Scalable Vector Graphics (SVG) Tiny 1.2 Specification

W3C Recommendation *22 December 2008*

This version:
http://www.w3.org/TR/2008/REC-SVGTiny12-20081222/
Latest version:
http://www.w3.org/TR/SVGTiny12/
Previous version:
http://www.w3.org/TR/2008/PR-SVGTiny12-20081117/
Editors:
Ola Andersson (Ikivo) <ola.andersson@ikivo.com>
Robin Berjon (Expway) <robin.berjon@expway.fr>
Erik Dahlström (Opera Software) <ed@opera.com>
Andrew Emmons (BitFlash) <andrew.emmons@bitflash.com>
Jon Ferraiolo (Adobe Systems until May 2006) <jon.ferraiolo@adobe.com>
Anthony Grasso (Canon, Inc.) <anthony.grasso@cisra.canon.com.au>
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>
Cameron McCormack (Invited Expert) <cam@mcc.id.au>
Andreas Neumann (ETH Zurich)
Craig Northway (Canon, Inc.) <craign@cisra.canon.com.au>
Antoine Quint (Invited Expert) <aq@fuchsia-design.com>
Nandini Ramani (Sun Microsystems)
Doug Schepers (W3C) <schepers@w3.org>
Andrew Shellshear (Canon, Inc.)
Authors:
See author list.

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

This document is also available in these non-normative formats: a single-page version, a zip archive of HTML (without external dependencies), and a PDF. See also **translations**, noting that the English version of this specification is the only normative version.

Abstract

This specification defines the features and syntax for Scalable Vector Graphics (SVG) Tiny, Version 1.2, a language for describing two-dimensional vector graphics in XML, combined with raster graphics and multimedia. Its goal is to provide the ability to create a whole range of graphical content, from static images to animations to interactive Web applications. SVG 1.2 Tiny is a profile of SVG intended for implementation on a range of devices, from cellphones and PDAs to laptop and desktop computers, and thus includes a subset of the features included in SVG 1.1 Full, along with new features to extend the capabilities of SVG. Further extensions are planned in the form of modules which will be compatible with SVG 1.2 Tiny, and which when combined with this specification, will match and exceed the capabilities of SVG 1.1 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 the 22 December 2008 Recommendation of SVG Tiny 1.2.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The SVG Working Group working closely with the developer community, has produced an implementation report to prove the implementability of this specification. Previous drafts for this specification resulted in a number of comments which have been addressed by the SVG Working Group, with a Disposition of Comments available on the W3C SVG site. A list of changes made since the Proposed Recommendation Working Draft is available in Appendix T.

As described in the abstract, this specification represents the core for a set of modular extensions, but is named SVG Tiny for historical reasons, as a profile for mobile devices. Future versions of this specification will maintain backwards compatibility with previous versions of the language, in a continuing line of technology, but will bear the name "SVG Core" to represent this relationship.

Please send questions or comments regarding the SVG 1.2 Tiny specification 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.

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.

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

Authors

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

- 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 Çapin, 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.
- Erik Dahlström, Opera Software (*Working Group Chair*)
- 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)
- Andrew Emmons, BitFlash (*Working Group Chair*)
- Jerry Evans, Sun Microsystems
- Jon Ferraiolo, Adobe Systems Inc.
- 藤沢 淳 (FUJISAWA Jun), Canon
- Darryl Fuller, Schema Software
- Scott Furman, Netscape Communications Corporation
- Brent Getlin, Macromedia
- Diego Gibellino, Telecom Italia
- Christophe Gillette, Motorola (formerly BitFlash)
- Peter Graffagnino, Apple
- Rick Graham, BitFlash
- Anthony Grasso, Canon Information Systems Research Australia
- Niklas Hagelroth, Ikivo
- Vincent Hardy, Sun Microsystems Inc.
- 端山 貴也 (HAYAMA Takanari), KDDI Research Labs
- Scott Hayman, Research In Motion Limited
- Stephane Heintz, OpenText (formerly BitFlash)
- Lofton Henderson, OASIS
- Jan Christian Herlitz, Excosoft
- Ivan Herman, W3C
- Alan Hester, Xerox Corporation
- Olaf Hoffmann, Invited Expert
- 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.
- Yuri Khramov, Schema Software
- Kelvin Lawrence, IBM
- Håkon Lie, Opera
- Chris Lilley, W3C (*Working Group Chair*)
- Vincent Mahe, France Telecom
- Philip Mansfield, Schema Software
- Lee Martineau, Quickoffice
- Charles McCathieNevile, Opera Software
- Kevin McCluskey, Netscape Communications Corporation
- Cameron McCormack, Invited Expert
- 水口 充 (MINAKUCHI Mitsuru), Sharp Corporation
- Luc Minnebo, Agfa-Gevaert N.V.
- Jean-Claude Moissinac, Groupe des Ecoles des Télécommunications (GET)
- Tuan Nguyen, Microsoft Corporation
- Craig Northway, Canon Information Systems Research Australia
- 小野 修一郎 (ONO Shuichiro), Sharp Corporation
- Lars Piepel, Vodafone
- Antoine Quint, Fuchsia Design (formerly ILOG)
- णन्दिनि ऱमनि (Nandini Ramani), Sun Microsystems
- Bruno David Simões Rodrigues, Vodafone
- 相良 毅 (SAGARA Takeshi), KDDI Research Labs
- Troy Sandal, Visio Corporation
- Peter Santangeli, Macromedia
- Doug Schepers, W3C (formerly Vectoreal) (*W3C Team Contact*)
- Sebastian Schnitzenbaumer, SAP AG
- Haroon Sheikh, Corel Corporation
- Andrew Shellshear, Canon Inc.
- Brad Sipes, Ikivo
- Andrew Sledd, Ikivo
- Пётр Соротокин (Peter Sorotokin), Adobe Systems Inc.
- Gavriel State, Corel Corporation
- Robert Stevahn, Hewlett-Packard
- Timothy Thompson, Eastman Kodak
- 上田 宏高 (UEDA Hirotaka), Sharp Corporation
- Rick Yardumian, Canon Development Americas
- Charles Ying, Openwave Systems Inc.
- Shenxue Zhou, Quark
- Atanas Zlatinski, Samsung Electronics

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, but are greatly appreciated. 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, and from public feedback.

Table of contents

1 Introduction 1
1.1 About SVG 1
1.2 SVG Tiny 1.2 1
1.2.1 Profiling the SVG specification 2

1.3 Defining an SVG Tiny 1.2 document 2
1.4 SVG MIME type, file name extension and Macintosh file type 3
1.5 Compatibility with other standards efforts 3
1.6 Definitions 3
1.7 How to reference this specification 9
1.8 How to use this specification 9

2 Concepts 10
2.1 Explaining the name: SVG 10
2.1.1 Scalable 10
2.1.2 Vector 10
2.1.3 Graphics 10
2.1.4 XML 11
2.1.5 Namespace 11
2.1.6 Scriptable 11

2.2 Important SVG concepts 11
2.2.1 Graphical objects 11
2.2.2 Re-use 11
2.2.3 Fonts 11
2.2.4 Animation 11

2.3 Options for using SVG in Web pages 11

3 Rendering Model 13
3.1 Introduction 13
3.2 The painters model 13
3.3 Rendering order 13
3.4 Types of graphics elements 13
3.4.1 Rendering shapes and text 13
3.4.2 Rendering raster images 14
3.4.3 Rendering video 14

3.5 Object opacity 14
3.6 Parent compositing 14

4 Basic Data Types 15
5 Document Structure 19
5.1 Defining an SVG document fragment: the 'svg' element 19
5.1.1 Overview 19
5.1.2 The 'svg' element 21

5.2 Grouping: the 'g' element 25
5.2.1 Overview 25
5.2.2 The 'g' element 25

5.3 The 'defs' element 26
5.4 The 'discard' element 27
5.5 The 'title' and 'desc' elements 28
5.5.1 Applicable 'title' and 'desc' 29

5.5.2 Multiple 'title' and 'desc' elements . . . 30
5.5.3 User interface behavior for 'title' and 'desc' . . . 30

5.6 The 'use' element . . . 31
5.7 The 'image' element . . . 36
5.8 Conditional processing . . . 39
5.8.1 Conditional processing overview . . . 39
5.8.2 The 'switch' element . . . 40
5.8.3 The 'requiredFeatures' attribute . . . 41
5.8.4 The 'requiredExtensions' attribute . . . 42
5.8.5 The 'systemLanguage' attribute . . . 42
5.8.6 The 'requiredFormats' attribute . . . 43
5.8.7 The 'requiredFonts' attribute . . . 43

5.9 External resources . . . 44
5.9.1 The 'externalResourcesRequired' attribute . . . 44
5.9.2 Progressive rendering . . . 44
5.9.3 The 'prefetch' element . . . 48

5.10 Common attributes . . . 51
5.10.1 Attributes common to all elements . . . 51
5.10.2 Attributes for character-content elements . . . 54

6 Styling . . . 56
6.1 SVG's styling properties . . . 56
6.2 Usage scenarios for styling . . . 57
6.3 Specifying properties using the presentation attributes . . . 57
6.4 Styling with XSL . . . 58
6.5 Case sensitivity of property names and values . . . 59
6.6 Facilities from CSS and XSL used by SVG . . . 59
6.7 Property inheritance and computation . . . 59

7 Coordinate Systems, Transformations and Units . . . 60
7.1 Introduction . . . 60
7.2 The initial viewport . . . 60
7.3 The initial coordinate system . . . 61
7.4 Coordinate system transformations . . . 62
7.5 Nested transformations . . . 66
7.6 The 'transform' attribute . . . 67
7.6.1 The TransformList value . . . 68

7.7 Constrained transformations . . . 70
7.7.1 The user transform . . . 70
7.7.2 ViewBox to viewport transformation . . . 70
7.7.3 Element transform stack . . . 70
7.7.4 The current transformation matrix . . . 71
7.7.5 The TransformRef value . . . 71

7.8 The 'viewBox' attribute . . . 74
7.9 The 'preserveAspectRatio' attribute . . . 76
7.10 Establishing a new viewport . . . 77
7.11 Units . . . 78
7.12 Bounding box . . . 78
7.13 Object bounding box units . . . 81
7.14 Intrinsic sizing properties of the viewport of SVG content . . . 82
7.15 Geographic coordinate systems . . . 83
7.16 The 'svg:transform' attribute . . . 83

8 Paths 87
8.1 Introduction 87
8.2 The 'path' element 87
8.2.1 Animating path data........ 88

8.3 Path data 88
8.3.1 General information about path data........ 88
8.3.2 The "moveto" commands 89
8.3.3 The "closepath" command 89
8.3.4 The "lineto" commands 90
8.3.5 The Curve commands........ 90
8.3.6 The Cubic Bézier curve commands 90
8.3.7 The Quadratic Bézier curve commands 92
8.3.8 The grammar for path data 93

8.4 Distance along a path 95

9 Basic Shapes........ 96
9.1 Introduction 96
9.2 The 'rect' element........ 96
9.3 The 'circle' element 98
9.4 The 'ellipse' element 100
9.5 The 'line' element 101
9.6 The 'polyline' element........ 102
9.7 The 'polygon' element 104
9.7.1 The grammar for points specifications in 'polyline' and 'polygon' elements 105

10 Text 106
10.1 Introduction........ 106
10.2 Characters and their corresponding glyphs 107
10.3 Fonts, font tables and baselines 108
10.4 The 'text' element 109
10.5 The 'tspan' element........ 111
10.6 Text layout 112
10.6.1 Text layout introduction 112
10.6.2 Relationship with bidirectionality 113
10.6.3 The 'direction' property........ 115
10.6.4 The 'unicode-bidi' property 115

10.7 Text rendering order........ 115
10.8 Alignment properties 115
10.8.1 Text alignment properties 115

10.9 Font selection properties 116
10.10 White space handling 118
10.11 Text in an area........ 119
10.11.1 Introduction to text in an area 119
10.11.2 The 'textArea' element 120
10.11.3 The 'tbreak' element 121
10.11.4 The 'line-increment' property........ 122
10.11.5 The 'text-align' property 122
10.11.6 The 'display-align' property........ 123
10.11.7 Text in an area layout rules 123

10.12 Editable text fields........ 124
10.12.1 The 'editable' attribute 124

10.13 Text selection and clipboard operations 126
10.14 Text search 127

11 Painting: Filling, Stroking, Colors and Paint Servers 129
11.1 Introduction 129
11.2 Specifying paint 129
11.3 Fill properties 130
11.4 Stroke properties 132
11.5 Non-scaling stroke 135
11.6 Simple alpha compositing 136
11.6.1 Compositing the currentColor value 137

11.7 The 'viewport-fill' property 137
11.8 The 'viewport-fill-opacity' property 138
11.9 Controlling visibility and rendering 139
11.10 Rendering hints 140
11.10.1 The 'color-rendering' property 140
11.10.2 The 'shape-rendering' property 140
11.10.3 The 'text-rendering' property 141
11.10.4 The 'image-rendering' property 142
11.10.5 The 'buffered-rendering' property 142

11.11 Inheritance of painting properties 143
11.12 Object and group opacity: the 'opacity' property 143
11.13 Color 146
11.13.1 Syntax for color values 146
11.13.2 HTML color keywords 147

11.14 Paint servers 147
11.14.1 System paint servers 147
11.14.2 The 'solidColor' element 148
11.14.3 The 'color' property 150

11.15 Gradients 151
11.15.1 Linear gradients 151
11.15.2 Radial gradients 153
11.15.3 Defining gradient stops: the 'stop' element 154

12 Multimedia 157
12.1 Media elements 157
12.1.1 Media timeline and document timeline 157
12.1.2 Media availability 158
12.1.3 Platform limits 159
12.1.4 Audio mixing for 'audio' and 'video' elements 159
12.1.5 Discrete control of audio and video 159
12.1.6 Controlling media playback through script 159

12.2 The 'audio' element 159
12.3 The 'video' element 161
12.3.1 Restricting the transformation of the 'video' element 164
12.3.2 Restricting compositing of the 'video' element 165
12.3.3 Examples 165

12.4 The 'animation' element 168
12.5 The 'audio-level' property 170
12.6 Attributes for runtime synchronization 170
12.7 The 'initialVisibility' attribute 172

13 Interactivity **173**
13.1 Introduction 173
13.2 Complete list of supported events 173
13.3 User interface events 177
13.4 Pointer events 178
13.5 Text events 178
13.6 Key events 178
13.7 Event flow 179
13.8 Event dispatching 179
13.9 Processing order for user interface events 179
13.10 The 'pointer-events' property 180
13.11 Magnification and panning 182
13.12 Element focus 182
13.12.1 The 'focusable' attribute 182

13.13 Navigation 183
13.13.1 Navigation behavior 183
13.13.2 Specifying navigation 184
13.13.3 Specifying focus highlighting 187
13.13.4 Obtaining and listening to focus programmatically 188

14 Linking **190**
14.1 References 190
14.1.1 Overview 190
14.1.2 IRIs and URIs 190
14.1.3 Syntactic forms: IRI and FuncIRI 191
14.1.4 Reference restrictions 191
14.1.5 IRI reference attributes 194
14.1.6 Processing of external references to documents 199

14.2 Links out of SVG content: the 'a' element 200
14.2.1 Indicating links 202

14.3 Linking into SVG content: IRI fragments and SVG views 203
14.3.1 Introduction: IRI fragments and SVG views 203
14.3.2 SVG fragment identifiers 203

15 Scripting **205**
15.1 Specifying the scripting language 205
15.1.1 Specifying the default scripting language 205
15.1.2 Local declaration of a scripting language 205

15.2 The 'script' element 205
15.2.1 Script processing 205

15.3 XML Events 208
15.4 The 'listener' element 210
15.5 The 'handler' element 212
15.5.1 Parameters to 'handler' elements 214

15.6 Event handling 214
15.7 Processing inline executable content 214

16 Animation **216**
16.1 Introduction 216
16.2 Animation elements 216
16.2.1 Overview 216
16.2.2 Relationship to SMIL 2.1 Animation 216

16.2.3 Animation elements example ... 217
16.2.4 Attributes to identify the target element for an animation ... 218
16.2.5 Attributes to identify the target attribute or property for an animation ... 219
16.2.6 Animation with namespaces ... 220
16.2.7 Paced animation and complex types ... 221
16.2.8 Attributes to control the timing of the animation ... 222
16.2.9 Attributes that define animation values over time ... 226
16.2.10 Attributes that control whether animations are additive ... 230
16.2.11 Inheritance ... 231
16.2.12 The 'animate' element ... 231
16.2.13 The 'set' element ... 232
16.2.14 The 'animateMotion' element ... 232
16.2.15 The 'mpath' element ... 236
16.2.16 The 'animateColor' element ... 237
16.2.17 The 'animateTransform' element. ... 237
16.2.18 Attributes and properties that can be animated ... 239

16.3 Animation using the SVG DOM ... 240
16.4 Animation and the bounding box ... 241

17 Fonts ... 242
17.1 Introduction ... 242
17.1.1 Describing fonts available to SVG ... 242
17.1.2 Defining fonts in SVG ... 242

17.2 Overview of SVG fonts ... 243
17.3 The 'font' element ... 244
17.4 The 'glyph' element ... 244
17.5 The 'missing-glyph' element ... 248
17.6 Glyph selection rules ... 248
17.7 The 'hkern' element ... 249
17.8 Describing a font ... 250
17.8.1 Overview of font descriptions ... 250
17.8.2 The 'font-face' element ... 250
17.8.3 The 'font-face-src' element ... 255
17.8.4 The 'font-face-uri' element ... 255

18 Metadata ... 258
18.1 Introduction ... 258
18.2 The 'metadata' element ... 258
18.2.1 A 'metadata' element example ... 259

18.3 Extensible metadata attributes ... 259

19 Extensibility ... 261
19.1 Foreign namespaces and private data ... 261
19.2 Embedding foreign object types ... 261
19.2.1 The 'foreignObject' element ... 262
19.2.2 Examples of 'foreignObject' ... 263

Appendix A The SVG Micro DOM (uDOM) ... 267
A.1 Introduction ... 268
A.2 Overview of the SVG uDOM ... 268
A.2.1 Document access ... 269
A.2.2 Tree navigation ... 269
A.2.3 Element creation ... 269
A.2.4 Element insertion ... 269

A.2.5 Element removal . . . 269
A.2.6 Attribute and property access . . . 270
A.2.7 Event listener registration and removal . . . 270
A.2.8 Animation . . . 271
A.2.9 Multimedia control . . . 271
A.2.10 Java package naming . . . 271

A.3 Conforming to the SVG uDOM . . . 271
A.3.1 Float values . . . 271
A.3.2 Attribute/property normalization . . . 271
A.3.3 Text content access . . . 272

A.4 Module: dom . . . 273
A.4.1 DOMException . . . 273
A.4.2 Node . . . 274
A.4.3 Element . . . 275
A.4.4 Document . . . 275
A.4.5 ElementTraversal . . . 276
A.4.6 Location . . . 276
A.4.7 Window . . . 277

A.5 Module: views . . . 278
A.5.1 AbstractView . . . 278
A.5.2 DocumentView . . . 278

A.6 Module: events . . . 279
A.6.1 EventTarget . . . 279
A.6.2 EventListener . . . 279
A.6.3 Event . . . 280
A.6.4 MouseEvent . . . 280
A.6.5 MouseWheelEvent . . . 281
A.6.6 TextEvent . . . 281
A.6.7 KeyboardEvent . . . 282
A.6.8 UIEvent . . . 286
A.6.9 ProgressEvent . . . 287

A.7 Module: smil . . . 288
A.7.1 ElementTimeControl . . . 288
A.7.2 TimeEvent . . . 289

A.8 Module: svg . . . 290
A.8.1 SVGException . . . 290
A.8.2 SVGDocument . . . 290
A.8.3 SVGUseElement . . . 290
A.8.4 SVGElementInstance . . . 291
A.8.5 SVGSVGElement . . . 292
A.8.6 SVGRGBColor . . . 296
A.8.7 SVGRect . . . 297
A.8.8 SVGPoint . . . 297
A.8.9 SVGPath . . . 298
A.8.10 SVGMatrix . . . 300
A.8.11 SVGLocatable . . . 302
A.8.12 SVGLocatableElement . . . 306
A.8.13 TraitAccess . . . 306
A.8.14 Additional accessing rules . . . 322
A.8.15 SVGElement . . . 323
A.8.16 SVGTimedElement . . . 324

A.8.17 SVGAnimationElement . . . 324
A.8.18 SVGVisualMediaElement . . . 324
A.8.19 SVGTimer . . . 325
A.8.20 SVGGlobal . . . 326
A.8.21 AsyncStatusCallback . . . 329
A.8.22 AsyncURLStatus . . . 329
A.8.23 EventListenerInitializer2 . . . 330

