , animateMotion , animateTransform , animation , audio , discard , font-face- uri , foreignObject , handler , image , mpath , prefetch , script , set , use , video

xlink:role	x	x	<iri>	a , animate , animateColor , animateMotion , animateTransform , animation , audio , discard , font-face- uri , foreignObject , handler , image , mpath , prefetch , script , set , use , video
xlink:show	x	x	'other'	animate , animateColor , animateMotion , animateTransform , discard , font-face- uri , handler , mpath , prefetch , script , set
xlink:show	x	x	'embed'	animation , audio , foreignObject , image , use , video
xlink:show	x	x	'new' 'replace'	a
xlink:title	x	x	<text>	a , animate , animateColor , animateMotion , animateTransform , animation , audio , discard , font-face- uri , foreignObject , handler , image , mpath , prefetch , script , set , use , video

xlink:type



'simple'

[a](#), [animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[animation](#), [audio](#),
[discard](#), [font-face-
uri](#), [foreignObject](#),
[handler](#), [image](#),
[mpath](#), [prefetch](#),
[script](#), [set](#), [use](#),
[video](#)

xml:base



[<iri>](#)

[a](#), [animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[animation](#), [audio](#),
[circle](#), [defs](#), [desc](#),
[discard](#), [ellipse](#),
[font](#), [font-face](#),
[font-face-name](#),
[font-face-src](#), [font-
face-uri](#),
[foreignObject](#), [g](#),
[glyph](#), [handler](#),
[hkern](#), [image](#),
[line](#),
[linearGradient](#),
[listener](#),
[metadata](#),
[missing-glyph](#),
[mpath](#), [path](#),
[polygon](#), [polyline](#),
[prefetch](#),
[radialGradient](#),
[rect](#), [script](#), [set](#),
[solidColor](#), [stop](#),
[svg](#), [switch](#),
[tbreak](#), [text](#),
[textArea](#), [title](#),
[tspan](#), [use](#), [video](#)

xml:id

x **x** [<ID>](#)

[a](#), [animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[animation](#), [audio](#),
[circle](#), [defs](#), [desc](#),
[discard](#), [ellipse](#),
[font](#), [font-face](#),
[font-face-name](#),
[font-face-src](#), [font-
face-uri](#),
[foreignObject](#), [g](#),
[glyph](#), [handler](#),
[hkern](#), [image](#),
[line](#),
[linearGradient](#),
[listener](#),
[metadata](#),
[missing-glyph](#),
[mpath](#), [path](#),
[polygon](#), [polyline](#),
[prefetch](#),
[radialGradient](#),
[rect](#), [script](#), [set](#),
[solidColor](#), [stop](#),
[svg](#), [switch](#),
[tbreak](#), [text](#),
[textArea](#), [title](#),
[tspan](#), [use](#), [video](#)

xml:lang

x **x** [<LanguageID>](#)

[a](#), [animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[animation](#), [audio](#),
[circle](#), [defs](#), [desc](#),
[discard](#), [ellipse](#),
[font](#), [font-face](#),
[font-face-name](#),
[font-face-src](#), [font-
face-uri](#),
[foreignObject](#), [g](#),

[glyph](#), [handler](#),
[hkern](#), [image](#),
[line](#),
[linearGradient](#),
[listener](#),
[metadata](#),
[missing-glyph](#),
[mpath](#), [path](#),
[polygon](#), [polyline](#),
[prefetch](#),
[radialGradient](#),
[rect](#), [script](#), [set](#),
[solidColor](#), [stop](#),
[svg](#), [switch](#),
[tbreak](#), [text](#),
[textArea](#), [title](#),
[tspan](#), [use](#), [video](#)

xml:space

✘ ✘

'default' | 'preserve'

[a](#), [animate](#),
[animateColor](#),
[animateMotion](#),
[animateTransform](#),
[animation](#), [audio](#),
[circle](#), [defs](#), [desc](#),
[discard](#), [ellipse](#),
[font](#), [font-face](#),
[font-face-name](#),
[font-face-src](#), [font-](#)
[face-uri](#),
[foreignObject](#), [g](#),
[glyph](#), [hkern](#),
[image](#), [line](#),
[linearGradient](#),
[listener](#),
[metadata](#),
[missing-glyph](#),
[mpath](#), [path](#),
[polygon](#), [polyline](#),
[prefetch](#),
[radialGradient](#),
[rect](#), [set](#),
[solidColor](#), [stop](#),

				svg , switch , tbreak , text , textArea , title , tspan , use , video
xml:space	✗	✗	'preserve'	handler , script
y	✓	✗	<coordinate>	animation , foreignObject , image , rect , textArea , use , video
y1	✓	✗	<coordinate>	line , linearGradient
y2	✓	✗	<coordinate>	line , linearGradient
y	✓	✗	<list of Coordinates>	text
zoomAndPan	✗	✗	'disable' 'magnify'	svg

M Media Type registration for image/svg+xml

Contents

- M.1 [Introduction](#)
- M.2 [Registration of Media Type image/svg+xml](#)

This appendix is normative.