Appendix B IDL Definitions . . . 331
Appendix C Implementation Requirements . . . 338
C.1 Introduction . . . 338
C.2 Unsupported elements, attributes, properties, attribute values and property values . . . 338
C.3 Error processing . . . 338
C.4 Namespace, version, baseProfile, requiredFeatures and requiredExtensions . . . 339
C.5 Clamping of color and opacity Values . . . 339
C.6 'path' element implementation notes . . . 340
C.7 Text selection implementation notes . . . 340
C.8 Printing implementation notes . . . 341

Appendix D Conformance Criteria . . . 342
D.1 Introduction . . . 342
D.2 Terminology . . . 342
D.3 SVG content conformance . . . 342
D.3.1 Conforming SVG Document Fragments . . . 342
D.3.2 Conforming SVG Stand-Alone Documents . . . 343
D.3.3 Conforming SVG Included Document Fragments . . . 343
D.3.4 Conditionally Conforming SVG Tiny 1.2 Document Fragments . . . 344

D.4 SVG writer conformance . . . 344
D.4.1 Conforming SVG Generators . . . 344
D.4.2 Conforming SVG Authoring Tools . . . 344
D.4.3 Conforming SVG Servers . . . 344

D.5 SVG reader conformance . . . 345
D.5.1 Conforming SVG Interpreters . . . 345
D.5.2 Conforming SVG Viewers . . . 345

D.6 Extension conformance requirements . . . 347
D.7 Non-XML encoding conformance requirements . . . 347

Appendix E Conformance to QA Framework Specification Guidelines . . . 348
E.1 Introduction . . . 348
E.2 Checklist table . . . 348
E.3 List of deprecated features . . . 350

Appendix F Accessibility Support . . . 351
F.1 WAI accessibility guidelines . . . 351
F.2 SVG content accessibility guidelines . . . 351
F.3 SVG user agent accessibility guidelines . . . 352

Appendix G Internationalization Support . . . 353
G.1 Introduction . . . 353
G.2 Internationalization and SVG . . . 353
G.3 SVG internationalization guidelines . . . 353
G.4 Markup for the internationalization and localization of SVG . . . 353

Appendix H JPEG Support . . . 355
H.1 Introduction . . . 355

H.2 Required support . . . 355

Appendix I Minimizing SVG File Sizes . . . 356

Appendix J Feature Strings . . . 357

J.1 General feature strings . . . 357

J.2 Specific feature strings . . . 358

Appendix K Element Table . . . 360

Appendix L Attribute and Property Tables . . . 368

L.1 Property table . . . 368

L.2 Attribute table . . . 370

Appendix M Media Type Registration for image/svg+xml . . . 381

M.1 Introduction . . . 381

M.2 Registration of media type image/svg+xml . . . 381

Appendix N RelaxNG Schema for SVG Tiny 1.2 . . . 383

Appendix O ECMAScript Language Binding . . . 384

O.1 Module dom . . . 384

O.2 Module views . . . 386

O.3 Module events . . . 386

O.4 Module smil . . . 388

O.5 Module svg . . . 388

O.6 Sequences . . . 393

Appendix P Java Language Binding . . . 395

P.1 Package org.w3c.dom . . . 395

P.2 Package org.w3c.dom.views . . . 397

P.3 Package org.w3c.dom.events . . . 397

P.4 Package org.w3c.dom.smil . . . 398

P.5 Package org.w3c.dom.svg . . . 399

Appendix Q Perl Language Binding . . . 406

Q.1 Module dom . . . 406

Q.2 Module views . . . 408

Q.3 Module events . . . 408

Q.4 Module smil . . . 409

Q.5 Module svg . . . 410

Appendix R Python Language Binding . . . 416

R.1 Module dom . . . 416

R.2 Module views . . . 418

R.3 Module events . . . 418

R.4 Module smil . . . 419

R.5 Module svg . . . 420

Appendix S References . . . 426

S.1 Normative references . . . 0

S.2 Informative references . . . 0

Appendix T Change History . . . 434

T.1 Changes over the whole document . . . 434

1 Introduction

Contents

1.1 About SVG 1
1.2 SVG Tiny 1.2 1
1.2.1 Profiling the SVG specification 2
1.3 Defining an SVG Tiny 1.2 document 2
1.4 SVG MIME type, file name extension and Macintosh file type 3
1.5 Compatibility with other standards efforts. 3
1.6 Definitions 3
1.7 How to reference this specification. 9
1.8 How to use this specification. 9

1.1 About SVG

SVG is a language for describing two-dimensional graphics in XML [XML10, 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, video, and audio), and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects.

SVG documents 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 *Architecture of the World Wide Web* [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 *Mobile SVG Profiles* specification [SVGM11] (also known as SVG Mobile 1.1) 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 Micro DOM, 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 Profiling the SVG specification

The Tiny profile of SVG 1.2 consists of all of the features defined within this specification. As a 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 [SVGM11]. Backwards compatible means that conformant SVG Tiny 1.1 content will render the same in conformant SVG Tiny 1.2 user agents as it did in conformant SVG Tiny 1.1 user agents. 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 the description of version control in the Implementation Requirements appendix for details.
- 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 ([XML10], section 2.8, and [XML11], section 2.8), 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 can be performed using the 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` and is *mutable* [NSState]; names may be added over time by the W3C SVG Working Group by publication in W3C Technical Reports.

Here is an example of an SVG Tiny 1.2 document:

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 Tiny 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 [RFC1952].

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).

(See Conformance Criteria for more information about gzip-compressed SVG files transmitted over HTTP.)

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 both the *Extensible Markup Language (XML) 1.1* [XML11] and *Extensible Markup Language (XML) 1.0 (Third Edition)* [XML10] Recommendations.
- SVG Tiny 1.2 is compatible with both the *Namespaces in XML 1.0* [XML-NS10] and the *Namespaces in XML 1.1* [XML-NS] Recommendations.
- SVG Tiny 1.2 utilizes *XML Linking Language (XLink)* [XLINK10] 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 as defined in *xml:id Version 1.0* [XMLID].
- SVG Tiny 1.2 content can be generated using *XSL Transformations (XSLT) Version 1.0* [XSLT] or *Version 2.0* [XSLT2]. (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 Core* [DOM3], including namespace support and event handling.
- SVG Tiny 1.2 incorporates some features from the *Synchronized Multimedia Integration Language (SMIL) 2.1 Specification* [SMIL21], including the **'prefetch'** and **'switch'** elements, 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 2.1. In addition, SVG Tiny 1.2 has been designed to allow SMIL 2.1 to use animated or static SVG content as media components.
- SVG is compatible with W3C work on internationalization. References (W3C and otherwise) include: *The Unicode Standard* [UNICODE] and the *Character Model for the World Wide Web 1.0* [CHARMOD]. (See Internationalization Support.)
- SVG is compatible with W3C work on Web Accessibility [WAI]. (See Accessibility Support).

In environments which support the *Document Object Model (DOM) Core* [DOM3] for other XML grammars (e.g., *XHTML 1.0* [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 ([XSL], section 4.2.3).

animation element
Using the various animation elements, you can define motion paths, fade-in or fade-out effects, and allow objects to grow, shrink, spin or change color. The following five elements are animation elements: **'animate'**, **'animateColor'**, **'animateMotion'**, **'animateTransform'** and **'set'**. Animation elements are further described in Animation elements.

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

before-edge
Defined in the XSL Area Model ([XSL], section 4.2.3).

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 description of the canvas in the Coordinate Systems, Transformations and Units chapter.

bounding box
A bounding box is the tightest fitting rectangle aligned with the axes of that element's user coordinate system that entirely encloses it and its descendants. For details, see the description of the bounding box in the Coordinate Systems, Transformations and Units chapter.

conditional processing attribute
A conditional processing attribute is one of the five attributes that may appear on most SVG elements to control whether or not that element will be processed. Those attributes are **'requiredExtensions'**, **'requiredFeatures'**, **'requiredFonts'**, **'requiredFormats'** and **'systemLanguage'**.

container element
An element which can have graphics elements and other container elements as child elements. Specifically, the following elements are container elements: **'a'**, **'defs'**, **'g'**, **'svg'** and **'switch'**.

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

- The sub-tree is a valid SVG document fragment.
- The sub-tree contains the element in question.
- 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] * matrix**. The current transformation matrix defines the mapping from the user coordinate system into the viewport coordinate system. See Coordinate system transformations.

decorated bounding box
The decorated bounding box follows the definition for bounding box, with the exception that it takes into account not only the geometry, but also all geometry-based drawing operations that are marked in their definitions as contributing to this calculation.

descriptive element
An element, not itself in the rendering tree, which provides supplementary information about the container element or graphics element to which it applies (i.e., the described element or elements). Specifically, the following elements are descriptive elements: **'desc'**, **'metadata'**, and **'title'**.

document begin

The document begin for a given SVG document fragment is the time at which the document's timeline is considered to begin. It depends on the value of the **'timelineBegin'** attribute:

- If **'timelineBegin'** is **'onLoad'**, then the document begin is the exact time at which the **'svg'** element's `load` event is triggered.
- Otherwise, if **'timelineBegin'** is **'onStart'**, then the document begin is the exact time at which the **'svg'** element's start-tag ([XML10, XML11], section 3.1) is fully parsed and processed.

document end

The document end of an SVG document fragment is the time at which the document fragment has been released and is no longer being processed by the user agent.

document time

Indicates the position in the timeline relative to the document begin of a given document fragment. Document time is sometimes also referred to as *presentation time*. For additional information see the SMIL 2.1 definition of document time ([SMIL21], section 10.7.1).

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

A graphics element is an SVG element that can cause graphics to be drawn onto the target canvas. Specifically, the following elements are graphics elements: **'animation'**, **'circle'**, **'ellipse'**, **'image'**, **'line'**, **'path'**, **'polygon'**, **'polyline'**, **'rect'**, **'text'**, **'textArea'**, **'use'** and **'video'**.

graphics referencing element

A graphics referencing element is a graphics element that uses a reference to a different document or element as the source of its graphical content. The following elements are graphics referencing elements: **'animation'**, **'foreignObject'**, **'image'**, **'use'** and **'video'**.

host language

A host language is a syntax which incorporates one or more SVG document fragments by inclusion or by reference, and which defines the interactions between document fragments; an example of this is *WICD Core 1.0*, an XML framework which defines how XHTML, SVG, MathML, XForms, SMIL, and other syntaxes interact [WICD].

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.

inactive element

An element is inactive when it is outside the active duration or when it is paused. Aural aspects of elements which are inactive (e.g. audio, and the audio track of a video element) are silent. SMIL defines the behavior of inactive elements with respect to timing, events, and hyperlinking. See Modelling interactive, event-based content in SMIL, Paused Elements and Active Duration and Event Sensitivity ([SMIL21], sections 10.11.2 and 10.4.3).

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

Invalid IRI reference
An invalid IRI reference is an IRI reference that is syntactically invalid, cannot be resolved to a resource or takes a form that is not allowed for a given attribute, as defined in Reference restrictions.

lacuna value
A lacuna value is a defined behavior used when an attribute or property is not specified, or when an attribute or property has an unsupported value. This value is to be used for the purposes of rendering, calculating animation values, and when accessing the attribute or property using the `TraitAccess` interface. As opposed to an XML default value, however, the attribute or property and its value are not visible in the DOM, and cannot be accessed with DOM methods (e.g. getAttribute). For lacunae which are properties, if the property is inherited and there is no inherited value (for example, on the root element), the lacuna value is the initial value as specified in the definition of that property ([CSS2], section 6.1.1). For non-inherited properties, the lacuna value is always the initial value.

Note that a lacuna value is distinct from the XML term default value, which uses DTD lookup to determine whether an attribute is required and what its value is, and inserts required attributes and their values into the DOM ([XML10], section 3.3.2). At the XML parser level, SVG Tiny 1.2 does not have default values; lacunae are part of the SVG application layer, and their values are derived from the UA.

local IRI reference
A local IRI reference is an IRI reference that references a fragment within the same resource. See References.

navigation attribute
A navigation attribute is an XML attribute that specifies the element to be focused when the user instructs the SVG user agent to navigate the focus in a particular direction or to set the focus to the next or previous element in the focus ring. Specifically, the following attributes are navigation attributes: **'nav-next'**, **'nav-prev'**, **'nav-up'**, **'nav-down'**, **'nav-left'**, **'nav-right'**, **'nav-up-left'**, **'nav-up-right'**, **'nav-down-left'** and **'nav-down-right'**. See Specifying navigation.

non-local IRI reference
A non-local IRI reference is an IRI reference that references a different document or an element within a different document.

media element
A media element is an element which defines its own timeline within its own time container. The following elements are media elements: **'animation'**, **'audio'** and **'video'**. See Multimedia.

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 Tiny 1.2 supports two types of built-in paint: color and gradients.

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

property
A property is a parameter that helps specify how a document should be rendered. A complete list of the SVG properties can be found in the Attribute and Property Table appendix. Properties are assigned to elements in the SVG language by presentation attributes. See Styling.

rendering tree

The rendering tree is the set of elements being rendered, aurally or visually using the painters model, in an SVG document fragment. The following elements in the fragment and their children are part of the SVG document fragment, but *not* part of the rendering tree (and thus are not rendered):

- a **'defs'**, **'discard'**, **'font'**, **'handler'**, **'linearGradient'**, **'listener'**, **'metadata'**, **'mpath'**, **'prefetch'**, **'radialGradient'**, **'script'** or **'solidColor'** element
- elements whose **'display'** property is set to **'none'**
- elements with one or more conditional processing attributes that evaluate to false
- direct children of a **'switch'** element, other than the child that evaluates to true
- animation elements

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. Note that elements with zero opacity, or no **'fill'** and no **'stroke'**, or with an **'audio-level'** of zero, or with the **'visibility'** property set to **hidden**, *are* still in the rendering tree.

rootmost 'svg' element

The rootmost **'svg'** element is 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 XBL [XBL2]. See also SVG document fragment.

shadow tree

A tree fragment that is not part of the DOM tree, but which is attached to a referencing element (e.g. **'use'** element) in a non-parent-child relationship, for the purpose of rendering and event propagation. The shadow tree is composed as if it were deep-structure clone of the referenced element in the rendering tree. The shadow tree is kept in synchronization with the contents of the referenced element, so that any animation, DOM manipulation, or non-DOM interactive state occurring on the referenced element are also applied to all the referencing instances. In SVG Tiny 1.2, only a subset of all SVG DOM methods to access the shadow tree are available.

Also referred to as an "instance tree".

shape

A shape is a graphics element that comprises a defined combination of straight lines and curves. Specifically, the following elements are shapes: **'circle'**, **'ellipse'**, **'line'**, **'path'**, **'polygon'**, **'polyline'** and **'rect'**.

stroke

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

SVG context

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

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

In SVG Tiny 1.2, an SVG context contains one SVG document fragment.

SVG document fragment

An SVG document fragment is the XML document sub-tree whose rootmost element is an **'svg'** element (that is, the rootmost 'svg' element.)

An SVG document fragment consists of either a stand-alone SVG document, or a fragment of a parent XML document where the fragment is enclosed by the rootmost 'svg' element.

In SVG Tiny 1.2, the SVG document fragment must not contain nested **'svg'** elements. Nested **'svg'** elements are unsupported elements and must not be rendered. Note that document conformance is orthogonal to SVG document fragment conformance.

For further details, see the section on Conforming SVG Document Fragments.

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

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

syncbase
The syncbase of an animation element timing specifier is the element whose timing this element is relative to, as defined in SMIL 2.1 ([SMIL21], section 10.7.1).

text content element
A text content element is an SVG element that causes a text string to be rendered onto the canvas. The SVG Tiny 1.2 text content elements are the following: **'text'**, **'textArea'** and **'tspan'**.

text content block element
A text content block element is a text content element that serves as a standalone element for a unit of text, and which may optionally contain certain child text content elements (e.g. **'tspan'**). SVG Tiny 1.2 defines two text content block elements: **'text'** and **'textArea'**.

timed element
A timed element is an element that supports the SVG timing attributes. The following elements are timed elements: **'audio'**, **'animate'**, **'animateColor'**, **'animateMotion'**, **'animateTransform'**, **'animation'**, **'set'** and **'video'**.

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

unsupported value
An unsupported value is a value that does not conform to this specification, but is not specifically listed as being 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 [UAAG].

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
User space is 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 viewport is a rectangular region within the current canvas onto which graphics elements are to be rendered. See the description of the initial viewport in the Coordinate Systems, Transformations and Units chapter.

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

Viewport space is 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 local name is the string in quotes (e.g., "svg" or "path"). An exception to this is the **'listener'** element, whose namespace URI is `http://www.w3.org/2001/xml-events`.

1.7 How to reference this specification

When referencing this specification as a whole or when referencing a chapter or major section, use the undated URI, `http://www.w3.org/TR/SVGTiny12/`, where possible. This allows the reference to always refer to the latest version of this specification.

1.8 How to use this specification

This section is informative.

This specification is meant to serve both as a guide to authors in creating SVG content, and as a detailed reference for implementors of browsers, viewers, authoring tools, content processing tools, and other user agents to create conforming interoperable implementations for viewing SVG documents or outputting robust SVG code. It is not intended as a comprehensive manual for authoring content, and it is expected that books, tutorials, and other materials based on this specification will be produced to appeal to different audiences. It is meant to serve as a definitive source for authors and users to reference when reporting bugs and feature requests to implementations.

When reading this specification, in order to gain a complete understanding of the syntax concepts, readers should reference the individual definitions for elements, attributes, and properties, but also consult the definitions list, the element, attribute, property tables, and for more technically adept readers, the RelaxNG schema. For understanding scripting in SVG, readers should consult the sections on Interactivity, Scripting, and the SVG Micro DOM (uDOM).

2 Concepts

Contents

2.1 Explaining the name: SVG 10
2.1.1 Scalable 10
2.1.2 Vector 10
2.1.3 Graphics 10
2.1.4 XML 11
2.1.5 Namespace 11
2.1.6 Scriptable 11

2.2 Important SVG concepts 11
2.2.1 Graphical objects 11
2.2.2 Re-use 11
2.2.3 Fonts 11
2.2.4 Animation 11

2.3 Options for using SVG in Web pages 11

This chapter is informative.

2.1 Explaining the name: SVG

SVG stands for Scalable Vector Graphics, 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 [XML10, XML11], 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 [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 Re-use

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 (X)HTML **'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 (X)HTML **'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 (X)HTML **'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'** ([CSS2], section 14.2.1) and **'list-style-image'** ([CSS2], section 12.6.2) properties that are included in both CSS and XSL.

3 Rendering Model

Contents

3.1 Introduction . . . 13
3.2 The painters model . . . 13
3.3 Rendering order . . . 13
3.4 Types of graphics elements . . . 13
    3.4.1 Rendering shapes and text . . . 13
    3.4.2 Rendering raster images . . . 14
    3.4.3 Rendering video . . . 14
3.5 Object opacity . . . 14
3.6 Parent compositing . . . 14

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 simple alpha compositing.

3.3 Rendering order

SVG defines a rendering tree. Elements in the rendering tree have an implicit drawing order. Elements 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 lines 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., paint can be applied to the interior of the shape) and stroked (i.e., painted applied 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 compositing.

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

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

<boolean>
A boolean value, specified as either **'true'** or **'false'**.

<Char>
A character, as defined by the Char production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.2), or the Char production in *Extensible Markup Language (XML) 1.1* ([XML11], section 2.2) if the document is an XML 1.1 document.

<Clock-value>
An amount of time, used by various attributes on timed elements. The grammar describing possible values for a <Clock-value> is given in the Clock values section of the Animation chapter.

<color>
The basic type <color> defines a color within the sRGB color space [SRGB]. The <color> type is used as the value of the **'color'** property and is a component of the definitions of properties **'fill'**, **'stroke'**, **'stop-color'**, **'solid-color'** and **'viewport-fill'**.
All of the syntax alternatives for <color> defined in Syntax for color values must be supported. All RGB colors are specified in the sRGB color space [SRGB]. Using sRGB provides an unambiguous and objectively measurable definition of the color, which can be related to international standards [COLORIMETRY].

<content-type>
An Internet media type, as per *Multipart Internet Mail Extensions: (MIME) Part Two: Media Types* [RFC2046].

<coordinate>
A <coordinate> is a length in the user coordinate system that is the given distance from the origin of the user coordinate system along the relevant axis (the x-axis for X coordinates, the y-axis for Y coordinates). Its syntax is the same as that for <length>.

<focus>
The type of value that can be used in the various navigation attributes, such as **'nav-next'**, **'nav-prev'**, etc. See Specifying navigation for the definition of the values that can be used in those attributes.

<font-family-value>
A list of font family names and generic names. Specifically, this is the type of value that can be used for the **'font-family'** property, excluding the **'inherit'** value.

<family-name>
A single font family name as given by a <family-name>, as defined in *Extensible Stylesheet Language (XSL) Version 1.1* ([XSL], section 7.9.2).

<font-size-value>
A value that can be used for the **'font-size'** property, excluding the **'inherit'** value.

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

<ID>
The type of value that can be used in an XML attribute of type ID (such as **'id'** and **'xml:id'**); that is, a string matching the Name production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.3), or the Name production in *Extensible Markup Language (XML) 1.1* ([XML11], section 2.3) if the document is an XML 1.1 document.

<IDREF>

The type of value that can be used in an XML attribute of type IDREF (such as **'observer'**); that is, a string matching the Name production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.3), or the Name production in *Extensible Markup Language (XML) 1.1* ([XML11], section 2.3) if the document is an XML 1.1 document.

<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.

<integer> values in conforming SVG Tiny 1.2 content must be within the range of -32,768 to 32,767, inclusive.

<IRI>

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

<language-id>

The type of value accepted by the **'xml:lang'** attribute as defined in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.12) and *Extensible Markup Language (XML) 1.1* ([XML11], section 2.12). Specifically, this is either a language tag as defined by *BCP 47* ([BCP 47], section 2) or the empty string.

<length>

A length is a distance measurement. The format of a <length> is a <number> optionally followed 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 optional units on the **'width'** and **'height'** attributes on the **'svg'** element. These can specify values in any of the following 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 percentage of the viewport size (refer to the section that discusses Units in general).

<list-of-content-types>

A space-separated list of Internet media types, as used by the **'requiredFormats'** attribute.

The following is an EBNF grammar describing the <list-of-content-types> syntax [EBNF]:

```
list-of-content-types ::= content-type
                          | content-type wsp list-of-content-types
wsp                   ::= (#x9 | #xD | #xA | #x20)*
```

<list-of-family-names>

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

<list-of-language-ids>

A <list-of-language-ids> is a comma separated list of non-empty <language-id> values. White space may be used before or after the commas.

<list-of-strings>

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

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

```
list-of-strings ::= string
                    | string wsp list-of-strings
wsp             ::= (#x9 | #xD | #xA | #x20)*
```

<list-of-Ts>

(Where *T* is a type other than <content-type>, <string>, <language-id> and <family-name>.) A list consists of a separated sequence of values. Unless explicitly described differently, each pair of list items can be separated either by a comma (with optional whitespace before and after the comma) or by white space alone.

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).

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

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

Substituting a type other than <content-type>, <string>, <language-id> and <family-name> for *T* will yield a grammar for a list of that type.

<long>

A <long> 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.

<long> values in conforming SVG Tiny 1.2 content must be within the range of -2,147,483,648 to 2,147,483,647, inclusive.

<NCName>

An XML name without colons, as defined by the NCName production in *Namespaces in XML 1.0* ([XML-NS10], section 3), or the NCName production in *Namespaces in XML 1.1* ([XML-NS], section 3) if the document is an XML 1.1 document.

<number>

A <number> value is specified in either decimal or scientific notation. A <number> using decimal notation 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. Using scientific notation, it consists of a number in decimal notation followed by the letter "E" or "e", followed by an <integer>.

The following is an EBNF grammar describing the <number> syntax [EBNF]:

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

<number> values in conforming SVG Tiny 1.2 content must have no more than 4 decimal digits in the fractional part of their decimal expansion and must be in the range -32,767.9999 to +32,767.9999, inclusive. 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.

<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.

<path-data>

The <path-data> type is used to represent path data, as can be specified in the **'d'** attribute on a **'path'** element. See the detailed description of path data, including the grammar for path data.

<points-data>

The <points-data> type is used to represent a list of points, as can be specified in the **'points'** attribute on a **'polyline'** or **'polygon'** element. See the grammar for points data.

<QName>

The <QName> type is a qualified name, as defined by the QName production in *Namespaces in XML 1.0* ([XML-NS10], section 3), or the QName production in *Namespaces in XML 1.1* ([XML-NS], section 3) if the document is an XML 1.1 document. If the <QName> has a prefix, then the prefix is expanded into a tuple of an IRI reference and a local name, using the namespace declarations in effect where the name occurs. Note that, as with unprefixed attributes, the default namespace is *not* used for unprefixed names.

<string>
A sequence of zero or more <Char>s.

<transform>
A <transform> is a transformation specification, as can be specified in the **'transform'** attribute. As described in The 'transform' attribute, three types of values are allowed: a transform list, a transform reference and the **'none'** value.
The following is an EBNF grammar describing the <transform> syntax [EBNF]:

```
transform ::= transform-list | transform-ref | "none"
```

<XML-Name>
An XML name, as defined by the Name production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.3), or the Name production in *Extensible Markup Language (XML) 1.1* ([XML11], section 2.3) if the document is an XML 1.1 document.

<XML-NMTOKEN>
An XML name token, as defined by the Nmtoken production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.3), or the Nmtoken production in *Extensible Markup Language (XML) 1.1* ([XML11], section 2.3) if the document is an XML 1.1 document.

<XML-NMTOKENS>
An space separated sequence of XML name tokens, as defined by the Nmtokens production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.3), or the Nmtokens production in *Extensible Markup Language (XML) 1.1* ([XML11], section 2.3) if the document is an XML 1.1 document.

5 Document Structure

Contents

5.1 Defining an SVG document fragment: the 'svg' element . . . 19
- 5.1.1 Overview . . . 19
- 5.1.2 The 'svg' element . . . 21

5.2 Grouping: the 'g' element . . . 25
- 5.2.1 Overview . . . 25
- 5.2.2 The 'g' element . . . 25

5.3 The 'defs' element . . . 26
5.4 The 'discard' element . . . 27
5.5 The 'title' and 'desc' elements . . . 28
- 5.5.1 Applicable 'title' and 'desc' . . . 29
- 5.5.2 Multiple 'title' and 'desc' elements . . . 30
- 5.5.3 User interface behavior for 'title' and 'desc' . . . 30

5.6 The 'use' element . . . 31
5.7 The 'image' element . . . 36
5.8 Conditional processing . . . 39
- 5.8.1 Conditional processing overview . . . 39
- 5.8.2 The 'switch' element . . . 40
- 5.8.3 The 'requiredFeatures' attribute . . . 41
- 5.8.4 The 'requiredExtensions' attribute . . . 42
- 5.8.5 The 'systemLanguage' attribute . . . 42
- 5.8.6 The 'requiredFormats' attribute . . . 43
- 5.8.7 The 'requiredFonts' attribute . . . 43

5.9 External resources . . . 44
- 5.9.1 The 'externalResourcesRequired' attribute . . . 44
- 5.9.2 Progressive rendering . . . 44
- 5.9.3 The 'prefetch' element . . . 48

5.10 Common attributes . . . 51
- 5.10.1 Attributes common to all elements . . . 51
- 5.10.2 Attributes for character-content elements . . . 54

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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="5cm" height="4cm" 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 either the *Namespaces in XML 1.0* or *Namespaces in XML 1.1* Recommendations [XML-NS10, 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 example 05_01, above). For more information, refer to the *Namespaces in XML 1.0* or *Namespaces in XML 1.1* Recommendations [XML-NS10, 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'>
      <text/>
    </attribute>
  </optional>
  <optional>
    <attribute name='zoomAndPan' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>disable</value>
        <value>magnify</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='version' 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>
        <value type='string'>basic</value>
        <value type='string'>full</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name='contentScriptType' 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' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value type='string'>onLoad</value>
        <value type='string'>onStart</value>
      </choice>
    </attribute>
  </optional>
  <optional>
```

```
        <attribute name='playbackOrder' 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 value **'1.2'**. See rules for version processing for further instructions, notably on handling of unsupported values.

Modifying the **'version'** attribute using the DOM does not cause any change in behavior. In this case, the original value of the attribute is the one used for document processing.

Animatable: no.

`baseProfile` = **"none" | "full" | "basic" | "tiny"**

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* [SVG11]. The value **'basic'** was defined in the *Mobile SVG Profiles: SVG Tiny and SVG Basic* [SVGM11]. 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.

The lacuna value is **'none'**.

Modifying the **'baseProfile'** attribute using the DOM does not cause any change in behavior. In this case, the original value of the attribute is the one used for document processing.

Animatable: no.

`width` = **"<length>"**

The intrinsic width of the SVG document fragment. Together with the **'height'**, **'viewBox'** and **'preserveAspectRatio'** attributes it defines the intrinsic aspect ratio and (unless both width and height are percentages) the intrinsic size of the svg element. See The initial viewport.

A negative value is unsupported. A value of zero disables rendering of the element.

The lacuna value is **'100%'**.

Animatable: yes.

`height` = **"<length>"**

The intrinsic height of the SVG document fragment.

A negative value is unsupported. A value of zero disables rendering of the element.

The lacuna value is **'100%'**.

Animatable: yes.

`viewBox` = **"<list-of-numbers>" | "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 example 05_22 for an example of using the **'snapshotTime'** attribute.

The lacuna value is **'none'**.

Animatable: no.

`playbackOrder` = **"forwardOnly" | "all"**

Indicates whether it is possible to seek backwards in the document. 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 lacuna 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 *start-tag* (as defined in XML 1.0 ([XML10], section 3.1), or XML 1.1 ([XML11], section 3.1), if the document is an XML 1.1 document) is fully parsed and processed.

The lacuna 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 <content-type> value specifies a media type, per *Multipart Internet Mail Extensions: (MIME) Part Two: Media Types* [RFC2046]. The lacuna value is **"application/ecmascript"**.

Animatable: no.

`zoomAndPan` = **"magnify" | "disable"**

See attribute definition for description.

Animatable: no.

`focusable` = **"true" | "false" | "auto"**

See attribute definition for description.

Animatable: yes.

`Navigation Attributes`

See definition.

Note that **'animateMotion'** and **'animateTransform'** are legal as children to **'svg'** but don't apply to their **'svg'** parent (since the **'svg'** element doesn't have a **'transform'** attribute). They only have any effect if the **'xlink:href'** attribute is specified so that they target a different element for animation.

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

Normally, SVG content is designed to be scalable. In order for the SVG content to scale automatically to fill the available viewport, it 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.

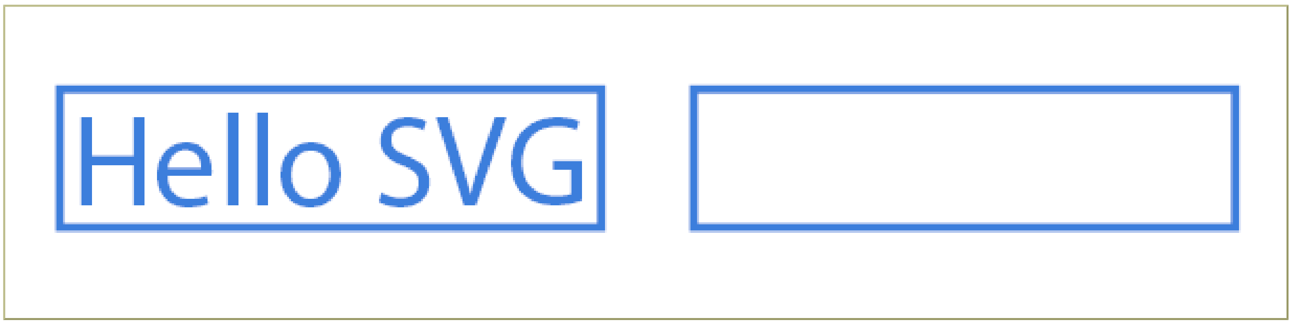
Example: viewBox.svg

```
<?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 which generates thumbnails based on **'snapshotTime'** at the left, UA which doesn't generate thumbnails based on **'snapshotTime'** at the right, e.g. a static viewer).

Example: 05_22.svg

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     viewBox="0 0 400 300" snapshotTime="3">

  <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="white" font-size="60">
    Hello SVG
    <animateColor attributeName="fill" begin="0" dur="3" from="white" to="rgb(60,126,220)"/>
  </text>
</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 'id' or '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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="5cm" height="5cm" 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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="5cm" height="5cm">

  <desc>Groups can nest</desc>

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

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

5.2.2 The 'g' element

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'/>
      <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'**. For performance reasons, authors should put the **'defs'** element before other document content, so that all resources are available to be referenced.

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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="100%" height="100%" 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, thereby reducing the resources required by an SVG user agent. This is particularly useful to help SVG viewers conserve memory while displaying 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.XLink.attr'/>
      <ref name='svg.AnimateBegin.attr'/>
      <ref name='svg.Conditional.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 that identifies the target element to discard. See the definition of **'xlink:href'** on animation elements for details on identifying a target element.

Note that if the target element is not part of the current SVG document fragment then whether the target element will be removed or not is defined by the host language.

If the **'xlink:href'** attribute is not provided, then the target element will be the immediate parent element of the discard element.

Animatable: no.

`begin` = **"*begin-value-list*"**

Indicates when the target element will be discarded. See the definition of **'begin'** on animation elements for details.

The lacuna value is **'0s'**. This indicates that the target element should be discarded immediately once the document begins.

Animatable: no.

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 ([SMIL21], section 10.4.3). 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.

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 **'xlink:href'** attribute has an invalid IRI reference (the target element did not exist, for example), the **'discard'** element itself must still be removed following activation.

Seeking backwards in the timeline ([SMIL21], section 10.4.3) 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. SVG user agents must allow the **'discard'** element to be the target of another **'discard'** element.

The following example demonstrates a simple usage of the **'discard'** element. The list below describes relevant behavior 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" fill="red" stroke="black"
        transform="translate(30 301)">
    <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 'title' and 'desc' elements

Each container element or graphics element in an SVG document may contain one or more of each of the **'title'** and **'desc'** descriptive elements, which together comprise a sort of heading and summary of the containing element. The **'title'** element must contain a brief plain text passage representing the title for the container or graphics element containing it. This short title must provide information supplementary to the rendering of the element, but will normally not be sufficient to replace it. The **'desc'** element must contain a longer, more detailed plain text description for the container or graphics element containing it. This description, along with the content of the **'title'** element, must be usable as replacement content for cases when the user cannot see the rendering of the SVG element for whatever reason.

Authors should always provide at least a **'title'**, and preferably a **'desc'**, as an immediate child element to the **'svg'** element within an SVG document, and to every significant individual graphical composition within the 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). If for any reason, a graphical presentation of the document is not available, the rootmost descriptive elements may represent the complete document and its purpose in a textual manner, and authors should supply meaningful content accordingly.

When descriptive elements are present, alternate presentations of the document are possible, both visual and aural, which display the **'title'** and **'desc'** elements but do not display graphics elements.

For both the **'title'** and the **'desc'** element, the content must be plain text. To provide structured data in other markup languages, authors should use the **'metadata'** or **'foreignObject'** elements instead, as appropriate. When markup is included as a child of the **'title'** or the **'desc'**, a user agent should present only the text content of the descriptive elements.

Note that the **'title'** element is distinct in purpose from the **'xlink:title'** attribute of the **'a'** element. The **'xlink:title'** attribute content is intended not to describe the current resource, but the nature of the linked resource.

Schema: title

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

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'/>
      <ref name='svg.Conditional.attr'/>
      <ref name='svg.Media.attr'/>
    </define>

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

5.5.1 Applicable 'title' and 'desc'

Normally, the descriptive elements that describe a container element or graphics element are direct children of that element. However, SVG documents can have a rich structure, with nested elements each potentially containing **'title'** or **'desc'** child elements, as well as **'use'** elements with **'title'** or **'desc'** in both the **'use'** element itself and in the referenced content. Because of this complex structure, and because the descriptive elements may or may not be present at any given level, the applicable descriptive elements for any given content is determined by the structure, as described here.

For each container element or graphics element, the applicable descriptive elements shall be those which are most shallowly nested in that document fragment, without taking into account descriptive elements in any **'use'** element shadow trees. If the document fragment has no descriptive elements, and it is a **'use'** element, the applicable descriptive elements shall be those contained in the shadow tree. If no descriptive elements are found in the document fragment or any shadow tree, the applicable descriptive elements shall be the nearest ancestor descriptive elements. This algorithm allows authors to reuse descriptive elements defined in referenced resources when desired, or to override them as needed, as well as to describe whole groups of elements. Note that the **'title'** and **'desc'** are not necessarily always paired (i.e., in the same document hierarchy level), and if the user agent should stop searching for an applicable descriptive element if only one or the other is encountered at a particular hierarchy level.

Note that the applicable descriptive elements for elements in a container element does not necessarily entail a description of the individual graphical elements in question, but rather their membership in a more inclusive group (e.g., if the image is of a basket of fruit, with a title of "Fruit Basket" for the containing group and no other descriptive elements, while no one piece of fruit is a fruit basket, the title would still be applicable through inclusion). In essence, there is a difference between container elements and graphics elements when it comes to determining the applicability of a descriptive element; a **'title'** or **'desc'** for a graphics element should be assumed to apply only to that element, while a **'title'** or **'desc'** for a **'g'** may apply to each of the children of that group. Authors should take care to designate all important elements with their own descriptive elements to avoid misconstrued identities and entailments.

5.5.2 Multiple **'title'** and **'desc'** elements

It is strongly recommended that authors use at most one **'title'** and at most one **'desc'** element as an immediate child of any particular element, and that these elements appear before any other child elements (except possibly **'metadata'** elements) or character data content.

Authors may wish to deliberately provide multiple descriptive elements, such as to provide alternate content for different languages. In this case, the author should use conditional processing attributes to allow the user agent to select the best choice according to the user's preferences. For example, the **'systemLanguage'** attribute, with or without the **'switch'** element, will determine the applicable descriptive elements.

If an SVG user agent needs to choose among multiple **'title'** or **'desc'** elements for processing (e.g., to decide which string to use for a tooltip), and if any available conditional processing attributes are insufficient to resolve the best option, the user agent must choose the first of each of the available descriptive elements as the applicable **'title'** and **'desc'**.

5.5.3 User interface behavior for **'title'** and **'desc'**

When the current SVG document fragment is rendered as SVG on visual media, **'title'** and **'desc'** elements are not rendered as part of the canvas. Often, the intent of authors is for descriptive elements to remain hidden (e.g., for aesthetic reasons in pieces of art). However, other authors may wish for this content to be displayed, and providing tangible benefit to these authors encourages best practice in providing descriptive elements. In this case, authors are encouraged to use the **'role'** attribute, with the value **tooltip** ([ARIA], section 4.4.1) to indicate their intent. Future SVG specifications may define an explicit mechanism for indicating whether a tooltip should be displayed.

In order to honor authorial intent, it is strongly recommended that when, and only when, the appropriate **'role'** attribute value is present, user agents display the text content of the applicable **'title'** and **'desc'** elements in a highly visible manner supported by the user agent, such as in a tooltip or status bar, when the pointing device is hovered over the described element or elements, or when the described element is given focus (e.g., through keyboard or pointer navigation). If a tooltip is provided, the user agent is recommended to display the applicable title and descriptions on separate lines, title first, with font styling that distinguishes the two. For long descriptions, the tooltip may wrap the text, and truncate longer passages to a reasonable length. A user agent may preserve spaces and line breaks in the text content in order to structure the presentation of the text.

When an element with descriptive elements is itself the child of an **'a'** element with an **'xlink:title'** attribute, the user agent should display as much of the available information as possible. The user agent is suggested to display the **'xlink:title'** attribute value on a separate line, with a label to identify it, such as "link: ". Commonly, many user agents display the URI of the link (i.e., the value of the **'xlink:href'** attribute) in the status bar or other display area. This information is important, and should not be overridden by any descriptive element content, but may be supplemented by such content.

The rootmost **'title'** element should be used as the document title, and for stand-alone SVG documents, the title should not be displayed as a tooltip, but rather in the browser chrome (as appropriate for the user agent). For embedded SVG documents, such as an SVG image referenced in an HTML document, displaying the rootmost title and description as a tooltip is more appropriate, and the user agent should do so.

If a user agent is an accessibility tool, all available descriptions of the currently focused or hovered element should be exposed to the user in a categorical manner, such that the user may selectively access the various descriptions. The **'desc'** element, in particular, may be given different semantic distinctions by use of values in the **'role'** attribute, such as the ARIA ontology value **description** ([ARIA], section 4.4.1) for textual equivalents of the graphics (the default role).

The following is an example in which an SVG user agent might present the **'title'** and **'desc'** elements as a tooltip.

Example: title-desc-tooltip.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2"
     width="100%" height="100%" viewBox="0 0 160 170">

  <title>Titles and Descriptions</title>
  <desc>An example of how the contents of the 'title' and 'desc' elements may be presented in a user agent.</desc>

  <defs>
    <polygon id="beeCell" points="-14,-24.2487 14,-24.2487 28,0 14,24.2487 -14,24.2487 -28,0" stroke="yellow"
stroke-width="3" stroke-linejoin="round">
      <title role="tooltip">Beehive cell</title>
      <desc role="tooltip">A simple hexagon with a yellow outline and no fill.</desc>
    </polygon>
  </defs>

  <g fill="white">
    <use xlink:href="#beeCell" x="30" y="60" />
    <use xlink:href="#beeCell" x="75" y="34.0192" />
    <use xlink:href="#beeCell" x="120" y="60" />
    <use xlink:href="#beeCell" x="120" y="111.9615" />
    <use xlink:href="#beeCell" x="30" y="111.9615" />
    <use xlink:href="#beeCell" x="75" y="137.9423" />

    <a xlink:href="http://www.example.com/bees.html" xlink:title="Beekeeper's Hive: an apiary resource">
      <title role="tooltip">Link to Beekeeper's Hive: an apiary resource</title>
      <use xlink:href="#beeCell" x="75" y="85.9808" fill="#9900CC">
        <title role="tooltip">Queen's Cell</title>
        <desc role="tooltip">
          A hexagonal beehive cell.

          A purple hexagon in the middle of 6 other empty hexagons, symbolizing that it's filled with royal jelly.
        </desc>
      </use>
    </a>

  </g>

</svg>
```

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, DOM manipulation, or non-DOM interactive state occurring on the referenced element are also applied to the **'use'** element's deeply-cloned tree.