M.1 Introduction

This appendix registers a new MIME media type, "image/svg+xml" in conformance with [RegMedia](#) and [W3CRegMedia](#).

M.2 Registration of Media Type image/svg+xml

MIME media type name:

image

MIME subtype name:

svg+xml

Required parameters:

None.

Optional parameters:

None

The encoding of an SVG document shall be determined by the XML encoding declaration. This has identical semantics to the application/xml media type in the case where the charset parameter is omitted, as specified in [RFC3023](#) sections 8.9, 8.10 and 8.11.

Encoding considerations:

Same as for application/xml. See [RFC3023](#), section 3.2.

Restrictions on usage:

None

Security considerations:

As with other XML types and as noted in [RFC3023](#) section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in

constrained environments to fail.

SVG documents may be transmitted in compressed form using gzip compression. For systems which employ MIME-like mechanisms, such as HTTP, this is indicated by the Content-Transfer-Encoding header; for systems which do not, such as direct filesystem access, this is indicated by the filename extension and by the Macintosh File Type Codes. In addition, gzip compressed content is readily recognised by the initial byte sequence as described in [RFC1952](#) section 2.3.1.

Several SVG elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986](#), section 7, should be considered.

In common with HTML, SVG documents may reference external media such as images, audio, video, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply.

In addition, because of the extensibility features for SVG and of XML in general, it is possible that "image/svg+xml" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the SVG namespace and shall be ignored. Only in the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Interoperability considerations:

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the SVG namespace and in other namespaces.

Because SVG is extensible, conformant "image/svg+xml" processors must expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

SVG has a published Test Suite and associated implementation report showing which implementations passed which tests at the time of the report. This information is periodically updated as new tests are added or as implementations improve.

Published specification:

This media type registration is extracted from Appendix M of the [SVG 1.2 Tiny specification](#).

Additional information:

Person & email address to contact for further information:

Dean Jackson, Chris Lilley (member-svg-media-type@w3.org).

Intended usage:

COMMON

Author/Change controller:

The SVG specification is a work product of the World Wide Web Consortium's SVG Working Group. The W3C has change control over these specifications.

N RelaxNG Schema for SVG Tiny 1.2

The schema for SVG Tiny 1.2 is written in [RelaxNG \[RelaxNG\]](#), a namespace-aware schema language that uses the datatypes from [XML Schema Part 2 \[Schema2\]](#). This allows namespaces and modularity to be much more naturally expressed than using DTD syntax. The RelaxNG schema for SVG Tiny 1.2 may be imported by other RelaxNG schemas, or combined with other schemas in other languages into a multi-namespace, multi-grammar schema using [Namespace-based Validation Dispatching Language \[NVDL\]](#).

Unlike a DTD, the schema used for validation is not hardcoded into the document instance. There is no equivalent to the DOCTYPE declaration. Simply point your editor or other validation tool to the IRI of the schema (or your local cached copy, as you prefer).

The schema is available in an unchanging *dated* version, considered to be normative at the time of publication, and the *latest* version (possibly incorporating bug fixes made since publication) which will evolve over time. At the date of publication, both versions are identical.

The driver schema is available as a [latest version](#) and also the [dated version for this Working Draft](#). A zip file of all the schema modules is available for download in [latest](#) and [dated](#) versions also.

O References

Contents

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

O.1 Normative references

[ATAG]

[Authoring Tool Accessibility Guidelines 1.0](#), J. Trevisanus, J. Richards, I. Jacobs, C. McCathieNeville, eds. World Wide Web Consortium, 03 February 2000.

Latest version available at <http://www.w3.org/TR/ATAG10/>

[AWWW]

[Architecture of the World Wide Web, Volume One](#), I. Jacobs, N. Walsh, editors. World Wide Web Consortium, 15 December 2004.

Latest version available at <http://www.w3.org/TR/webarch/>

[BCP 47]

ETF BCP 47, currently represented by RFC 3066 "Tags for the Identification of Languages", H. Alvestrand, January 2001

Available at <http://www.ietf.org/rfc/rfc3066.txt>.

[COLORIMETRY]

[Colorimetry, Third Edition](#), Commission Internationale de l'Eclairage, CIE Publication 15:2004, ISBN 3-901-906-33-9.

Available at <http://www.cie.co.at/publ/abst/15-2004.html>.

[CHARMOD]

[Character Model for the World Wide Web 1.0: Fundamentals](#), M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin, eds.. World Wide Web Consortium, 15 February 2005.