Relative IRIs on a node in a shadow tree are resolved relative to any **'xml:base'** on the node itself, then recursively on any **'xml:base'** on its `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.

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"**.

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.

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.

Except for resolution of relative IRI references as noted and until the referenced elements are modified, 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'**, **'xml:base'** 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*** are 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.

Note also that any changes to the used element are *immediately* reflected in the generated content.

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.
The lacuna value is **'0'**.
Animatable: yes.

`y` = **"<coordinate>"**
The y-axis coordinate of one corner of the rectangular region into which the referenced element is placed.
The lacuna value is **'0'**.
Animatable: yes.

`xlink:href` = **"<IRI>"**
An IRI reference to an element/fragment within an SVG document. An invalid IRI reference is an unsupported value. An empty attribute value (**xlink:href=""**) disables rendering of the element. The lacuna value is the empty string.
Animatable: yes.

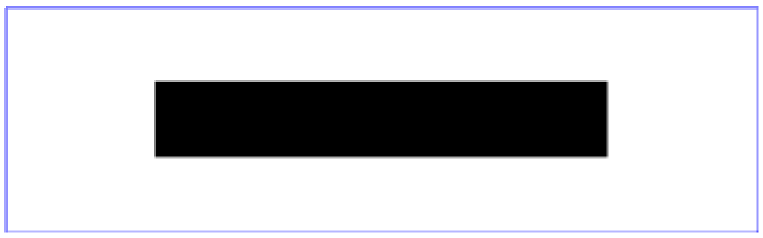
`focusable` = **"true" | "false" | "auto"**
See attribute definition for description.
Animatable: yes.

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

  <desc>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:

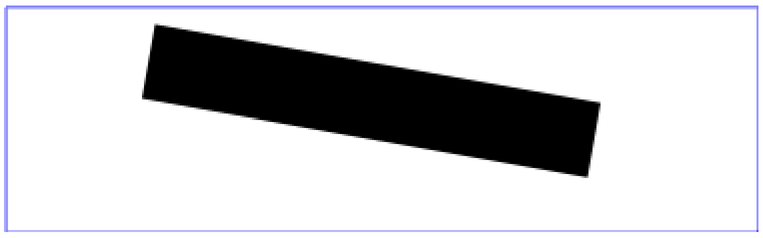
Example: 05_14.svg

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

  <desc>
    The equivalent rendering tree of example 05_13 once the
    'use' element's shadow tree has been created.
  </desc>

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

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

  <!-- begin shadow tree content that the <use> element in the original
       file would generate -->
  <g transform="translate(20,10)">
    <rect width="60" height="10"/>
  </g>
  <!-- end of shadow tree content -->
</svg>
```

Example 05_17 illustrates what happens when a **'use'** has a **'transform'** attribute.

Example: 05_17.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"
     width="10cm" height="3cm" viewBox="0 0 100 30">

  <desc>'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:

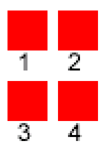
Example: 05_18.svg

```
<?xml version="1.0"?>
<svg 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 0 100 30">

  <desc>'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"/>

  <!-- begin shadow tree content that the <use> element in the original
       file would generate -->
  <g transform="translate(20,2.5) rotate(10)">
    <rect x="0" y="0" width="60" height="10"/>
  </g>
  <!-- end of shadow tree content-->
</svg>
```

Example use-bubble-example-1.svg illustrates four cases of event bubbling with use elements. In case 1, all instances of the **'rect'** element are filled blue on mouse over. For cases 2 and 3, in addition to the **'rect'** elements being filled blue, a black stroke will also appear around the referencing rectangle on mouse over. In case 4, all the rectangles turn blue on mouse over, and a black stroke appears on mouse click.

Example: use-bubble-example-1.svg

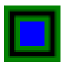
```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny">

  <defs>
    <rect id="rect" width="20" height="20" fill="red">
      <set attributeName="fill" begin="mouseover" end="mouseout" to="blue"/>
    </rect>
  </defs>

  <use fill="red" x="5" y="5" xlink:href="#rect"/>
  <text x="10" y="35">1</text>

  <use id="use2" fill="red" x="30" y="5" xlink:href="#rect"/>
  <rect pointer-events="none" x="30" y="5" width="20" height="20"
        fill="none" stroke-width="3" stroke="none">
    <set attributeName="stroke" begin="use2.mouseover" end="use2.mouseout" to="black"/>
  </rect>
  <text x="35" y="35">2</text>

  <g id="g1">
    <use fill="red" x="5" y="40" xlink:href="#rect"/>
    <rect pointer-events="none" x="5" y="40" width="20" height="20"
          fill="none" stroke-width="3" stroke="none">
      <set attributeName="stroke" begin="g1.mouseover" end="g1.mouseout" to="black"/>
    </rect>
  </g>
  <text x="10" y="70">3</text>

  <use id="use3" fill="red" x="30" y="40" xlink:href="#rect"/>
  <rect pointer-events="none" x="30" y="40" width="20" height="20"
        fill="none" stroke-width="3" stroke="none">
    <set attributeName="stroke" begin="use3.click" dur="500ms" to="black"/>
  </rect>
  <text x="35" y="70">4</text>
</svg>
```



Example use-bubble-example-2.svg illustrates event bubbling with nested **'use'** elements. On mouse over, the **'rect'** element is filled blue and displays a green and black ring.

Example: use-bubble-example-2.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny">

  <defs>
    <rect id="rect" width="20" height="20" fill="red">
      <set attributeName="fill" begin="mouseover" end="mouseout" to="blue"/>
    </rect>
    <g id="use">
      <use fill="red" xlink:href="#rect"/>
      <rect pointer-events="none" width="20" height="20"
            fill="none" stroke-width="8" stroke="none">
        <set attributeName="stroke" begin="use.mouseover" end="use.mouseout" to="green"/>
      </rect>
    </g>
  </defs>

  <use x="5" y="5" id="use2" fill="red" xlink:href="#use"/>
  <rect pointer-events="none" x="5" y="5" width="20" height="20" fill="none" stroke-width="3" stroke="none">
    <set attributeName="stroke" begin="use2.mouseover" end="use2.mouseout" to="black"/>
  </rect>
</svg>
```



Example image-use-base.svg illustrates the handling of relative IRI references. All three use elements result in the same image being displayed, `http://a.example.org/aaa/bbb/ddd/foo.jpg`.

Example: image-use-base.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" viewBox="00 100 100">

  <g xml:base="http://a.example.org/aaa/">
    <g xml:base="/bbb/ccc/">
      <g xml:base="../ddd/" xml:id="bar">
        <image xml:id="foo" xlink:href="foo.jpg" width="100" height="100"/>
      </g>
    </g>
  </g>
  <g xml:base="http://z.example.net/zzz/">
    <g xml:base="/yyy/xxx/">
      <g xml:base="../xxx/">
        <use xlink:href="#foo" />
        <use xlink:href="#bar" />
        <use xlink:href="#bar" xml:base="../ggg/" />
      </g>
    </g>
  </g>
</svg>
```

5.7 The **'image'** element

The **'image'** element indicates that the contents of an image 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 or JPEG [PNG, JPEG]. 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 in the *Portable Network Graphics (PNG) Specification (Second Edition)* [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 (such as PNG and 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 positioned 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 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. an image which 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.ContentTypeAnim.attr'/>
</define>

<define name='image.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.
  The lacuna value is **'0'**.
  Animatable: yes.

y = **"<coordinate>"**
: The y-axis coordinate of one corner of the rectangular region.
  The lacuna value is **'0'**.
  Animatable: yes.

width = **"<length>"**
: The width of the rectangular region.
  A negative value is unsupported. A value of zero disables rendering of the element. The lacuna value is **'0'**.
  Animatable: yes.

height = **"<length>"**
: The height of the rectangular region.
  A negative value is unsupported. A value of zero disables rendering of the element. The lacuna value is **'0'**.
  Animatable: yes.

preserveAspectRatio = **"[defer] <align> [<meet>]"**
: See attribute definition for description.
  Animatable: yes.

xlink:href = **"<IRI>"**
: An IRI reference to the image. An invalid IRI reference is an unsupported value. An empty attribute value (**xlink:href=""**) disables rendering of the element. The lacuna value is the empty string.
  Animatable: yes.

type = **"<content-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 Metadata hints in specifications in the *Authoritative Metadata* TAG finding ([MIME-RESPECT], section 5). To ensure that a user agent only downloads media in formats that it supports, thus optimizing download time and bandwidth usage, authors are encouraged to use **'requiredFormats'**, instead of **'type'**.
  Animatable: yes.

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 xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny">
```

```
  <desc>This document has a reference 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 provides a **'switch'** element and five conditional processing attributes — **'requiredExtensions'**, **'requiredFeatures'**, **'requiredFonts'**, **'requiredFormats'** and **'systemLanguage'** — which provide the ability to specify alternate content 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='ListOfIRI.datatype'/>
    </attribute>
  </optional>
  <optional>
    <attribute name='requiredExtensions' svg:animatable='false' svg:inheritable='false'>
      <ref name='ListOfIRI.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='LanguageIDs.datatype'/>
    </attribute>
  </optional>
</define>
```

Conditional processing attributes do not affect the processing of all elements. They can be specified only on graphics elements, container elements, text content elements, descriptive elements, timed elements and the **'foreignObject'** and **'discard'** elements. A conditional processing attribute on any other element does not affect whether that element will be processed. When a conditional processing attribute is specified on a container element, it affects only the elements on which conditional processing attributes can be specified. For example, a **'requiredExtensions'** attribute on a **'script'** element will not control whether the script is executed. Note that if a conditional processing attribute is specified on a container element which contains scripts, it has no effect on whether the script is executed. In particular, all scripts contained in a **'switch'** element are processed.

The conditional processing attributes act as boolean tests and evaluate to either true or false. If one is not specified, then it is assumed to evaluate to true. The attributes can be used in two ways, depending on the context of the element on which the attributes are specified. If the element's parent node is a **'switch'** element, then at most one of the **'switch'** element's children that conditional processing attributes apply to will be processed. (See the description of the **'switch'** element for details.) Otherwise, if the element's parent node is not a **'switch'** element, and conditional processing attributes do apply to the element, then the attributes determine whether that element will be processed.

What it means for an element not to be processed because of conditional processing attributes specified on it, or because it is a child of a **'switch'** that has selected a different child for processing, depends on the type of element:

- If the element is a graphics element, container element, text content element or a **'foreignObject'** element, then the element is not rendered and is not a part of the rendering tree.

- If the element is a timed element, then the element will never begin, regardless of its timing attributes and any invocations of methods on the `SVGTimedElement` and `ElementTimeControl` interfaces. If the element serves as a syncbase for any other timed elements in the document, then those syncbase references will never resolve to a concrete time. Thus, for example, if the element is an animation element, the animation will never have an effect, and if the element is an **'audio'** element, then it will never generate any sound.
- If the element is a 'discard' element, then it will never trigger the removal of its target element, nor will it remove itself.

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

Example systemLanguage below displays one of three text strings (in Welsh, Greek, or Spanish) if one of those is the user's preferred language. Otherwise, in this example, it displays nothing.

Example: systemLanguage.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" viewBox="0 0 170 200">

  <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 μιλήσουν 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>
```

5.8.2 The 'switch' element

The **'switch'** element is a container element that can be used to select one of its child elements to process based on their conditional processing attributes. The first direct child element of a **'switch'** whose conditional processing attributes all evaluate to true will be processed as normal. All other direct child elements of the **'switch'** that support conditional processing attributes will not be processed. The elements that support conditional processing attributes are listed in the Conditional processing overview section, above.

While conditional processing attributes are supported only on certain elements, those attributes on *all* direct child elements of a **'switch'** are used to determine which children to disable processing for.

The values of the **'display'** and **'visibility'** properties 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.

Note that regardless of whether they are processed or disabled, child elements of the **'switch'** element are still part of the DOM, and rules applying to the uniqueness of the **'id'** and **'xml:id'** attributes still apply. Additionally, elements which would not otherwise be rendered due to conditional processing can still be referenced, for example as the target of a **'use'** element or as a paint server reference in a **'fill'** property.

The element definition schema and content model for **'switch'** are not defined here. It is defined in all the places it can occur.

Schema: switch.at

```
    <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:

`requiredExtensions` = **"<list-of-strings>"**
See attribute definition for description.
Animatable: no.

`requiredFeatures` = **"<list-of-strings>"**
See attribute definition for description.
Animatable: no.

`requiredFonts` = **"<list-of-strings>"**
See attribute definition for description.
Animatable: no.

`requiredFormats` = **"<list-of-content-types>"**
See attribute definition for description.
Animatable: no.

`systemLanguage` = **"<list-of-language-ids>"**
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-strings>"**
A conditional processing attribute that controls conditional processing based on whether the specified features are supported by the SVG user agent. The value is a list of feature strings, with the individual values separated by white space. Only feature strings defined in an existing version of the SVG specification at the time the document is authored (such as those listed in this document's 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.

This attribute evaluates to true for the purpose of conditional processing if and only if all of the specified features are supported. As with all conditional processing attributes, if **'requiredFeatures'** is not specified, then it implicitly evaluates to true. However, if the attribute is specified, but has an empty string value, it evaluates to false. See Conditional processing overview for details on how conditional processing attributes influence document processing.
Animatable: no.

5.8.4 The 'requiredExtensions' attribute

The **'requiredExtensions'** attribute specifies 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 an IRI reference.

Language extensions may be vendor-specific or experimental features for the SVG language itself, or may be separate languages (e.g., XHTML, MathML). If an extension is a separate language that supports the Namespaces in XML 1.0 specification [XML-NS10] or the Namespaces in XML 1.1 specification [XML-NS], the IRI reference should be the Namespace URI for that language (e.g., "http://www.w3.org/1999/xhtml", "http://www.w3.org/1998/Math/MathML"). If the language does not support either Namespaces in XML specification, the IRI reference should be an otherwise unique identifier for that language.

Definition of **'requiredExtensions'**:

`requiredExtensions` = **"<list-of-strings>"**

A conditional processing attribute that controls conditional processing based on whether the specified extensions are supported by the SVG user agent. The value is a list of IRI references which identify the required extensions, with the individual values separated by white space.

This attribute evaluates to true for the purpose of conditional processing if and only if all of the specified extensions are supported. As with all conditional processing attributes, if **'requiredExtensions'** is not specified, then it implicitly evaluates to true. However, if the attribute is specified, but has an empty string value, it evaluates to false. See Conditional processing overview for details on how conditional processing attribute influence document processing.

Animatable: no.

Since white space is used to separate values in the attribute, any white space characters in the IRI reference must be escaped.

5.8.5 The 'systemLanguage' attribute

Definition of **'systemLanguage'**:

`systemLanguage` = **"<list-of-language-ids>"**

A conditional processing attribute that controls conditional processing based on the system language. The value is a comma-separated list of language tags as defined in *BCP 47* ([BCP 47], section 2).

This attribute evaluates to true for the purpose of conditional processing if one of the languages indicated by user preferences equals one of the languages given in the value of this attribute, 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 attribute such that the first tag character following the prefix is U+002D HYPHEN-MINUS ("-"). As with all conditional processing attributes, if **'systemLanguage'** is not specified, then it implicitly evaluates to true. However, if the attribute is specified, but has an empty string value, it evaluates to false. See Conditional processing overview for details on how conditional processing attribute influence document processing.

Animatable: no.

Note that the use of a prefix matching rule to determine whether the attribute evaluates to true or false 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 element on which the **'systemLanguage'** conditional processing 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'** conditional processing 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.

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 for content to provide alternatives so that SVG user agents can choose the format they support. The **'requiredFormats'** attribute can be used to control conditional processing based on whether a particular format is supported by the user agent.

Definition of **'requiredFormats'**:

`requiredFormats` = **"<list-of-content-types>"**

A conditional processing attribute that controls conditional processing based on whether the specified formats are supported by the SVG user agent. The value is a list of Internet media types, with the individual values separated by white space. For a list of registered Internet media types (formerly called MIME types), see the IANA Media Type registry [MIMETYPES]. For a list of Internet media types types for audio and video codecs, see the IANA codec registry and *WAVE and AVI Codec Registries* [CODECS, RFC2361].

As with all conditional processing attributes, if **'requiredFormats'** is not specified, then it implicitly evaluates to true. However, if the attribute is specified, but has an empty string value, it evaluates to false. See Conditional processing overview for details on how conditional processing attribute influence document processing.

Animatable: no.

The following formats must always evaluate to true in conforming SVG viewers:

- image/png
- image/jpeg
- image/svg+xml

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-family-names>"**

A conditional processing attribute that controls conditional processing based on whether the specified fonts are available. The value is a list of font family names, using the same syntax as the **'font-family'** property, for example when processing quoted strings, multiple, leading and trailing spaces, and case sensitivity. Generic family names may not be used, however.

This attribute evaluates to true for the purpose of conditional processing if and only if all of the specified fonts are available, either installed on the system or as an SVG font defined or embedded within the document. As with all conditional processing attributes, if **'requiredFonts'** is not specified, then it implicitly evaluates to true. However, if the attribute is specified, but has an empty string value, it evaluates to false. See Conditional processing overview for details on how conditional processing attribute influence document processing.

Animatable: no.

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.

The **'externalResourcesRequired'** attribute 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"**

An attribute that specifies whether external resources are required for correct rendering of this element and its descendants.

false

(The lacuna 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 (for example the request for the required resource times out), progressive rendering is suspended, the `load` event is not fired for the element, and the document becomes in error (see Error processing). The document remains in error until all required resources become available.

Animatable: no.

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

Because setting **externalResourcesRequired="true"** on a container element will have the effect of disabling progressive display of the contents of that container, if that container includes elements that reference external resources, tools that generate SVG content 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.

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 a *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 an *end-tag* is read. The terms start-tag, end-tag and empty-element tag are as defined in *Extensible Markup Language (XML) 1.0* ([XML10], section 3.1) and *Extensible Markup Language (XML) 1.1* ([XML11], section 3.1).

When loading a document following the *start element* event on a node, that node becomes part of the document tree in the unresolved state. It is appended as the last child of the most recently opened element that is still open (that is, the most recent element for which a *start element* event has occurred with no corresponding *end element* event). 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.

When an *end element* event occurs for a **'script'** element, that element is processed according to the Script processing section of the Scripting chapter. Further parsing of the document will be blocked until processing of the **'script'** is complete.

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. Empty elements (elements without children) become resolved when the *end element* event occurs on the element; elements with child nodes become resolved when all their children are resolved and when the *end element* event occurs on the 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. For example, **'font'** element children (**'font-face'**, **'hkern'**, **'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.

Note that forward referencing from a **'discard'** element should be avoided when using progressive rendering. If it fails to find (and thus discard) an element, it will not later discard the element when it has finally loaded.

In Example progRend01 below, the **'g'** element rendering may 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 may start when the group has been fully parsed and myImage.png has been successfully retrieved.

Example: progRend01.svg

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

Example progRend02 demonstrates how progressive rendering is performed when there is a **'use'** element with a forward reference.