Latest version available at <http://www.w3.org/TR/charmod/>

[CHARMOD-RI]

[Character Model for the World Wide Web 1.0: Resource Identifiers](#), M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin, eds.. World Wide Web Consortium, *work in progress* 22 November 2004

Latest version available at <http://www.w3.org/TR/charmod-resid/>

[CSS2]

[Cascading Style Sheets, level 2](#), B. Bos, H. W. Lie, C. Lilley, I. Jacobs, eds. World Wide Web Consortium, 12 May 1998.

Latest version available at <http://www.w3.org/TR/REC-CSS2/>.

[DOM3]

[Document Object Model \(DOM\) Level 3 Core Specification](#), A. Le Hors, P. Le Hégarret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, eds. World Wide Web Consortium, 07 April 2004

Latest version available at <http://www.w3.org/TR/DOM-Level-3-Core/>

[DOM3Events]

[Document Object Model \(DOM\) Level 3 Events Specification](#), P. Le Hégarret, T. Pixley, eds. World Wide Web Consortium, *work in progress* 07 November 2003.

Latest version available at <http://www.w3.org/TR/DOM-Level-3-Events>

[EBNF]

[Extended Backus-Naur form \(EBNF\)](#) is defined by ISO/IEC 14977 : 1996(E). This specification uses the subset of EBNF used in the grammar of XML, which is available at <http://www.w3.org/TR/REC-xml/#sec-notation>

[JPEG]

ISO/IEC 10918. *Information Technology – Digital Compression And Coding Of Continuous-tone Still Images International Organization for Standardization (ISO)*. The Requirements And Guidelines are also available as [ITU T.81](#)

[JFIF]

[JPEG File Interchange Format, version 1.02](#). E. Hamilton, ed. Independent JPEG Group, 01 September 1992. Available at <http://www.w3.org/Graphics/JPEG/jfif3.pdf>

[NVDL]

[Document Schema Definition Languages \(DSDL\) — Part 4: Namespace-based Validation Dispatching Language — NVDL. ISO/IEC FCD 19757-4](#)

[OMG IDL]

"OMG IDL Syntax and Semantics" defined in [The Common Object Request Broker: Architecture and Specification, version 2](#), Object Management Group. The latest version of CORBA version 2.0 is available at http://www.omg.org/technology/documents/formal/corba_2.htm.

[OpenGIS Coordinate Systems]

[Recommended Definition Data for Coordinate Reference Systems and Coordinate Transformations Version 1.0.1](#). OpenGIS Recommendation

[PNG]

[Portable Network Graphics \(PNG\) Specification \(Second Edition\)](#)

Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E)

David Duce editor, 10 November 2003.

Latest version available at <http://www.w3.org/TR/PNG/>.

[QAF-SPEC]

[QA Framework: Specification Guidelines](#), Karl Dubost, Lynne Rosenthal, Dominique Hazaël-Massieux, Lofton Henderson. World Wide Web Consortium, 17 August 2005 Latest version available at <http://www.w3.org/TR/qaframe-spec/>.

[RelaxNG]

[Document Schema Definition Languages \(DSDL\) — Part 2: Regular grammar- based validation — RELAX NG. ISO/IEC FDIS 19757-2:2002\(E\)](#), J. Clark, 村田 真 (Murata M.), eds., 12 December 2002.

Available at http://www.y12.doe.gov/sgml/sc34/document/0362_files/relaxng-is.pdf

[RFC1951]

[DEFLATE Compressed Data Format Specification version 1.3](#), P. Deutsch, May 1996.

Available at <http://www.ietf.org/rfc/rfc1952.txt>.

[RFC1952]

[GZIP file format specification version 4.3](#), P. Deutsch, May 1996.

Available at <http://www.ietf.org/rfc/rfc1952.txt>.

[RFC2046]

[Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types](#), N. Freed and N. Borenstein, November 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

Available at <http://www.ietf.org/rfc/rfc2046.txt>.

[RFC2119]

[Key words for use in RFCs to Indicate Requirement Levels](#), S. Bradner, March 1997.

Available at <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC2397]

[The "data" URL scheme](#), L. Masinter, August 1998.

Available at <http://www.ietf.org/rfc/rfc2397.txt>.

[RFC2616]

[Hypertext Transfer Protocol -- HTTP/1.1](#), R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999. This RFC obsoletes RFC 2068.

Available at <http://www.ietf.org/rfc/rfc2616.txt>.

[RFC2781]

[UTF-16, an encoding of ISO 10646](#), P. Hoffman, F. Yergeau, February 2000.

Available at <http://www.ietf.org/rfc/rfc2781.txt>

[RFC3023]

[XML Media Types](#), M. Murata, S. St.Laurent, D. Kohn, January 2001.

Available at <http://www.ietf.org/rfc/rfc3023.txt>. Note that a revision to RFC3023 is in preparation.

[RFC3629]

[UTF-8, a transformation format of ISO 10646](#), F. Yergeau, November 2003. Note that this RFC obsoletes RFC 2044 and RFC 2279

Available at <http://www.ietf.org/rfc/rfc3629.txt>.

[RFC3986]

[Uniform Resource Identifiers \(URI\): Generic Syntax](#), T. Berners-Lee, R. Fielding, L. Masinter, January 2005. Note that RFC 3986 updates RFC1738 and obsoletes RFC 2732, RFC 2396, and RFC1808.

Available at <http://www.ietf.org/rfc/rfc3986.txt>.

[RFC3987]

[Internationalized Resource Identifiers \(IRIs\)](#). M. Duerst, M. Suignard, January 2005.

Available at <http://www.ietf.org/rfc/rfc3987.txt>

[SMIL 2.0]

[Synchronized Multimedia Integration Language \(SMIL 2.0\) - \[Second Edition\]](#). J. Ayars, D. Bulterman et. al.. World Wide Web Consortium, 07 January 2005.

Available at <http://www.w3.org/TR/2005/REC-SMIL2-20050107/>

[SRGB]

[IEC 61966-2-1 \(1999-10\) - "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB](#), ISBN: 2-8318-4989-6 - ICS codes: 33.160.60, 37.080 - TC 100 - 51 pp.

Available at: <http://domino.iec.ch/webstore/webstore.nsf/artnum/025408>.

[SVG M11]

[Mobile SVG Profiles: SVG Tiny and SVG Basic](#). T. Capin, ed. World Wide Web Consortium, 14 January 2003.

Latest version available at: <http://www.w3.org/TR/SVGMobile11/>

[UAAG]

[User Agent Accessibility Guidelines 1.0](#), I. Jacobs, J. Gunderson, E. Hansen, eds. World Wide Web Consortium, 17 December 2002.

Latest version available at <http://www.w3.org/TR/UAAG10/>

[UNICODE]

[The Unicode Standard](#), Version 4.0.0 or later. The Unicode Consortium. The Unicode Standard, Version 4.0.0 is defined by "The Unicode Standard, Version 4.0" (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1).

Refer to <http://www.unicode.org/unicode/standard/versions/>.

[XLINK]

[XML Linking Language \(XLink\)](#), S. DeRose, E. Maler, D. Orchard, eds.. World Wide Web Consortium, 27 June 2001.

Latest version available at: <http://www.w3.org/TR/xlink/>

[XML10]

[Extensible Markup Language \(XML\) 1.0 \(Third Edition\)](#), T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, eds. World Wide Web Consortium, 04 February 2004.

Latest version available at: <http://www.w3.org/TR/REC-xml>.

[XML11]

[Extensible Markup Language \(XML\) 1.1](#), T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan, eds. World Wide Web Consortium, 15 April 2004.

Latest version available at: <http://www.w3.org/TR/xml11/>

[XMLID]

[xml:id Version 1.0](#), J. Marsh, D. Veillard, N. Walsh, eds. World Wide Web Consortium, 9 September 2005.

Latest version available at: <http://www.w3.org/TR/xml-id/>

[XML-BASE]

[XML Base](#), J. Marsh, editor. World Wide Web Consortium, 27 June 2001.

Latest version available at: <http://www.w3.org/TR/xmlbase/>.

[XML-EVENTS]

[XML Events, An Events Syntax for XML](#), S. McCarron, S. Pemberton, T. V. Raman, editors, 14 October 2003.

Latest version available at <http://www.w3.org/TR/xml-events/>.

[XML-NS]

[Namespaces in XML 1.1](#), T. Bray, D. Hollander, A. Layman, R. Tobin, eds. World Wide Web Consortium, 4 February 2004.

Latest version available at: <http://www.w3.org/TR/xml-names11/>.

[XPTRFW]

[XPointer Framework](#), P. Grosso, E. Maler, J. Marsh, N. Walsh, eds. World Wide Web Consortium, 25 March 2003

Latest version available at: <http://www.w3.org/TR/xptr-framework/>

O.2 Informative references

[CHARMOD-norm]

[Character Model for the World Wide Web 1.0: Normalization](#), M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin, A. Phillips, eds.. World Wide Web Consortium *work in progress*, 25 February 2005.

Available at <http://www.w3.org/TR/charmod-norm/>

[DCORE]

[The Dublin Core](#). For more information, refer to <http://purl.org/DC>.

[ECMAScript]

[ECMA 262, ECMAScript Language Specification, 3rd Edition](#). European Computer Manufacturers Association.

Available at <http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>

[FOLEY-VANDAM]

Computer Graphics : Principles and Practice, Second Edition, James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips, Addison-Wesley, pp. 488-491.