Example: progRend02.svg

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

The following list shows the possible renderings of the document as it is parsed (the rendering state follows the 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

Example progRend03 demonstrates how progressive rendering is performed when there is a **'use'** element with a forward reference and which has **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" 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>
```

The possible rendering states are as follows:

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

Example progRend04 shows a **'use'** element with a reference to an element that is in a container with **externalResourcesRequired="true"**.

Example: progRend04.svg

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

The possible rendering states as the document is parsed are as follows:

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 the **'g'**).
4. Yellow **'circle'** → *start element*: no update (rendering suspended because of **'g'**)
5. **'rect'** → *start element*: no update
6. **'g'** → *end element* (resources referenced by **'use'** become resolved and can be rendered, so rendering can proceed): green rect, pink circle, yellow circle, red rect

Example progRend05 shows an example of progressive rendering with a forward reference to an SVG font. Rendering updates do not occur mid-way through parsing a **'font'** element.

Example: progRend05.svg

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny" 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 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 0"
        ascent="917"
        descent="-250"
        alphabetic="0"/>
      <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 0"
        ascent="917"
        descent="-250"
        alphabetic="0"/>
      <missing-glyph d="..."/>
      <glyph unicode="A" glyph-name="C" d="..."/>
    </font>
  </defs>
</svg>
```

Rendering update possibilities as the document is parsed are as follows:

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

14. **#fontC** / **'font-face'** → *start element*: no update
15. **#fontC** / **'missingGlyph'** → *start element*: no update
16. **#fontC** / **'glyph'** → *start element*: no update
17. **#fontC** → *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 downloading 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 Tiny 1.2 does not require SVG user agents to download referenced media that is not visible 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 Tiny 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 SMIL 2.1 'prefetch' element ([SMIL21], section 4.4) has been incorporated into SVG as the **'prefetch'** element, with the following modifications:

- Attributes cannot be given **<percent-value>** values.
- The **'xlink:href'** attribute 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 playback can start.
- In order to adequately support non-local IRI references, the **'mediaCharacterEncoding'** and **'mediaContentEncodings'** attributes have been added.

The **'prefetch'** element provides a hint to the 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 document playback smoother. As it is a hint, user agents may ignore **'prefetch'** elements, although 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 rootmost 'svg' element.

To enable smooth playback during progressive downloading in this scenario, it is recommended that each adjacent top level **'g'** element contains 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='Number.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'>
          <choice>
            <ref name='Number.datatype'/>
            <value>auto</value>
          </choice>
        </attribute>
      </optional>
    </define>

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

Attribute definitions:

`mediaSize` = **"<long>"**

Defines how much of the media to fetch in bytes in terms 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 [RFC3629] and gzipped [RFC1952], then the size of the gzipped UTF-8 fragment applies. If that same document were decompressed and transcoded to UTF-16 [RFC2781] 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.

Animatable: no.

`mediaTime` = **"<clock-value>"**

Defines how much of the media to fetch in terms of the duration of the media. For discrete media (non-time based media such as PNG) using this attribute causes the entire resource to be prefetched.

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 beforehand (e.g. if the 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.

Animatable: no.

`bandwidth` = **"<long>"**

Defines how much network bandwidth, in bits per second, the SVG user agent should use when performing 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 that was used to calculate the size is that which is returned by the server.

Animatable: no.

`mediaContentEncodings` = **"<list-of-strings>"**

The **'mediaContentEncodings'** attribute is a white space separated list of the content encodings as defined in section 3.5 of HTTP/1.1 [RFC2616] (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 had 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 invalid IRI reference is an unsupported value. An empty attribute value (**xlink:href=""**) means that no prefetching will occur. The lacuna value is the empty string.

Animatable: no.

When **'prefetch'** refers to external media, if both **'mediaSize'** and **'mediaTime'** are specified, then **'mediaSize'** shall be used and **'mediaTime'** is ignored. If neither **'mediaSize'** nor **'mediaTime'** is specified, the behavior is that the entire resource should be fetched.

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.

Note that whereas the **'externalResourcesRequired'** attribute is used to designate that a resource is required, the **'prefetch'** element is used to optimize the retrieval of a resource. Setting the **'externalResourcesRequired'** attribute does not influence the behavior of the **'prefetch'** element and vice-versa. This is true whether the **'prefetch'** element points to an internal **'g'** element or external resource.

Example prefetch01 demonstrates the use 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/huge1.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/huge1.png"
        display="none">

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

        <animate attributeName="xlink:href" values="
               http://www.example.com/images/huge1.png;
               http://www.example.com/images/huge2.png;
               http://www.example.com/images/huge3.png"
            begin="15s" dur="30s"/>
     </image>

   </svg>
```

Example prefetch02 is an example of the **'prefetch'** element referring to a resource in the same document (i.e. a top level **'g'** element):

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="15s" mediaSize="627638" />

       </defs>

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

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

       <g xml: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

The **'id'**, **'xml:id'**, **'xml:base'**, **'class'**, **'role'**, **'rel'**, **'rev'**, **'about'**, **'content'**, **'datatype'**, **'property'**, **'resource'**, and **'typeof'** attributes are available on all elements defined by SVG Tiny 1.2. Some of these elements, such as **'id'**, **'xml:id'**, and **'xml:base'** may have a direct effect on the structure and rendering of SVG, while others may only affect SVG indirectly, or may be used only for auxiliary processing of SVG content. See extensible metadata attributes for more details.

Attribute definitions:

`id` = **"<NCName>"**

This attribute specifies a unique identifier for the element. Because of wider use and compatibility with legacy content and existing user agents (including authoring tools), it is recommended that content intended for use in a Web browser environment use **'id'** instead of **'xml:id'**. (See details on ID attributes below.)

Animatable: no.

`xml:id` = **"<NCName>"**

This attribute specifies a unique identifier for the element. Refer to *xml:id Version 1.0* [XMLID]. It is recommended that content intended for use with generic XML processing tools, particularly in a scenario where the datatype of the **'id'** attribute is not known, use **'xml:id'**. (See details on ID attributes below.)

Animatable: no.

`xml:base` = **"<IRI>"**
: This attribute specifies a base IRI other than the base IRI of the document or external entity. Refer to *XML Base* [XML-BASE].
Animatable: no.

`class` = **"<XML-NMTOKENS>"**
: The **'class'** attribute assigns one or more class names to an element. The element may be said to belong to these classes. A class name may be shared by several element instances. The class attribute has several roles:

- As a style sheet selector (when an author wishes to assign style information to a set of elements). Note: SVG Tiny 1.2 does not mandate the support of style sheets.
- For general purpose processing by user agents.

The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.
Animatable: yes.

`role` = **"<list-of-strings>"**
: The **'role'** attribute assigns one or more role values to an element. The element may be said to have these roles. A role value may be shared by several element instances. Unlike the **'class'** attribute, **'role'** attribute values are intended to be selected from a predefined set of values with specific semantic aspects that are assigned to the element, such as those defined in the ARIA ontology [ARIA], XHTML Role Attribute Module [ROLE], XHTML Vocabulary collection [XHTMLVOCAB], and in future SVG specifications.
The **'role'** attribute is intended to functionally align with the XHTML Role Attribute Module [ROLE].
The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.
Animatable: yes.

`rel` = **"<list-of-strings>"**
: The **'rel'** attribute assigns one or more relationship values to an element. The value of the **'rel'** attribute expresses the relationships between two resources. For **'a'** elements in particular, the **'rel'** attribute indicates the relationship that the linked resource holds for the element's children or the element's containing document.
This attribute is an analog of the HTML [HTML4] attribute of the same name. It is intended to be used in the same manner, such as with RDFa [RDFA], Microformats [MF], and other semantic purposes.
The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.
Animatable: yes.

`rev` = **"<list-of-strings>"**
: The **'rev'** attribute assigns one or more relationship values to an element. The value of the **'rev'** attribute expresses the reverse relationships between two resources. For **'a'** elements in particular, the **'rev'** attribute indicates the relationship that the element's children or the element's containing document holds for the linked resource.
This attribute is an analog of the HTML [HTML4] attribute of the same name. It is intended to be used in the same manner, such as with RDFa [RDFA], Microformats [MF], and other semantic purposes.
The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.
Animatable: yes.

`about` = **"<list-of-strings>"**
: The **'about'** attribute assigns one or more relationship values to an element. The value of the **'about'** attribute is intended to be used for stating the subject of the element's data.
This attribute is intended to functionally align with the attribute of the same name in the RDFa [RDFA] specification, but is not limited to use with that format.
The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.
Animatable: yes.

`content` = **"<string>"**

The **'content'** attribute provides a plain text value that may be suitable for humans, or may be machine-readable, or both, depending on the context. In general, this should only be used to supplement textual child content, or to be used on elements which do not normally take text as child content.

This attribute is intended to functionally align with the attribute of the same name in the RDFa [RDFA] specification, but is not limited to use with that format.

Animatable: yes.

`datatype` = **"<string>"**

The **'datatype'** attribute specifies a semantic datatype for the content of the element, or the value of the **'content'** attribute if one is provided for the element.

This attribute should not be confused with the **'type'** attribute, and has no direct effect on the rendering or execution of the element.

This attribute is intended to functionally align with the attribute of the same name in the RDFa [RDFA] specification, but is not limited to use with that format.

Animatable: yes.

`property` = **"<list-of-strings>"**

The **'property'** attribute is used for expressing relationships between the element's subject (e.g., the text content of a child node or of an attribute value) and a set of known or referenced properties.

This attribute is intended to functionally align with the attribute of the same name in the RDFa [RDFA] specification, but is not limited to use with that format.

The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.

Animatable: yes.

`resource` = **"<string>"**

The **'resource'** attribute associates a resource, typically expressed with an IRI reference, to the element, in a manner that does not normally resolve the IRI reference.

This attribute is intended to functionally align with the attribute of the same name in the RDFa [RDFA] specification, but is not limited to use with that format.

Animatable: yes.

`typeof` = **"<list-of-strings>"**

The **'typeof'** attribute associates one or more datatypes with the element. This attribute should not be confused with the **'type'** attribute, and has no direct effect on the rendering or execution of the element.

This attribute is intended to functionally align with the attribute of the same name in the RDFa [RDFA] specification, but is not limited to use with that format.

The attribute value indicates membership in one or more sets. Any number of elements may be assigned to the same set. Multiple set names must be separated by white space characters.

Animatable: yes.

The 'id' and 'xml:id' attributes

Both the **'id'** and **'xml:id'** attributes specify a unique identifier for the element. Both are have the data type <NCName>, but are of type <ID> for purposes of validation. **'xml:id'** is intended to represent type **<ID>** universally across all document types. This makes it more suitable for certain compound documents with arbitrary XML, or with generic XML toolchains which require explicit knowledge of **<ID>**-typed attributes.

It is strongly recommended that SVG generators only use **'id'** to assign identity to elements, to maintain backwards compatibility with existing viewers, authoring tools, and other content.

There remains only one single `id` field on the `SVGElement` interface, which can be used to change the value of either attribute (e.g. by using the `setAttributeNS()`, `setTraitNS()`, or `setTrait()` methods). Likewise, the `getElementById` method on the `Document` interface applies equally to both the **'id'** and **'xml:id'** attributes.

Because they are intended for different environments, the **'id'** and **'xml:id'** attributes must not be used together on SVG elements in the same document. Such documents are not conforming SVG 1.2 Tiny content, and the behavior is not specified.

Transforming between 'id' and 'xml:id'

In order to facilitate the creation of content that can be used in both primary scenarios (that is, in existing desktop browsers and authoring tools, and in XML toolchains), an XSLT stylesheet can be used to convert the **'id'** attributes to **'xml:id'** attributes (and vice versa). For example, when content that has been authored with the browser environment in mind is being prepared for consumption by a generic XML tool, it can be preprocessed by using the id2xmlid.xsl sample stylesheet. This allows the same content to be used with little overhead or risk of breaking content. Example transformation stylesheets are provided below:

Example: id2xmlid.xsl

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="node() | @*"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@id">
    <xsl:attribute name="xml:id">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>

</xsl:stylesheet>
```

Example: xmlid2id.xsl

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="node() | @*"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@xml:id">
    <xsl:attribute name="id">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>

</xsl:stylesheet>
```

5.10.2 Attributes for character-content elements

Elements that might contain character data content have attributes **'xml:lang'** and **'xml:space'** to specify the natural language and whitespace processing of the content.

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` = **"<language-id>"**

This is a standard XML attribute used to specify the language (e.g., English) used in the child text content and attribute values of the element it occurs on. The value is either a language tag as defined in IETF BCP 47 [BCP 47] or the empty string, **""**. Refer to *Extensible Markup Language (XML) 1.0* ([XML10], section 2.12) and *Extensible Markup Language (XML) 1.1* ([XML11], section 2.12) for the definition of this attribute.

Animatable: *no.*

`xml:space` = **"default" | "preserve"**

This is a standard XML attribute used to specify whether white space is preserved in character data. The only possible values are **"default"** and **"preserve"**. Refer to *Extensible Markup Language (XML) 1.0* ([XML10], section 2.10) and *Extensible Markup Language (XML) 1.1* ([XML11], section 2.10) for the definition of this attribute. See also the discussion of white space handling for text content elements in SVG.

Animatable: *no.*

6 Styling

Contents

6.1 SVG's styling properties 56
6.2 Usage scenarios for styling 57
6.3 Specifying properties using the presentation attributes 57
6.4 Styling with XSL 58
6.5 Case sensitivity of property names and values 59
6.6 Facilities from CSS and XSL used by SVG 59
6.7 Property inheritance and computation 59

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 CSS 2 specification [CSS2]. **Note:** The CSS 2 specification is no longer maintained, and implementors may wish to refer instead to its future replacement, CSS 2.1 [CSS21], for more precise details. SVG 1.2 Tiny refers to CSS 2 due to the maturity of that specification on the W3C Recommendation track.

The following properties are shared between CSS 2 and SVG. Apart from **'display'**, these properties are also defined in XSL:

- Font properties:
  - **'font-family'**
  - **'font-size'**
  - **'font-style'**
  - **'font-weight'**
- Text properties:
  - **'direction'**
  - **'unicode-bidi'**
- Other properties for visual media:
  - **'color'**, which is used to provide a potential indirect value (**currentColor**) for the **'fill'**, **'stroke'** and **'stop-color'** properties
  - **'display'**
  - **'visibility'**

The following SVG properties are not defined in CSS 2. 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:
  - **'buffered-rendering'**
  - **'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 external, author stylesheets 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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     viewBox="0 0 1000 500">
  <rect x="200" y="100" width="600" height="300"
        fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

The presentation attributes offer the following advantages:

- **Broad support**. All versions of Conforming SVG Interpreters and Conforming SVG Viewers are required to support the presentation attributes.
- **Simplicity**. Styling properties can be attached to elements by simply providing a value for the presentation attribute on the proper elements.
- **Restyling**. SVG content that uses the presentation attributes is highly compatible with downstream processing using XSLT [XSLT] 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.

Note: An **!important** declaration ([CSS2], section 6.4.2) within a presentation attribute definition is unsupported and causes that attribute to have an unsupported value.

Note: there are no presentation attributes for shorthand properties ([CSS2], section 1.3.3), only for the individual properties that make up the shorthand. (In XML, attribute order is not significant.)

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 [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"/>
  <!-- Add version to topmost 'svg' element -->
  <xsl:template match="/svg:svg">
```

```
      <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:attribute name="version">1.2</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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="10cm" height="5cm" viewBox="0 0 100 50">
  <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" version="1.2" baseProfile="tiny"
     width="10cm" height="5cm" viewBox="0 0 100 50">
  <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, 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. Many of SVG's properties are shared between CSS 2, XSL and SVG. (See list of shared properties).

6.7 Property inheritance and computation

SVG supports property inheritance to child elements. In the definition of each property it is stated whether it is inherited or not. Inherited properties inherit the computed value, and not the specified value. For the calculation of computed values, see the definition of each property. Note that the keyword **inherit** may be used to force the property value of the parent to be used, even for non-inherited properties.

7 Coordinate Systems, Transformations and Units

Contents

7.1 Introduction 60
7.2 The initial viewport 60
7.3 The initial coordinate system 61
7.4 Coordinate system transformations 62
7.5 Nested transformations 66
7.6 The 'transform' attribute 67
7.6.1 The TransformList value 68

7.7 Constrained transformations 70
7.7.1 The user transform 70
7.7.2 ViewBox to viewport transformation 70
7.7.3 Element transform stack 70
7.7.4 The current transformation matrix 71
7.7.5 The TransformRef value 71

7.8 The 'viewBox' attribute 74
7.9 The 'preserveAspectRatio' attribute 76
7.10 Establishing a new viewport 77
7.11 Units 78
7.12 Bounding box 78
7.13 Object bounding box units 81
7.14 Intrinsic sizing properties of the viewport of SVG content 82
7.15 Geographic coordinate systems 83
7.16 The 'svg:transform' attribute 83

7.1 Introduction

For all media, the canvas describes "the space where the SVG content is rendered." The canvas is infinite for each dimension of the space, but rendering occurs relative to a finite rectangular region of the canvas. This finite rectangular region is called the SVG viewport. For visual media ([CSS2], section 7.3.1), the SVG viewport is the viewing area where the user sees the SVG content.

The size of the SVG viewport (i.e., its width and height) is determined by a negotiation process (see Establishing the size of the initial viewport) between the SVG document fragment and its parent (real or implicit). Once the viewport is established, the SVG user agent must establish the initial viewport coordinate system and the initial user coordinate system (see Initial coordinate system). The viewport coordinate system is also called *viewport space* and the user coordinate system is also called *user space*.

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 (see Coordinate system transformations). 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, one 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.

If the parent document format defines rules for referenced or embedded graphics content, then the negotiation process is determined by the parent document format specification. If the parent document is styled with CSS, then the negotiation process must follow the CSS rules for replaced elements. If there are CSS width and height properties (or corresponding XSL properties) on the referencing element (or rootmost 'svg' element for inline SVG content) that are sufficient to establish the width and height of the viewport, then these positioning properties establish the viewport's width, height, and aspect ratio.

If there is no parent document, the SVG user agent must use the **'width'** and **'height'** attributes on the rootmost 'svg' element element as the width and height for the viewport.

Note that the time at which the viewport size negotiation is finalized is implementation-specific. Authors who need to be sure of the dimensions of the viewport should do so with load-event or resize-event handlers.

7.3 The initial coordinate system

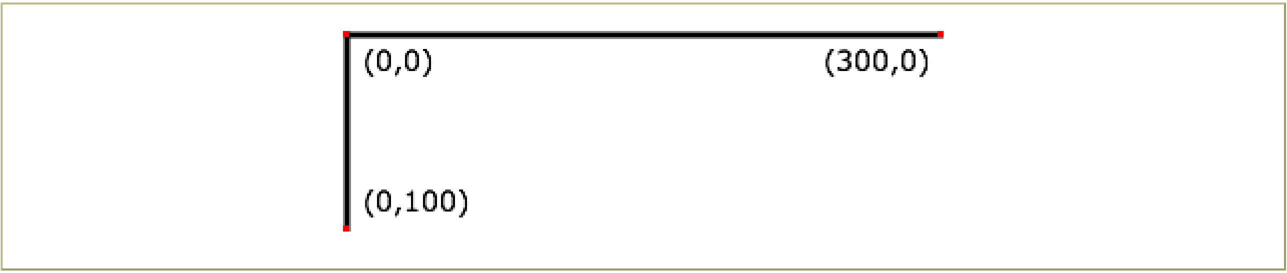
For the **'svg'** element, the SVG user agent must establish an initial viewport coordinate system and an initial user coordinate system such that the two coordinates systems are identical. The origin of both coordinate systems must be at the origin of the viewport, and one unit in the initial coordinate system must equal one "pixel" (i.e., a *px* unit as defined in CSS ([CSS2], section 4.3.2)) in the viewport. In most cases, such as stand-alone SVG documents or SVG document fragments embedded (by reference or inline) within XML parent documents where the parent's layout is determined by CSS [CSS2] or XSL [XSL], the SVG user agent must establish the initial viewport coordinate system (and therefore the initial user coordinate system) such that its origin is 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 CSS ([CSS2], section 4.3.2).

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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="300px" height="100px">

  <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>
```



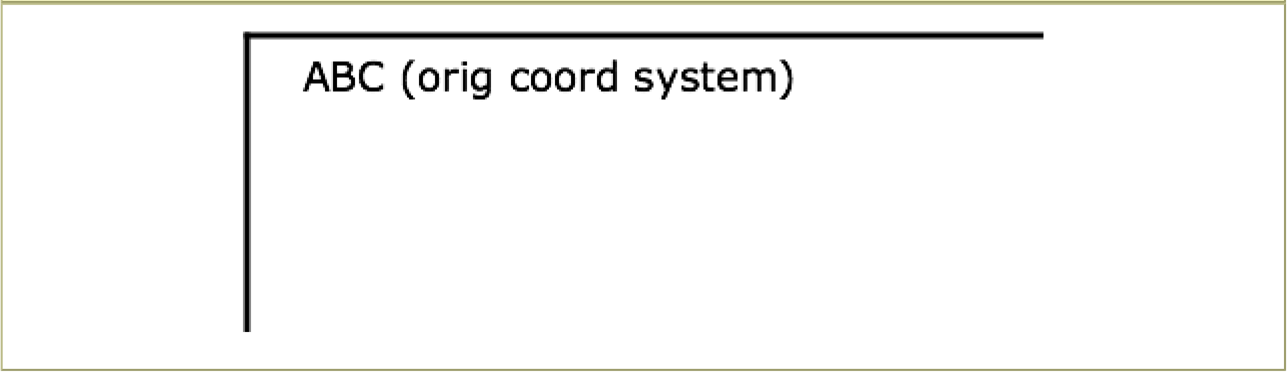
7.4 Coordinate system transformations

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 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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="400px" height="150px">