[HTML4]

[HTML 4.01 Specification](#), D. Raggett, A. Le Hors, I. Jacobs, 24 December 1999.

Available at <http://www.w3.org/TR/html401/>. The Recommendation defines three document type definitions: Strict, Transitional, and Frameset, all reachable from the Recommendation.

[JAVA]

[The Java Language Specification](#), J. Gosling, B. Joy, and G. Steele, Authors. Addison-Wesley, September 1996. Available at <http://java.sun.com/docs/books/jls>

[JSR226]

[JSR 226: Scalable 2D Vector Graphics API for J2ME™](#), S. Chitturi, Specification Lead, 21 December 2004. Java Community Process,

Latest version available at <http://forum.nokia.com/java/jsr226>

[MATHML]

[Mathematical Markup Language \(MathML\) Version 2.0 \(Second Edition\)](#), D. Carlisle, P. Ion, R. Miner, N. Poppelier, eds.. World Wide Web Consortium, 21 October 2003.

Latest version available at <http://www.w3.org/TR/MathML2/>.

[RDF]

[RDF Primer](#), Frank Manola, Eric Miller, eds. World Wide Web Consortium, 10 February 2004.

Latest version available at <http://www.w3.org/TR/rdf-primer/>. This specification in turn links to [Concepts](#), [Syntax](#), [Semantics](#), [Vocabulary](#), and [Test Cases](#).

[Schema2]

[XML Schema Part 2: Datatypes Second Edition](#). P. Biron, A. Malhotra, eds. World Wide Web Consortium, 28 October 2004.

Latest version available at <http://www.w3.org/TR/xmlschema-2/>. See also [Processing XML 1.1 documents with XML Schema 1.0 processors](#).

[SVG11]

[Scalable Vector Graphics \(SVG\) 1.1](#), J. Ferraiolo, 藤沢 淳 (Fujisawa Jun), D. Jackson, eds. World Wide Web Consortium, 14 January 2003..

Latest version available at: <http://www.w3.org/TR/SVG11/>

[SVG-ACCESS]

[Accessibility Features of SVG](#), C. McCathieNevile, M. Koivunen, 7 August 2000.

Latest version available at <http://www.w3.org/TR/SVG-access/>.

[WAI]

[*The Web Accessibility Initiative*](#)

[WCAG]

[*Web Content Accessibility Guidelines 1.0*](#), W. Chisholm, G. Vanderheiden, I. Jacobs, eds. World Wide Web Consortium, 5-May-1999.

Latest version available at:

<http://www.w3.org/TR/WAI-WEBCONTENT/>.

[WebCGM]

[*WebCGM 1.0 Second Release*](#), D. Cruikshank et. al, eds. World Wide Web Consortium, 17 December 2001.

Latest version available at <http://www.w3.org/TR/REC-WebCGM/>

[XHTML]

[*XHTML\(tm\) 1.0: The Extensible HyperText Markup Language \(Second Edition\)*](#). World Wide Web Consortium, 1 August 2002.

Latest version available at <http://www.w3.org/TR/xhtml1/>.

[XHTMLplusMathMLplusSVG]

[*An XHTML + MathML + SVG Profile*](#), 石川 雅康 (Ishikawa Masayasu), editor. World Wide Web Consortium, *work in progress*, 9 August 2002.

Latest version available at: <http://www.w3.org/TR/XHTMLplusMathMLplusSVG>

[XSL]

[*Extensible Stylesheet Language \(XSL\) Version 1.0*](#), S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, eds. World Wide Web Consortium, 15 October 2001.

Latest version available at <http://www.w3.org/TR/xsl/>

[XSLT]

[*XSL Transformations \(XSLT\) Version 1.0*](#), J. Clark, editor. World Wide Web Consortium, 16 November 1999.

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

P Change History

A number of changes have been made to this Last Call Working Draft relative to the previous public Working Draft of SVG Tiny 1.2. These changes were made in response to last Call feedback. The changes are collected in a separate document together with the corresponding LastCall comment and the SVG Working Group's reply. [\[LastCall Comments\]](#)

Expanded Table of Contents

- 1 [Introduction](#)
 - 1.1 [About SVG](#)
 - 1.2 [SVG Tiny 1.2](#)
 - 1.2.1 [Modularization](#)
 - 1.2.2 [Element and Attribute collections](#)
 - 1.2.3 [Profiling the SVG specification](#)
 - 1.3 [Defining an SVG Tiny 1.2 document](#)
 - 1.4 [SVG MIME type, file name extension and Macintosh file type](#)
 - 1.5 [Compatibility with Other Standards Efforts](#)
 - 1.6 [Definitions](#)
- 2 [Concepts](#)
 - 2.1 [Explaining the name: SVG](#)
 - 2.1.1 [Scalable](#)
 - 2.1.2 [Vector](#)
 - 2.1.3 [Graphics](#)
 - 2.1.4 [XML](#)
 - 2.1.5 [Namespace](#)
 - 2.1.6 [Scriptable](#)
 - 2.2 [Important SVG concepts](#)
 - 2.2.1 [Graphical Objects](#)
 - 2.2.2 [Reuse](#)
 - 2.2.3 [Fonts](#)
 - 2.2.4 [Animation](#)
 - 2.3 [Options for using SVG in Web pages](#)
- 3 [Rendering Model](#)
 - 3.1 [Introduction](#)
 - 3.2 [The painters model](#)
 - 3.3 [Rendering Order](#)
 - 3.4 [Types of graphics elements](#)
 - 3.4.1 [Rendering shapes and text](#)
 - 3.4.2 [Rendering raster images](#)
 - 3.4.3 [Rendering video](#)
 - 3.5 [Object Opacity](#)
 - 3.6 [Parent Compositing](#)
- 4 [Basic Data Types and Color Keywords](#)
 - 4.1 [Basic Data Types](#)
- 5 [Document Structure](#)
 - 5.1 [Defining an SVG document fragment: the 'svg' element](#)
 - 5.1.1 [Overview](#)
 - 5.1.2 [The 'svg' element](#)
 - 5.2 [Grouping: the 'g' element](#)
 - 5.2.1 [Overview](#)
 - 5.2.2 [The 'g' element](#)
 - 5.3 [The 'defs' element](#)
 - 5.4 [The 'discard' element](#)
 - 5.5 [The 'desc' and 'title' elements](#)
 - 5.6 [The 'use' element](#)
 - 5.7 [The 'image' element](#)
 - 5.8 [Conditional processing](#)
 - 5.8.1 [Conditional processing overview](#)
 - 5.8.2 [The 'switch' element](#)
 - 5.8.3 [The 'requiredFeatures' attribute](#)
 - 5.8.4 [The 'requiredExtensions' attribute](#)
 - 5.8.5 [The 'systemLanguage' attribute](#)

- 5.8.6 [The 'requiredFormats' attribute](#)
 - 5.8.7 [The 'requiredFonts' attribute](#)
 - 5.9 [External Resources](#)
 - 5.9.1 [The 'externalResourcesRequired' attribute](#)
 - 5.9.2 [Progressive Rendering](#)
 - 5.9.3 [The 'prefetch' element](#)
 - 5.10 [Common attributes](#)
 - 5.10.1 [Attributes common to all elements: 'class', 'id', 'xml:id' and 'xml:base'](#)
 - 5.10.2 [The 'xml:lang' and 'xml:space' attributes](#)
 - 5.11 [Core Attribute Module](#)
 - 5.12 [Structure Module](#)
 - 5.13 [Conditional Processing Module](#)
 - 5.14 [Conditional Processing Attribute Module](#)
 - 5.15 [Image Module](#)
 - 5.16 [Prefetch Module](#)
 - 5.17 [Discard Module](#)
 - 5.18 [ExternalResourcesRequired Attribute Module](#)
 - 6 [Styling](#)
 - 6.1 [SVG's styling properties](#)
 - 6.2 [Usage scenarios for styling](#)
 - 6.3 [Specifying properties using the presentation attributes](#)
 - 6.4 [Styling with XSL](#)
 - 6.5 [Case sensitivity of property names and values](#)
 - 6.6 [Facilities from CSS and XSL used by SVG](#)
 - 6.7 [Property inheritance and computation](#)
 - 7 [Coordinate Systems, Transformations and Units](#)
 - 7.1 [Introduction](#)
 - 7.2 [The initial viewport](#)
 - 7.3 [The initial coordinate system](#)
 - 7.4 [Coordinate system transformations](#)
 - 7.5 [Nested transformations](#)
 - 7.6 [The transform attribute](#)
 - 7.6.1 [The TransformList value](#)
 - 7.7 [Constrained Transformations](#)
 - 7.7.1 [The User Transform](#)
 - 7.7.2 [ViewBox to Viewport transformation](#)
 - 7.7.3 [Element Transform Stack](#)
 - 7.7.4 [The Current Transform Matrix](#)
 - 7.7.5 [The ref\(\) transform value](#)
 - 7.8 [The viewBox attribute](#)
 - 7.9 [The preserveAspectRatio attribute](#)
 - 7.10 [Establishing a new viewport](#)
 - 7.11 [Units](#)
 - 7.12 [Object bounding box units](#)
 - 7.13 [Intrinsic Sizing Properties of the Viewport of SVG content](#)
 - 7.14 [Geographic Coordinate Systems](#)
 - 8 [Paths](#)
 - 8.1 [Introduction](#)
 - 8.2 [The 'path' element](#)
 - 8.2.1 [Animating path data](#)
 - 8.3 [Path data](#)
 - 8.3.1 [General information about path data](#)
 - 8.3.2 [The "moveto" commands](#)
 - 8.3.3 [The "closepath" command](#)
 - 8.3.4 [The "lineto" commands](#)
 - 8.3.5 [The Curve commands](#)
 - 8.3.6 [The Cubic Bézier curve commands](#)
 - 8.3.7 [The Quadratic Bézier curve commands](#)
 - 8.3.8 [The grammar for path data](#)
 - 8.4 [Distance along a path](#)