  <desc>Example OrigCoordSys - Simple transformations: original picture</desc>

  <g fill="none" stroke="black" stroke-width="3">
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5"/>
    <line x1="1.5" y1="0" x2="1.5" y2="150"/>
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana">
      ABC (orig coord system)
    </text>
  </g>
</svg>
```

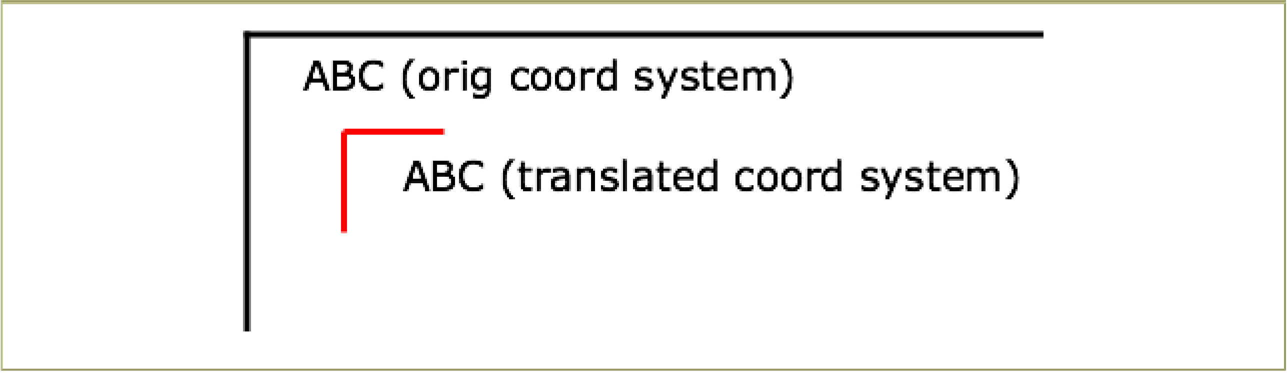


Example 07_04 establishes a new user coordinate system by specifying **transform="translate(50,50)"** on the third **'g'** element below. The new user coordinate system has its origin at location (50,50) in the original coordinate system. The result of this transformation is that the coordinate (30,30) 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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="400px" height="150px">
```

```
  <desc>Example NewCoordSys - New user coordinate system</desc>

  <g fill="none" stroke="black" stroke-width="3">
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5"/>
    <line x1="1.5" y1="0" x2="1.5" y2="150"/>
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana">
      ABC (orig coord system)
    </text>
  </g>
  <!-- Establish a new coordinate system, which is
       shifted (i.e., translated) from the initial coordinate
       system by 50 user units along each axis. -->
  <g transform="translate(50,50)">
    <g fill="none" stroke="red" stroke-width="3">
      <!-- Draw lines of length 50 user units along
           the axes of the new coordinate system -->
      <line x1="0" y1="0" x2="50" y2="0" stroke="red"/>
      <line x1="0" y1="0" x2="0" y2="50"/>
    </g>
    <text x="30" y="30" font-size="20" font-family="Verdana">
      ABC (translated coord system)
    </text>
  </g>
</svg>
```



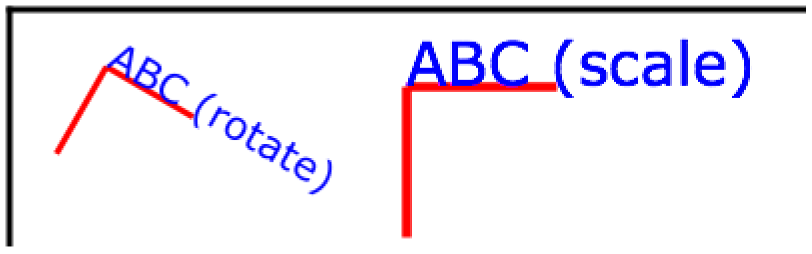
Example 07_05 illustrates simple **rotate** and **scale** transformations. The example defines two new coordinate systems:

- 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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="400px" height="120px">

  <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">
```

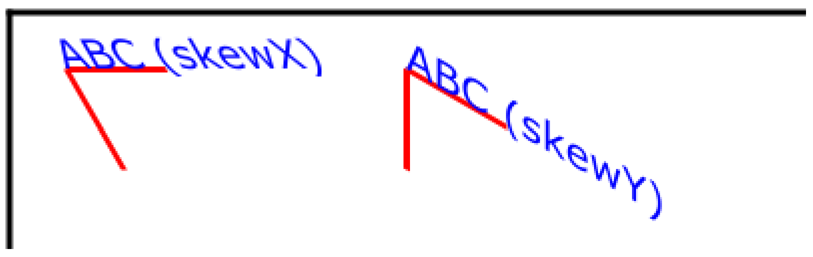
```
        ABC (rotate)
      </text>
    </g>
  </g>
  <!-- Establish a new coordinate system whose origin is at (200,40)
       in the initial coord. system and which is scaled by 1.5. -->
  <g transform="translate(200,40)">
    <g transform="scale(1.5)">
      <g fill="none" stroke="red" stroke-width="3">
        <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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="400px" height="120px">

  <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:

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

Simple transformations are represented in matrix form as follows:

- Translation is equivalent to the matrix:

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

  or **[1 0 0 1 tx ty]**, where *tx* and *ty* are the distances to translate coordinates in *x* and *y*, respectively.
- Scaling is equivalent to the matrix:

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

  or **[sx 0 0 sy 0 0]**. One unit in the *x* and *y* directions in the new coordinate system equals *sx* and *sy* units in the previous coordinate system, respectively.
- Rotation about the origin is equivalent to the matrix:

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

  or **[cos(a) sin(a) -sin(a) cos(a) 0 0]**, which has the effect of rotating the coordinate system axes by angle *a*.
- A skew transformation along the x-axis is equivalent to the matrix:

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

  or **[1 0 tan(a) 1 0 0]**, which has the effect of skewing *x* coordinates by angle *a*.
- A skew transformation along the y-axis is equivalent to the matrix:

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

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

7.5 Nested transformations

Transformations can be nested to any level. The effect of nested transformations is to post-multiply (i.e., concatenate) the subsequent transformation matrices onto previously defined transformations:

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

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

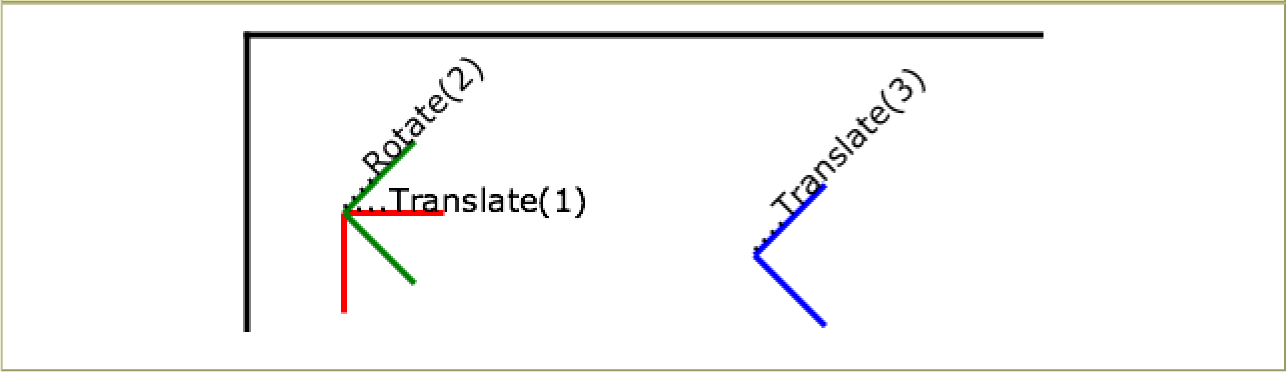
$$CTM = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \ldots \cdot \begin{bmatrix} a_n & c_n & e_n \\ b_n & d_n & f_n \\ 0 & 0 & 1 \end{bmatrix}$$

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

Example 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>
  </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:

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

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

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

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

7.6 The **'transform'** attribute

Attribute definition:

`transform` = **"<transform-list>" | "<transform-ref>" | "none"**

This attribute specifies a coordinate system transformation to apply to the element it appears on. The value of this attribute takes one of three forms:

<transform-list>

Specifies a list of affine transformations. See the definition in The TransformList value section below for details.

<transform-ref>

Specifies a constrained transformation. See the definition in The TransformRef value section below for details.

none

Specifies the identity transformation. Using this value has the same effect on the element's CTM as using the identity matrix (**transform="matrix(1 0 0 1 0 0)"**) or not specifying the **'transform'** attribute at all. This is the lacuna value.

Animatable: yes.

If the **'transform'** attribute cannot be parsed according to the syntaxes above, then it has an unsupported value. In this case, as with any instance of an unsupported value, the SVG user agent must process the element as if the **'transform'** attribute had not been specified, which will result in the element's transformation being the identity transformation.

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 white space and/or a comma. The available types of transform definitions are as follows:

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

  If optional parameters **<cx>** and **<cy>** are not supplied, the rotation 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 rotation is about the point (*cx*, *cy*). The operation represents the equivalent of the following specification: **translate(<cx>, <cy>) rotate(<rotate-angle>) translate(-<cx>, -<cy>)**.
- **skewX(<skew-angle>)**, which specifies a skew transformation along the x-axis.
- **skewY(<skew-angle>)**, which specifies a skew transformation along the y-axis.

All numeric values are real <number>s.

If the list of transforms includes a matrix with all values set to zero (that is, **'matrix(0,0,0,0,0,0)'**), then rendering of the element is disabled. Such a value is not an unsupported value.

If a list of transforms includes more than one transform definition, 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>
```

will have the same rendering as:

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 an EBNF grammar for **<transform-list>** values [EBNF]:

```
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 container elements and graphics element 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 SVG 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 SVG 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 = Matrix corresponding to the user scale  (currentScale on SVGSVGElement)
UP = Matrix corresponding to the user pan    (currentTranslate on SVGSVGElement)
UR = Matrix corresponding to the 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

Some 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 transform defined by the **'transform'** attribute on the given element with identifier *id*:

```
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 xml:id="root" version="1.2" baseProfile="tiny">
  <g xml:id="g" transform="scale(2)">
    <rect xml:id="r" transform="scale(4)"/>
    <g xml:id="g2">
      <rect xml: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(2)   (where I is the identity matrix)
TS(r2) = TS(g) . scale(0.5) = scale(1)
```

7.7.4 The current transformation matrix

Each element in the rendering tree has the concept of a current transformation 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 SVG 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 transformation matrix

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

With the above definitions for U, VB, and TS, the CTM for the rectangle with **xml:id="elt"** is computed as follows:

```
CTM(elt) = U . VB(root) . TS(elt)
         = U . VB(root) . Txf(g-n) . [...] . Txf(g-0) . Txf(elt)
```

Example: Current transformation matrix, n=2

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

This produces the following transformations:

```
CTM(elt) = U . VB(root) . Txf(g-1) . Txf(g-0) . Txf(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 TransformRef value

By using the **'ref(...)'** attribute value on the **'transform'** attribute it is possible to specify simple constrained transformations.

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

Note that the inverse of the parent element's CTM may not always exist. In such cases, the user agent can instead calculate the CTM of the element with the constrained transformation by looking up the CTM of the rootmost 'svg' element directly. The **'ref(...)'** value in this case is not an unsupported value.

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, and using "svg[0]" to denote the rootmost 'svg' element:

```
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)
```

Thus, 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 SVG user agent viewport is a square with sides of 200 units.

```
<svg xml:id="root" version="1.2" baseProfile="tiny" viewBox="0 0 100 100">
  <line x1="0" x2="100" y1="0" y2="100"/>
  <rect xml: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 (due to `currentTranslate`, `currentScale` and `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 SVG user agent viewport is a square with sides of 200 units.

```
<svg xml:id="root" version="1.2" baseProfile="tiny" viewBox="0 0 100 100">
  <line x1="0" x2="100" y1="0" y2="100"/>
  <g xml:id="g" transform="ref(svg, 50, 50)">
    <rect xml: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 SVG 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 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.

The following is an EBNF grammar for **<transform-ref>** values [EBNF]:

```
transform-ref ::=
    wsp* ref wsp*
ref ::=
    "ref" wsp* "(" wsp* "svg" wsp* ")"
    | "ref" wsp* "(" wsp* "svg" comma-wsp number comma-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)
```

7.8 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) can have the **'viewBox'** attribute specified on them.

Attribute definition:

`viewBox` = **"<list-of-numbers>" | "none"**

Specifies a rectangular region into which child graphical content must be fit. The value of this attribute takes one of two forms:

<list-of-numbers>

A list of four **<number>**s (**<min-x>**, **<min-y>**, **<width>** and **<height>**), separated by white space and/or a comma, which specify a rectangle in viewport space which must be mapped to the bounds of the viewport established by the given element, taking into account the **'preserveAspectRatio'** attribute. 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 a supplemental transformation due to the **'viewBox'** attribute must not be used. Using this value will have the same affect on child content as not specifying the **'viewBox'** attribute at all. This is the lacuna value.

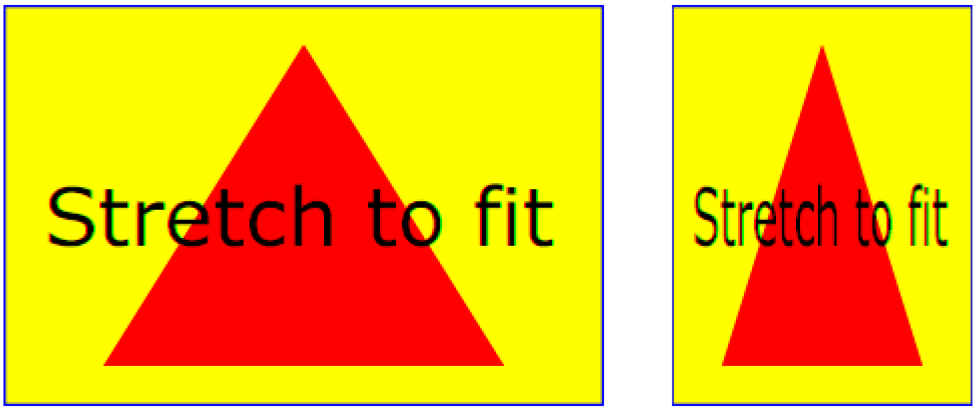
Animatable: yes.

A negative value for **<width>** or **<height>** is unsupported. A value of zero for either of these two parameters 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 must stretch to fit the bounds of the viewport.

Example: 07_12.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="300px" height="200px" viewBox="0 0 1500 1000"
     preserveAspectRatio="none">

  <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 SVG 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 SVG 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 xmlns="http://www.w3.org/2000/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 SVG 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 xmlns="http://www.w3.org/2000/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 SVG user agent will need to supply a **translate** transformation in addition to a **scale** transformation. For example, on an 'svg' element, a **translate** transformation will be needed if the 'viewBox' attribute specifies values other than zero for **<min-x>** or **<min-y>**.)

Unlike the 'transform' attribute (see effect of the 'transform' attribute 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.

The following is an EBNF grammar for values of the 'viewBox' attribute [EBNF]:

```
viewbox ::=
    wsp* viewboxSpec wsp*
viewboxSpec ::=
    number comma-wsp number comma-wsp number comma-wsp number
    | "none"
```

```
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)
```

7.9 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.

'preserveAspectRatio' 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.

Attribute definition:

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

defer

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 must be used. If the referenced content lacks a value for **'preserveAspectRatio'** then the **'preserveAspectRatio'** attribute must be processed as normal (ignoring **defer**). For **'preserveAspectRatio'** on all other elements the **defer** portion of the attribute is ignored.

<align>

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 lacuna value) - 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.

<meet>

Optional and 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**.

meet indicates to scale the graphic such that:

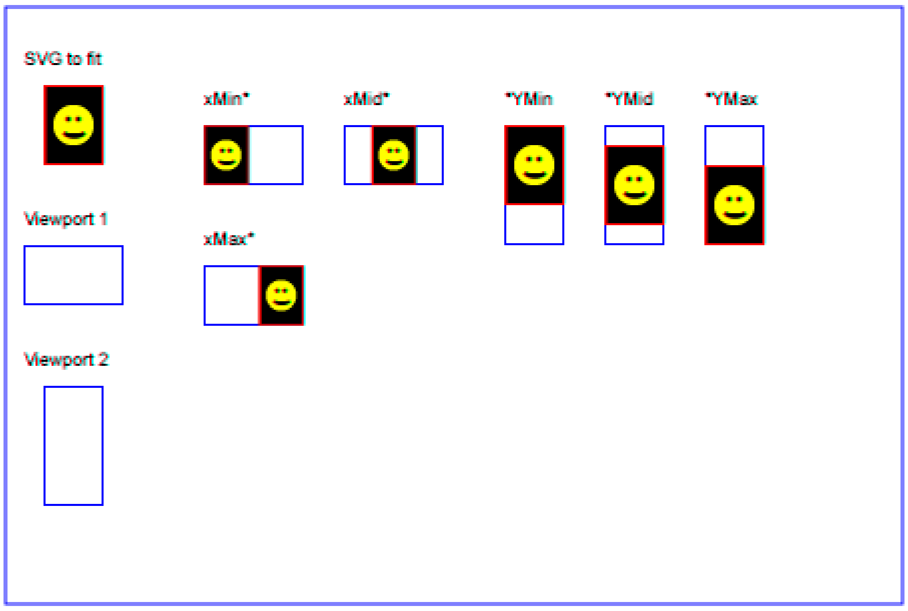
- aspect ratio is preserved
- the entire **'viewBox'** is visible within the viewport
- the **'viewBox'** is scaled up as much as possible, while still meeting the other criteria

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).

Animatable: yes.

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



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.
- The **'foreignObject'** element.

The following paragraph is informative.

Note that no clipping of overflow is performed, but that such clipping will take place if the content is viewed in an SVG user agent that supports clipping (e.g. a user agent that supports SVG 1.1 Full [SVG11]), since the initial value for the **'overflow'** property is **hidden** for non-root elements that establish viewports ([SVG11], section 14.3.3). Content authors that want content to be fully forward compatible are advised to either specify the **'overflow'** property or to make sure that content that shouldn't be clipped is inside of the established viewport.

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'** XML attributes on 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: in, cm, mm, pt, pc, px and percentages (%). These values on **'width'** and **'height'** contribute towards the calculation of the initial viewport.

Using percentage values on **'width'** and **'height'** attributes mandates how much space the SVG viewport must take of the available initial viewport. In particular:

- 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.

See the discussion on the initial viewport for more details.

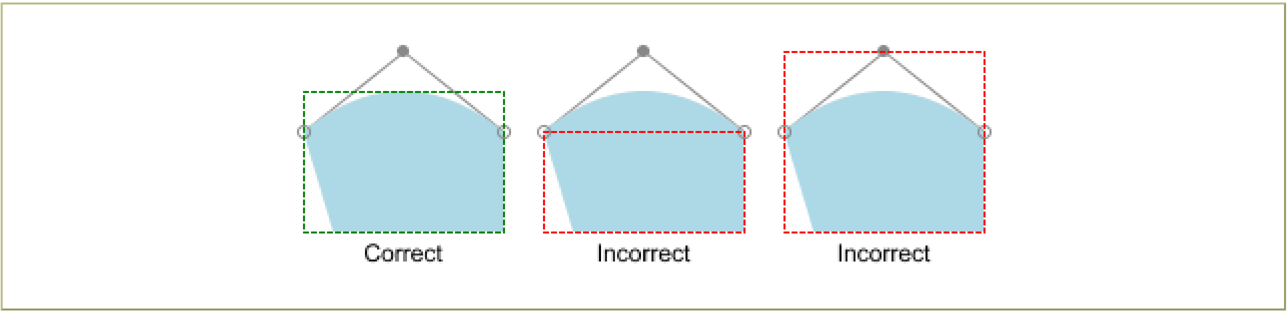
7.12 Bounding box

The bounding box (or "bbox") of an element is the tightest fitting rectangle aligned with the axes of that element's user coordinate system that entirely encloses it and its descendants. The bounding box must be computed exclusive of any values for the **fill** related properties, the **stroke** related properties, the **opacity** related properties or the **visibility** property. For curved shapes, the bounding box must enclose all portions of the shape along the edge, not just end points. Note that control points for a curve which are not defined as lying along the line of the resulting curve (e.g., the second coordinate pair of a Cubic Bézier command) must not contribute to the dimensions of the bounding box (though those points may fall within the area of the bounding box, if they lie within the shape itself, or along or close to the curve). For example, control points of a curve that are at a further distance than the curve edge, from the non-enclosing side of the curve edge, must be excluded from the bounding box.

Example bbox01 shows one shape (a **'path'** element with a quadratic Bézier curve) with three possible bounding boxes, only the leftmost of which is correct.

Example: bbox01.svg