- 9 [Basic Shapes](#)
 - 9.1 [Introduction](#)
 - 9.2 [The 'rect' element](#)
 - 9.3 [The 'circle' element](#)
 - 9.4 [The 'ellipse' element](#)
 - 9.5 [The 'line' element](#)
 - 9.6 [The 'polyline' element](#)
 - 9.7 [The 'polygon' element](#)
 - 9.7.1 [The grammar for points specifications in 'polyline' and 'polygon' elements](#)
 - 9.8 [Shape Module](#)
- 10 [Text](#)
 - 10.1 [Introduction](#)
 - 10.2 [Characters and their corresponding glyphs](#)
 - 10.3 [Fonts, font tables and baselines](#)
 - 10.4 [The 'text' element](#)
 - 10.5 [The 'tspan' element](#)
 - 10.6 [Text layout](#)
 - 10.6.1 [Text layout introduction](#)
 - 10.6.2 [Relationship with bidirectionality](#)
 - 10.7 [Text rendering order](#)
 - 10.8 [Alignment properties](#)
 - 10.8.1 [Text alignment properties](#)
 - 10.9 [Font selection properties](#)
 - 10.10 [White space handling](#)
 - 10.11 [Text in an area](#)
 - 10.11.1 [Introduction to text in an area](#)
 - 10.11.2 [The 'textArea' element](#)
 - 10.11.3 [The 'tbreak' element](#)
 - 10.11.4 [The 'line-increment' property](#)
 - 10.11.5 [The 'display-align' property](#)
 - 10.11.6 [Text in an area layout rules](#)
 - 10.12 [Editable Text Fields](#)
 - 10.12.1 [The editable attribute](#)
 - 10.13 [Text selection and clipboard operations](#)
 - 10.14 [Text Module](#)
- 11 [Painting: Filling, Stroking, Colors and Paint Servers](#)
 - 11.1 [Introduction](#)
 - 11.2 [Specifying paint](#)
 - 11.3 [Fill Properties](#)
 - 11.4 [Stroke Properties](#)
 - 11.5 [Non-Scaling Stroke](#)
 - 11.6 [Simple alpha compositing](#)
 - 11.6.1 [Compositing the currentColor value](#)
 - 11.7 [The 'viewport-fill' Property](#)
 - 11.8 [The 'viewport-fill-opacity' Property](#)
 - 11.9 [Controlling visibility](#)
 - 11.10 [Rendering hints](#)
 - 11.10.1 [The 'color-rendering' property](#)
 - 11.10.2 [The 'shape-rendering' property](#)
 - 11.10.3 [The 'text-rendering' property](#)
 - 11.10.4 [The 'image-rendering' property](#)
 - 11.11 [Inheritance of painting properties](#)
 - 11.12 [Object and group opacity: the 'opacity' property](#)
 - 11.13 [Color](#)
 - 11.13.1 [Syntax for color values](#)
 - 11.13.2 [HTML Color keywords](#)
 - 11.13.3 [System Paint](#)
 - 11.13.4 [The solidColor Element](#)
 - 11.13.5 [The SVG 'color' property](#)
 - 11.14 [Paint Servers](#)