```
<svg xmlns='http://www.w3.org/2000/svg'
     xmlns:xlink='http://www.w3.org/1999/xlink'
     version='1.1' width='380px' height='120px' viewBox='0 0 380 120'>

  <title>Bounding Box of a Path</title>
  <desc>
    Illustration of one shape (a 'path' element with a quadratic Bézier) with
    three bounding boxes, only one of which is correct.
  </desc>

  <defs>
    <g id='shape'>
      <line x1='120' y1='50' x2='70' y2='10' stroke='#888'/>
      <line x1='20' y1='50' x2='70' y2='10' stroke='#888'/>
      <path stroke-width='2' fill='rgb(173, 216, 230)' stroke='none' fill-rule='evenodd'
            d='M20,50
               L35,100
               H120
               V50
               Q70,10 20,50'/>
      <circle cx='120' cy='50' r='3' fill='none' stroke='#888'/>
      <circle cx='20' cy='50' r='3' fill='none' stroke='#888'/>
      <circle cx='70' cy='10' r='3' fill='#888' stroke='none'/>
    </g>
  </defs>

  <g text-anchor='middle'>
    <g>
      <title>Correct Bounding Box</title>
      <use xlink:href='#shape'/>
      <rect x='20' y='30' width='100' height='70'
            fill='none' stroke='green' stroke-dasharray='2' stroke-linecap='round'/>
      <text x='70' y='115'>Correct</text>
    </g>

    <g transform='translate(120)'>
      <title>Incorrect Bounding Box</title>
      <desc>Bounding box does not encompass entire shape.</desc>
      <use xlink:href='#shape'/>
      <rect x='20' y='50' width='100' height='50'
            fill='none' stroke='red' stroke-dasharray='2' stroke-linecap='round'/>
      <text x='70' y='115'>Incorrect</text>
    </g>

    <g transform='translate(240)'>
      <title>Incorrect Bounding Box</title>
      <desc>Bounding box includes control points outside shape.</desc>
      <use xlink:href='#shape'/>
      <rect x='20' y='10' width='100' height='90'
            fill='none' stroke='red' stroke-dasharray='2' stroke-linecap='round'/>
      <text x='70' y='115'>Incorrect</text>
    </g>
  </g>
</svg>
```



The bounding box must be applicable for any rendering element with positive **'width'** or **'height'** attributes and with a **'display'** property other than **none**, as well as for any container element that may contain such elements. Elements which do not partake in the rendering tree (e.g. elements in a **'defs'** element, elements whose **'display'** is **none**, etc.), and which have no child elements that partake in the rendering tree (e.g. **'g'** elements with no children), shall not contribute to the bounding box of the parent element. Elements that do not contribute to the bounding box of a parent element must still return their own bounding box value when required.

To illustrate, example bbox-calc below shows a set of elements. Given this example, the following results shall be calculated for each of the elements.

Example: bbox-calc.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>Bounding Box Calculation</title>
  <desc>Examples of elements with different bounding box results based on context.</desc>

  <defs id="defs-1">
     <rect id="rect-1" x="20" y="20" width="40" height="40" fill="blue" />
  </defs>

  <g id="group-1">
    <use id="use-1" xlink:href="#rect-1" x="10" y="10" />

    <g id="group-2" display="none">
      <rect id="rect-2" x="10" y="10" width="100" height="100" fill="red" />
    </g>
  </g>
</svg>
```

Element ID	Bounding Box Result
"defs-1"	{0, 0, 0, 0}
"rect-1"	{20, 20, 40, 40}
"group-1"	{30, 30, 40, 40}
"use-1"	{30, 30, 40, 40}
"group-2"	{10, 10, 100, 100}
"rect-2"	{10, 10, 100, 100}

Elements and document fragments which derive from `SVGLocatable` but are not in the rendering tree, such as those in a **'defs'** element or those which have been been created but not yet inserted into the DOM, must still have a bounding box. The geometry of elements outside the rendering tree must take into account only those properties and values (such as **'font-size'**) which are specified within that element or document fragment, or which have a lacuna value or an implementation-defined value.

For text content elements, for the purposes of the bounding box calculation, each glyph must be treated as a separate graphics element. The calculations must assume that all glyphs occupy the full glyph cell. For example, for horizontal text, the calculations must assume that each glyph extends vertically to the full ascent and descent values for the font. An exception to this is the **'textArea'**, which uses that element's geometry for the bounding box calculation.

Because declarative or scripted animation can change the shape, size, and position of an element, the bounding box is mutable. Thus, the bounding box for an element shall reflect the current values for the element at the snapshot in time at which the bounding box is requested, whether through a script call or as part of a declarative or linking syntax.

Note that an element which has either or both of **'width'** and **'height'** of **'0'** (such as a vertical or horizontal line, or a **'rect'** element with an unspecified **'width'** or **'height'**) still has a bounding box, with a positive value for the positive dimension, or with **'0'** for both **'width'** and **'height'** if no positive dimension is specified. Similarly, subpaths segments of a **'path'** element with **'0'** **'width'** and **'height'** must be included in that element's geometry for the sake of the bounding box. Note also that elements which do not derive from `SVGLocatable` (such as gradient elements) do not have a bounding box, thus have no interface to request a bounding box.

Elements in the rendering tree which reference unresolved resources shall still have a bounding box, defined by the position and dimensions specified in their attributes, or by the lacuna value for those attributes if no values are supplied. For example, the element `<use xlink:href="#bad" x="10" y="10"/>` would have a bounding box with an **'x'** and **'y'** of **'10'** and a **'width'** and **'height'** of **'0'**.

For a formal definition of bounding boxes, see [FOLEY-VANDAM], section 15.2.3, Extents and Bounding Volumes. For further details, see bounding box calculations, the effects of visibility on bounding box, object bounding box units and text elements, and fragment identifiers.

7.13 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'	**gradientUnits="objectBoundingBox"**	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'	**gradientUnits="objectBoundingBox"**	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.

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 are 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.

7.14 Intrinsic sizing properties of the viewport of SVG content

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

The intrinsic aspect ratio of the viewport of SVG content is necessary for example, when including SVG from an object element in XHTML styled with CSS. It is possible (indeed, common) for an SVG graphic to have an intrinsic aspect ratio but not to have an intrinsic width or height. The intrinsic aspect ratio must be calculated based upon the following rules:

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

Examples:

Example: Intrinsic Aspect Ratio 1

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

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

Example: Intrinsic Aspect Ratio 2

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

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

Example: Intrinsic Aspect Ratio 3

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

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

Example: Intrinsic Aspect Ratio 4

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

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

7.15 Geographic coordinate systems

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

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

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

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

7.16 The **'svg:transform'** attribute

Attribute definition:

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

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

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

Animatable: no.

This attribute describes an optional additional affine transformation that may have been applied during this mapping. This attribute may be added to the OpenGIS **'CoordinateReferenceSystem'** element. Note that, unlike the **'transform'** attribute, it does not indicate that a transformation is to *be applied* to the data within the file. Instead, it simply describes the transformation that *was already applied* to the data when being encoded in SVG.

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

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

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

Example: 07_19.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="100" height="100" viewBox="0 0 1000 1000">

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

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

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

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

Example: 07_20.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="100" height="100" viewBox="0 0 1000 1000">

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

  <metadata>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:crs="http://www.ogc.org/crs"
             xmlns:svg="http://wwww.w3.org/2000/svg">
      <rdf:Description rdf:about="">
        <!-- In case of a well-known Coordinate Reference System
             an 'Identifier' is enough to describe the CRS -->
        <crs:CoordinateReferenceSystem svg:transform="rotate(-90) scale(100, 100)">
          <crs:Identifier>
            <crs:code>4326</crs:code>
            <crs:codeSpace>EPSG</crs:codeSpace>
            <crs:edition>5.2</crs:edition>
          </crs:Identifier>
        </crs:CoordinateReferenceSystem>
      </rdf:Description>
    </rdf:RDF>
  </metadata>

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

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

Example: 07_21.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="100" height="100" viewBox="0 0 1000 1000">

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

```
<metadata>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:crs="http://www.ogc.org/crs"
           xmlns:svg="http://wwww.w3.org/2000/svg">
    <rdf:Description rdf:about="">
      <!-- For other CRS it should be entirely defined -->
      <crs:CoordinateReferenceSystem svg:transform="scale(1,-1)">
        <crs:NameSet>
          <crs:name>Mercator projection of WGS84</crs:name>
        </crs:NameSet>
        <crs:ProjectedCRS>
          <!-- The actual definition of the CRS -->
          <crs:CartesianCoordinateSystem>
            <crs:dimension>2</crs:dimension>
            <crs:CoordinateAxis>
              <crs:axisDirection>north</crs:axisDirection>
              <crs:AngularUnit>
                <crs:Identifier>
                  <crs:code>9108</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
              </crs:AngularUnit>
            </crs:CoordinateAxis>
            <crs:CoordinateAxis>
              <crs:axisDirection>east</crs:axisDirection>
              <crs:AngularUnit>
                <crs:Identifier>
                  <crs:code>9108</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
              </crs:AngularUnit>
            </crs:CoordinateAxis>
          </crs:CartesianCoordinateSystem>
          <crs:CoordinateReferenceSystem>
            <!-- the reference system of that projected system is
                       WGS84 which is EPSG 4326 in EPSG codeSpace -->
            <crs:NameSet>
              <crs:name>WGS 84</crs:name>
            </crs:NameSet>
            <crs:Identifier>
              <crs:code>4326</crs:code>
              <crs:codeSpace>EPSG</crs:codeSpace>
              <crs:edition>5.2</crs:edition>
            </crs:Identifier>
          </crs:CoordinateReferenceSystem>
          <crs:CoordinateTransformationDefinition>
            <crs:sourceDimensions>2</crs:sourceDimensions>
            <crs:targetDimensions>2</crs:targetDimensions>
            <crs:ParameterizedTransformation>
              <crs:TransformationMethod>
                <!-- the projection is a Mercator projection which is
                      EPSG 9805 in EPSG codeSpace -->
                <crs:NameSet>
                  <crs:name>Mercator</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>9805</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:description>Mercator (2SP)</crs:description>
              </crs:TransformationMethod>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>Latitude of 1st standart parallel</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8823</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>Longitude of natural origin</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8802</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>False Easting</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8806</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>False Northing</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8807</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
            </crs:ParameterizedTransformation>
          </crs:CoordinateTransformationDefinition>
        </crs:ProjectedCRS>
      </crs:CoordinateReferenceSystem>
    </rdf:Description>
  </rdf:RDF>
</metadata>

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

8 Paths

Contents

8.1 Introduction 87
8.2 The 'path' element 87
  8.2.1 Animating path data 88

8.3 Path data 88
  8.3.1 General information about path data 88
  8.3.2 The "moveto" commands 89
  8.3.3 The "closepath" command 89
  8.3.4 The "lineto" commands 90
  8.3.5 The Curve commands 90
  8.3.6 The Cubic Bézier curve commands 90
  8.3.7 The Quadratic Bézier curve commands 92
  8.3.8 The grammar for path data 93

8.4 Distance along a path 95

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. The lacuna value is the empty string.
    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 is an unsupported value.

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 on the given animation element.

Non-interpolated (i.e. **calcMode="discrete"**) path data animation is always possible.

8.3 Path data

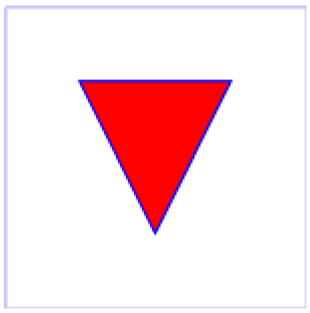
8.3.1 General information about path data

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 U+002E FULL STOP (".") character (also referred to in Unicode as PERIOD, dot and decimal point) [UNICODE] and no other delimiter characters are allowed. (For example, the following is an invalid numeric value in path data: "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 treated as a pair of absolute coordinates. If a 'moveto' is followed by multiple pairs of co-ordinates, 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 sub-path 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. Since the Z and z commands take no parameters, they have an identical effect.

8.3.4 The **"lineto"** commands

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

Command	**Name**	**Parameters**	**Description**
L (absolute) **l** (relative)	lineto	(x y)+	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 co-ordinate 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 horizonal 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

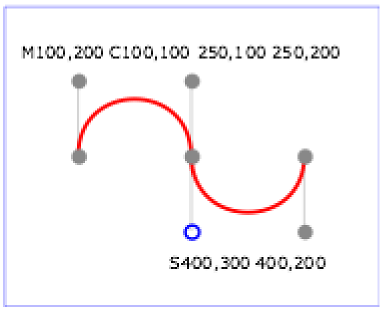
The 'Cubic Bézier' commands are as follows:

Command	**Name**	**Parameters**	**Description**
C (absolute) **c** (relative)	curveto	(x1 y1 x2 y2 x y)+	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.
S (absolute) **s** (relative)	shorthand/ smooth curveto	(x2 y2 x y)+	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 new current point shall be the final (x,y) coordinate pair used in the polybézier.

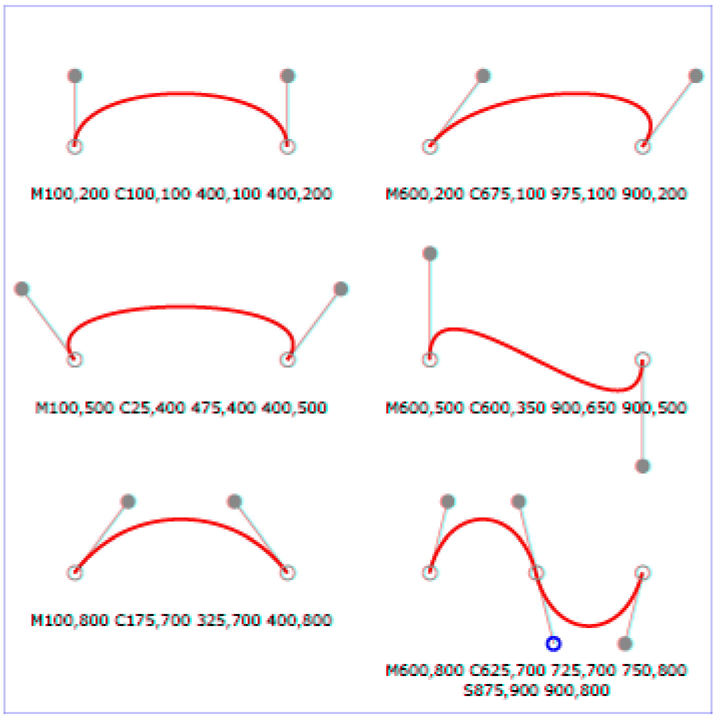
Example 08_02 shows some simple uses of 'Cubic Bézier' commands within a **'path'**. Note that the control point for the "S" command is computed automatically as the reflection of the control point for the previous "C" command relative to the start point of the "S" command.

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.


M100,200 C100,100 400,100 400,200
M600,200 C675,100 975,100 900,200
M100,500 C25,400 475,400 400,500
M600,500 C600,350 900,650 900,500
M100,800 C175,700 325,700 400,800
M600,800 C625,700 725,700 750,800
S875,900 900,800

8.3.7 The **Quadratic Bézier** curve commands

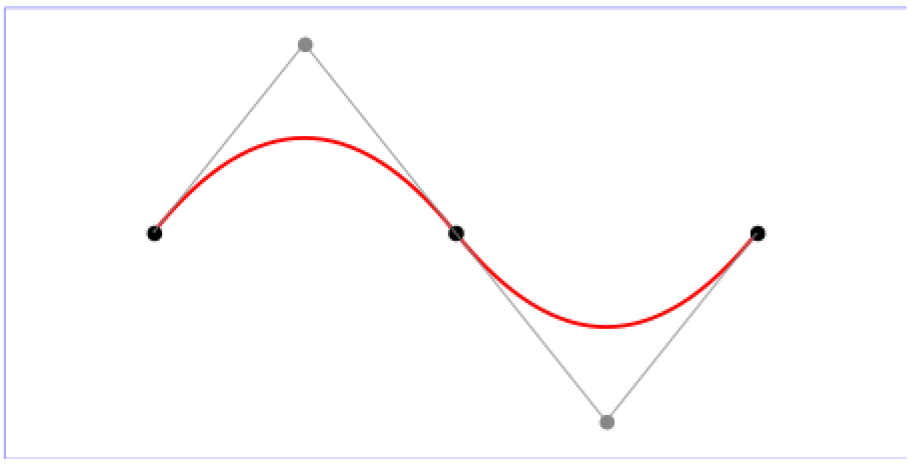
The 'Quadratic Bézier' commands are as follows:

Command	**Name**	**Parameters**	**Description**
Q (absolute) **q** (relative)	quadratic Bézier curveto	(x1 y1 x y)+	A quadratic Bézier curve is drawn from the current point to (x,y) using (x1,y1) 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.
T (absolute) **t** (relative)	Shorthand/ smooth quadratic Bézier curveto	(x y)+	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.

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]:

```
path-data ::=
    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-curveto-argument-sequence ::=
    smooth-curveto-argument
    | smooth-curveto-argument comma-wsp? smooth-curveto-argument-sequence
smooth-curveto-argument ::=
    coordinate-pair comma-wsp? coordinate-pair
quadratic-bezier-curveto ::=
    ( "Q" | "q" ) wsp* quadratic-bezier-curveto-argument-sequence
quadratic-bezier-curveto-argument-sequence ::=
    quadratic-bezier-curveto-argument
    | quadratic-bezier-curveto-argument comma-wsp?
        quadratic-bezier-curveto-argument-sequence
quadratic-bezier-curveto-argument ::=
    coordinate-pair comma-wsp? coordinate-pair
smooth-quadratic-bezier-curveto ::=
    ( "T" | "t" ) wsp* smooth-quadratic-bezier-curveto-argument-sequence
smooth-quadratic-bezier-curveto-argument-sequence ::=
    coordinate-pair
    | coordinate-pair comma-wsp? smooth-quadratic-bezier-curveto-argument-sequence
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 some stroke operations, require that the user agent compute the distance along the geometry of a graphics element, such as a **'path'**.

To aid hand authoring by allowing convenient round numbers to be used, 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 96
9.2 The 'rect' element 96
9.3 The 'circle' element 98
9.4 The 'ellipse' element 100
9.5 The 'line' element 101
9.6 The 'polyline' element 102
9.7 The 'polygon' element 104
 9.7.1 The grammar for points specifications in 'polyline' and 'polygon' elements 105

9.1 Introduction

SVG contains the following set of basic shape elements:

- rectangles (including optional rounded corners), created with the **'rect'** element,
- circles, created with the **'circle'** element,
- ellipses, created with the **'ellipse'** element,
- straight lines, created with the **'line'** element,
- polylines, created with the **'polyline'** element, and
- polygons, created with the **'polygon'** element.

Mathematically, these shape elements are equivalent to a **'path'** element that would construct the same shape. The basic shapes may be stroked, 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. The lacuna value is **'0'**.

Animatable: yes.

y = **"<coordinate>"**

The y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value in the current user coordinate system. The lacuna value is **'0'**.

Animatable: yes.

`width` = **"<length>"**
The width of the rectangle. A negative value is unsupported. A value of zero disables rendering of the element. The lacuna value is **'0'**.
Animatable: yes.

`height` = **"<length>"**
The height of the rectangle. A negative value is unsupported. A value of zero disables rendering of the element. The lacuna value is **'0'**.
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.

A **'rect'** element, taking its rounded corners into account, must be rendered in a way that produces the same result as if the following outline were specified instead (note: all coordinate and length values are first converted into user space coordinates according to Units):

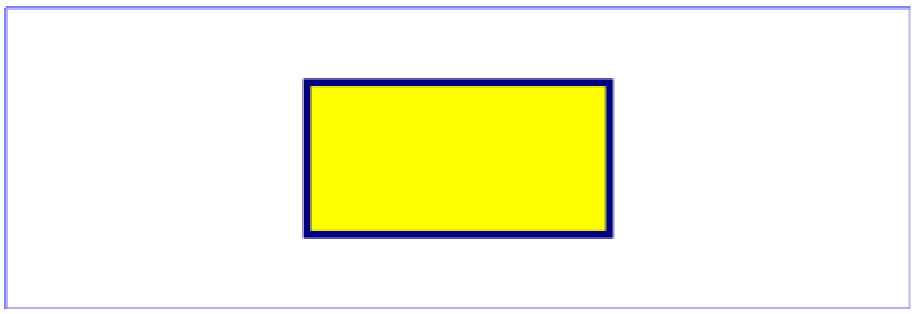
1. Perform an absolute moveto operation to location (*x*+*rx*,*y*), where *x* and *y* are the values of the **'rect'** element's **'x'** and **'y'** attribute converted to user space, and *rx* and *ry* are the effective values of the **'rx'** and **'ry'** attributes converted to user space.
2. Perform an absolute horizontal lineto operation to location (*x*+*width*-*rx*,*y*), where *width* is the **'rect'** element's **'width'** attribute converted to user space.
3. 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 semimajor and semiminor axis, respectively, zero x-axis-rotation, a clockwise sweep direction and choosing the smaller arc sweep.
4. Perform an absolute vertical *lineto* operation to location (*x*+*width*,*y*+*height*-*ry*), where *height* is the **'rect'** element's **'height'** attribute converted to user space.
5. Perform an absolute elliptical arc operation to coordinate (*x*+*width*-*rx*,*y*+*height*).
6. Perform an absolute horizontal *lineto* operation to location (*x*+*rx*,*y*+*height*).
7. Perform an absolute elliptical arc operation to coordinate (*x*,*y*+*height*-*ry*).
8. Perform an absolute vertical *lineto* operation to location (*x*,*y*+*ry*).
9. Perform an absolute elliptical arc operation to coordinate (*x*+*rx*,*y*).
10. Perform a closepath (z) to the coordinate specified in the original moveto operation.