- 11.15 [Gradients](#)
 - 11.15.1 [Linear gradients](#)
 - 11.15.2 [Radial gradients](#)
 - 11.15.3 [Gradient stops](#)
- 11.16 [Paint Attribute Module](#)
- 11.17 [Opacity Attribute Module](#)
- 11.18 [Graphics Attribute Module](#)
- 11.19 [Gradient Module](#)
- 11.20 [Solid Color Module](#)
- 12 [Multimedia](#)
 - 12.1 [Media elements](#)
 - 12.1.1 [Media timeline and document timeline](#)
 - 12.1.2 [Media availability](#)
 - 12.1.3 [Platform limits](#)
 - 12.2 [The 'audio' element](#)
 - 12.3 [The 'video' element](#)
 - 12.3.1 [Restricting the transformation of the 'video' element](#)
 - 12.3.2 [Restricting compositing of the 'video' element](#)
 - 12.3.3 [Examples](#)
 - 12.4 [The 'animation' element](#)
 - 12.5 [The audio-level property](#)
 - 12.6 [Attributes for run-time synchronization](#)
 - 12.7 [The 'initialVisibility' attribute](#)
 - 12.8 [Audio Module](#)
 - 12.9 [Video Module](#)
 - 12.10 [Animation Module](#)
- 13 [Interactivity](#)
 - 13.1 [Introduction](#)
 - 13.2 [Complete list of supported events](#)
 - 13.3 [User interface events](#)
 - 13.4 [Pointer events](#)
 - 13.5 [Text events](#)
 - 13.6 [Key events](#)
 - 13.7 [Event Dispatching](#)
 - 13.8 [Processing order for user interface events](#)
 - 13.9 [The 'pointer-events' property](#)
 - 13.10 [Magnification and panning](#)
 - 13.11 [Element focus](#)
 - 13.11.1 [The focusable attribute](#)
 - 13.12 [Navigation](#)
 - 13.12.1 [Navigation behavior](#)
 - 13.12.2 [Specifying navigation](#)
 - 13.12.3 [Specifying Focus Highlighting](#)
 - 13.12.4 [Obtaining and listening to focus programmatically](#)
- 14 [Linking](#)
 - 14.1 [References](#)
 - 14.1.1 [Overview](#)
 - 14.1.2 [IRIs and URIs](#)
 - 14.1.3 [Reference Restrictions](#)
 - 14.1.4 [IRI reference attributes](#)
 - 14.1.5 [Externally referenced documents](#)
 - 14.2 [Links out of SVG content: the 'a' element](#)
 - 14.3 [Linking into SVG content: IRI fragments and SVG views](#)
 - 14.3.1 [Introduction: IRI fragments and SVG views](#)
 - 14.3.2 [SVG fragment identifiers](#)
 - 14.4 [Hyperlinking Module](#)
 - 14.5 [XLink Attribute Module](#)
- 15 [Scripting](#)
 - 15.1 [Specifying the scripting language](#)
 - 15.1.1 [Specifying the default scripting language](#)

- A.1 [Introduction](#)
- A.2 [Overview of the SVG uDOM](#)
 - A.2.1 [Document Access](#)
 - A.2.2 [Tree Navigation](#)
 - A.2.3 [Element Creation](#)
 - A.2.4 [Element Addition](#)
 - A.2.5 [Element Removal](#)
 - A.2.6 [Attribute and Property Access](#)
 - A.2.7 [Attribute/Property Normalization](#)
 - A.2.8 [Text Node Access](#)
 - A.2.9 [Event Listener Registration and Removal](#)
 - A.2.10 [Animation](#)
 - A.2.11 [Java package naming](#)
- A.3 [Module: dom](#)
 - A.3.1 [DOMException](#)
 - A.3.2 [Node](#)
 - A.3.3 [Element](#)
 - A.3.4 [Document](#)
 - A.3.5 [DOMImplementation](#)
- A.4 [Module: events](#)
 - A.4.1 [EventTarget](#)
 - A.4.2 [EventListener](#)
 - A.4.3 [Event](#)
 - A.4.4 [OriginalEvent](#)
 - A.4.5 [MouseEvent](#)
 - A.4.6 [TextEvent](#)
 - A.4.7 [KeyboardEvent](#)
 - A.4.8 [TimeEvent](#)
 - A.4.9 [UIEvent](#)
 - A.4.10 [WheelEvent](#)
 - A.4.11 [ProgressEvent](#)
 - A.4.12 [ConnectionEvent](#)
- A.5 [Module: smil](#)
 - A.5.1 [ElementTimeControl](#)
- A.6 [Module: global](#)
 - A.6.1 [Global](#)
 - A.6.2 [GlobalException](#)
 - A.6.3 [Connection](#)
 - A.6.4 [Timer](#)
- A.7 [Module: svg](#)
 - A.7.1 [SVGException](#)
 - A.7.2 [SVGDocument](#)
 - A.7.3 [SVGElementInstance](#)
 - A.7.4 [SVGSVGElement](#)
 - A.7.5 [SVGRGBColor](#)
 - A.7.6 [SVGRect](#)
 - A.7.7 [SVGPoint](#)
 - A.7.8 [SVGPath](#)
 - A.7.9 [SVGMatrix](#)
 - A.7.10 [SVGLocatable](#)
 - A.7.11 [SVGLocatableElement](#)
 - A.7.12 [TraitAccess](#)
 - A.7.13 [Additional accessing rules](#)
 - A.7.14 [ElementTraversal](#)
 - A.7.15 [SVGElement](#)
 - A.7.16 [SVGAnimationElement](#)
 - A.7.17 [SVGVisualMediaElement](#)
 - A.7.18 [EventListenerInitializer2](#)
 - A.7.19 [SVGGlobal](#)
 - A.7.20 [AsyncStatusCallback](#)