In case the **'rx'** and **'ry'** attributes are not specified or set to a value of zero, the elliptical arc commands should be omitted.

Example 09_01 shows a rectangle with sharp corners. The **'rect'** element is filled with yellow and stroked with navy.

Example: 09_01.svg

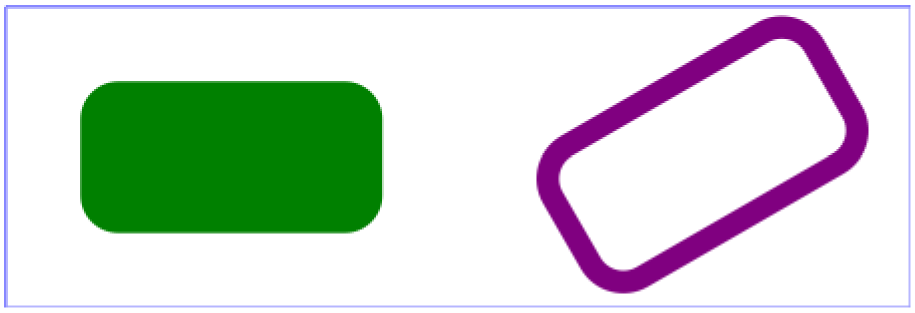
```
<?xml version="1.0"?>
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny">
  <desc>Example rect01 - rectangle with sharp corners</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" 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.

Within the current user coordinate system, stroking operations on a circle begin at the point (cx+r,cy) and then proceed through the points (cx,cy+r), (cx-r,cy), (cx,cy-r) and finally back to (cx+r,cy). For stroking operations, there is only one line segment which has its beginning joined to its end.

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. The lacuna value is **'0'**.
Animatable: yes.

`cy` = **"<coordinate>"**
The y-axis coordinate of the center of the circle. The lacuna value is **'0'**.
Animatable: yes.

`r` = **"<length>"**
The radius of the circle. A negative value is unsupported. A value of zero disables rendering of the element. The lacuna value is **'0'**.
Animatable: yes.

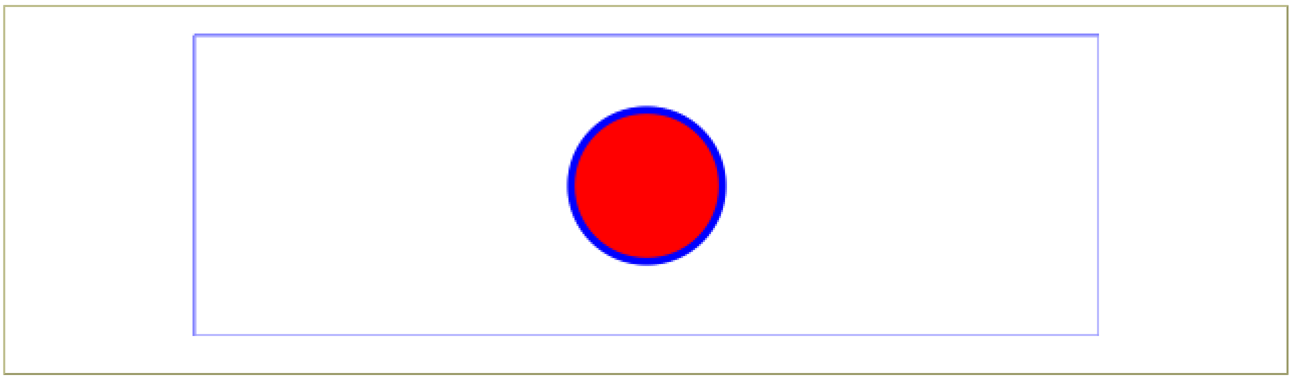
`focusable` = **"true" | "false" | "auto"**
See attribute definition for description.
Animatable: yes.

`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.

Within the current user coordinate system, stroking operations on a ellipse begin at the point (cx+rx,cy) and then proceed through the points (cx,cy+ry), (cx-rx,cy), (cx,cy-ry) and finally back to (cx+rx,cy). For stroking operations, there is only one line segment which has its beginning joined to its end.

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. The lacuna value is **'0'**.
Animatable: yes.

`cy` = **"<coordinate>"**
The y-axis coordinate of the center of the ellipse. The lacuna value is **'0'**.
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. The lacuna value is **'0'**.
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. The lacuna value is **'0'**.
Animatable: yes.

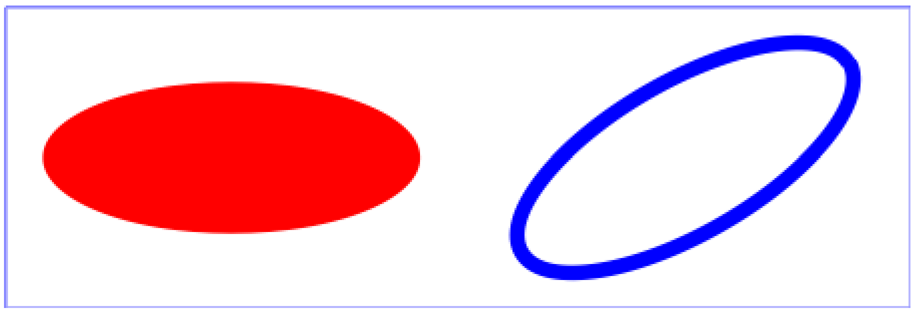
`focusable` = **"true" | "false" | "auto"**
See attribute definition for description.
Animatable: yes.

`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 lacuna value 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.X12Y12.attr'/>
    </define>
```

Attribute definitions:

x1 = **"<coordinate>"**
The x-axis coordinate of the start of the line. The lacuna value is **'0'**.
Animatable: yes.

y1 = **"<coordinate>"**
The y-axis coordinate of the start of the line. The lacuna value is **'0'**.
Animatable: yes.

x2 = **"<coordinate>"**
The x-axis coordinate of the end of the line. The lacuna value is **'0'**.
Animatable: yes.

`y2` = **"<coordinate>"**

The y-axis coordinate of the end of the line. The lacuna value is **'0'**.

Animatable: yes.

`focusable` = **"true" | "false" | "auto"**

See attribute definition for description.

Animatable: yes.

`Navigation Attributes`

See definition.

A **'line'** element must be rendered in a way that produces the same result as if the following path were specified instead (note: all coordinate and length values are first converted into user space coordinates according to Units):

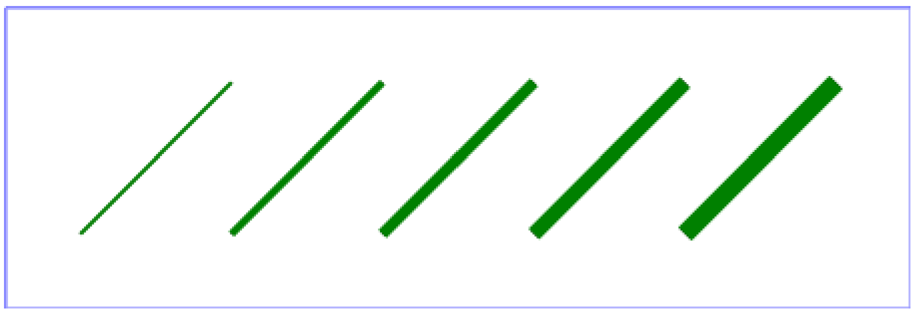
1. Perform an absolute moveto operation to absolute location (*x1*,*y1*), where *x1* and *y1* are the values of the **'line'** element's **'x1'** and **'y1'** attributes converted to user space, respectively.
2. Perform an absolute lineto operation to absolute location (*x2*,*y2*), where *x2* and *y2* 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'/>
  <optional>
    <attribute name='points' svg:animatable='true' svg:inheritable='false'>
      <ref name='Points.datatype'/>
    </attribute>
  </optional>
</define>
```

Attribute definitions:

`points` = **"<points-data>"**

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. The lacuna value is the empty string.

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.

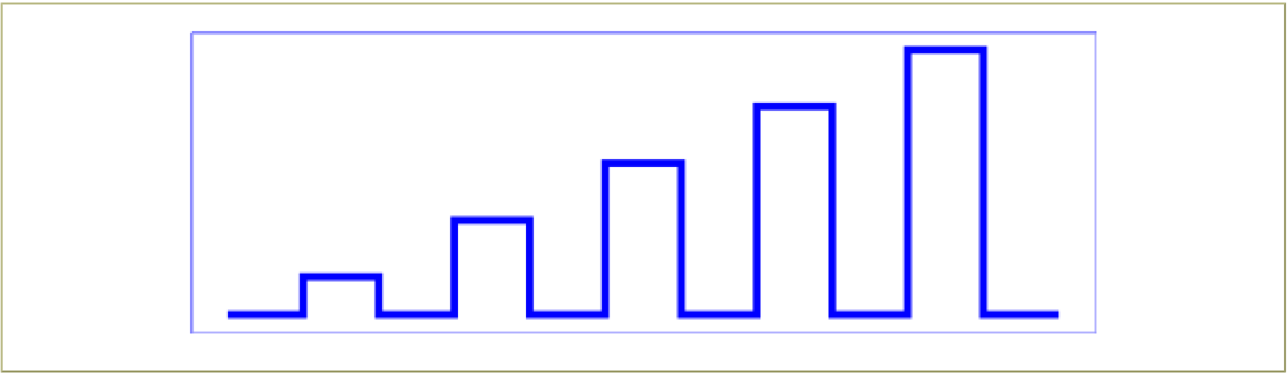
A **'polyline'** element must be rendered in a way that produces the same result as if the following path were specified instead:

1. Perform an absolute moveto operation to the first coordinate pair in the list of points.
2. 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` = **"<points-data>"**

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. The lacuna value is the empty string.

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 in the **'points'** attribute, then it is treated as an unsupported value.

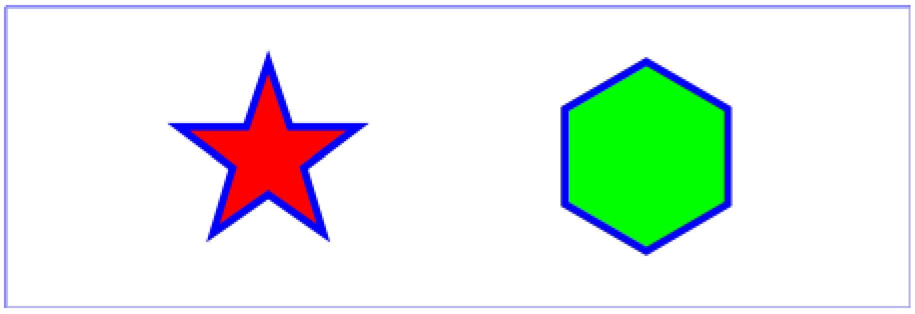
A **'polygon'** element must be rendered in a way that produces the same result as if the following path were specified instead:

1. Perform an absolute moveto operation to the first coordinate pair in the list of points.
2. For each subsequent coordinate pair, perform an absolute lineto operation to that coordinate pair.
3. 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 an EBNF grammar for **<points-data>** values on **'polyline'** and **'polygon'** elements [EBNF]:

```
points-data:
    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)+
```

10 Text

Contents

10.1 Introduction 106
10.2 Characters and their corresponding glyphs 107
10.3 Fonts, font tables and baselines 108
10.4 The 'text' element 109
10.5 The 'tspan' element 111
10.6 Text layout 112
    10.6.1 Text layout introduction 112
    10.6.2 Relationship with bidirectionality 113
    10.6.3 The 'direction' property 115
    10.6.4 The 'unicode-bidi' property 115

10.7 Text rendering order 115
10.8 Alignment properties 115
    10.8.1 Text alignment properties 115

10.9 Font selection properties 116
10.10 White space handling 118
10.11 Text in an area 119
    10.11.1 Introduction to text in an area 119
    10.11.2 The 'textArea' element 120
    10.11.3 The 'tbreak' element 121
    10.11.4 The 'line-increment' property 122
    10.11.5 The 'text-align' property 122
    10.11.6 The 'display-align' property 123
    10.11.7 Text in an area layout rules 123

10.12 Editable text fields 124
    10.12.1 The 'editable' attribute 124

10.13 Text selection and clipboard operations 126
10.14 Text search 127

10.1 Introduction

Text that is to be rendered as part of an SVG document fragment is specified using the text content block elements. The characters to be drawn are expressed as XML content inside the element.

SVG's text content block 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:

- Use the **'textArea'** element to specify a rectangular area in which to flow the text.
- Pre-compute the line breaks (which can be done by the author or authoring tool) and use individual **'text'** elements to manually place the lines of text. (Note: this is discouraged for accessibility reasons.)
- 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 shall be rendered in one straight line. 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 content:

- 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 search and Text selection and clipboard operations)
- XML-compatible Web search engines will find text strings in SVG content with no additional effort over what they need to do to find text strings in other XML documents

Multi-language SVG content is possible by substituting different text strings based on the user's preferred language.

For accessibility reasons, it is recommended that text which is included in a document have appropriate semantic markup to indicate its function. See SVG accessibility guidelines for more information.

10.2 Characters and their corresponding glyphs

In XML [XML10, 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 many languages, ligatures are required for certain character combinations, and are not optional typographic features. For example, this is the case for most languages throughout South and South East Asia.
- Composite characters — In many 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.
- Context-sensitive glyph positioning — In many scripts, the precise positioning of the glyph for a given character (especially diacritics) will vary according to the visual context. For example, Thai tone marks are rendered above the base consonant, but need to be moved upwards further if a vowel-sign also appears above the base consonant. The same character is used in memory, but the final location of the glyph is sensitive to context.
- Complex positioning of character glyphs — In scripts such as those used for Indian languages, a combining vowel character that appears after a base consonant in memory may be displayed to the left of the base consonant, or on two sides of the base consonant, (i.e., the left-most glyph in rendered text may not be the first character in a text element.) Indeed, such vowel characters may be rendered to the left of, or on more than one side of, a *cluster* of consonants that ends with the character they follow in memory. On the other hand, a Hindi *RA* character at the beginning of a consonant cluster in memory may be displayed over a following vowel sign to the right of the following syllabic cluster. The location, from left to right, in which glyphs are displayed in these scripts can differ significantly from the order of the characters in memory.
- Glyph substitution — Many 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 many 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 ([CSS2], section 15.4.3).) The box 1 EM high and 1 EM wide is called the *design space*. This space is given geometric coordinates by sub-dividing the EM into a number of units-per-em ([CSS2], section 15.3.4).

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, 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 content 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, in 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 glyph. In the simplest case, the new current text position is the previous current text position plus the glyph's 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>
```

Attribute definitions:

x = **"<list-of-coordinates>"**

If a single <coordinate> is provided, then the value represents the new absolute X coordinate for the current text position for rendering the glyphs that correspond to the first character within this element or any of its descendants.

If a comma- or space-separated list of *n* <coordinate>s is provided, then the values represent new absolute X coordinates for the current text position for rendering the glyphs corresponding to each of the first *n* characters within this element or any of its descendants.

If more <coordinate>s are provided than there are characters, then the extra <coordinate>s must be ignored.

If more characters are provided than <coordinate>s, then for each of these extra characters normal text layout processing describied in the Text layout section shall occur.

At least one <coordinate> value must be specified in the attribute.

The lacuna value is **'0'**.

Animatable: yes.

y = **"<list-of-coordinates>"**

The corresponding list of absolute Y coordinates for the glyphs corresponding to the characters within this element. The processing rules for the **'y'** attribute parallel the processing rules for the **'x'** attribute.

At least one <coordinate> value must be specified in the attribute.

The lacuna value is **'0'**.

Animatable: yes.

editable = **"none" | simple"**

This attribute indicates whether the text can be edited. See the definition of the **'editable'** attribute.

Animatable: yes.

rotate = **"<list-of-numbers>"**

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.

At least one <number> value must be specified in the attribute.

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.

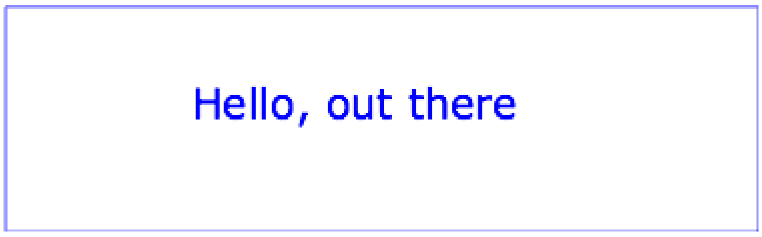
Navigation Attributes

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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="10cm" height="3cm" viewBox="0 0 1000 300">

  <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>
```



10.5 The 'tspan' element

Within a text content block element, graphic and font properties can be adjusted by including a **'tspan'** element.

Schema: tspan

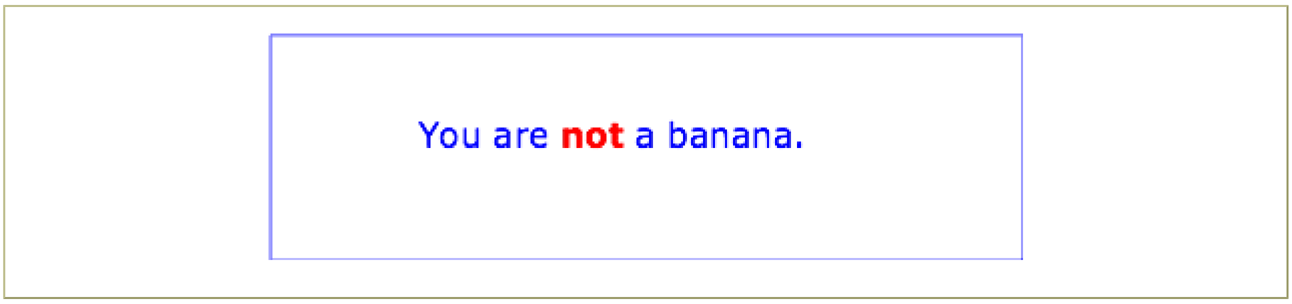
```
<define name='tspan'>
  <element name='tspan'>
    <ref name='tspan.AT'/>
    <zeroOrMore><ref name='svg.TextCommon.group'/></zeroOrMore>
  </element>
</define>

<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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="10cm" height="3cm" viewBox="0 0 1000 300">

  <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>
```



Within a text content block element, graphic and font properties can be adjusted by including a **'tspan'** element. Positional attributes such as **'x'**, **'y'**, 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 content block 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**.

In processing a given text content block 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 content block 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 content block 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] section 15.3.8 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 the discussion of the CSS2 spacing properties ([CSS2], section 16.4), 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 da<?turn around?>hut 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.

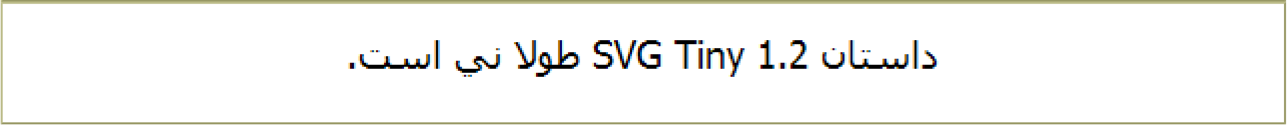
The Unicode standard ([UNICODE], specifically [UAX9]) defines a complex algorithm for determining the proper directionality of text. The algorithm consists of an implicit part based on character properties, as well as explicit controls for embeddings and overrides. The SVG user agent applies this bidirectional algorithm when determining the layout of characters within a text content block element.

The **'direction'** and **'unicode-bidi'** properties allow authors to override the inherent directionality of the content characters and thus explicitly control how the elements and attributes of a document language map to this algorithm. These two properties are applicable to all characters whose glyphs are perpendicular to the inline-progression-direction.

In many cases, the bidirectional algorithm from Unicode [UNICODE] produces the desired result automatically, and in such cases the author does not need to use these properties. For other cases, such as when using right-to-left languages, it may be sufficient to add the **'direction'** property to the rootmost 'svg' element, and allow that direction to inherit to all text elements, as in the following example (which may be used as a template):

Example: rtl-text.svg

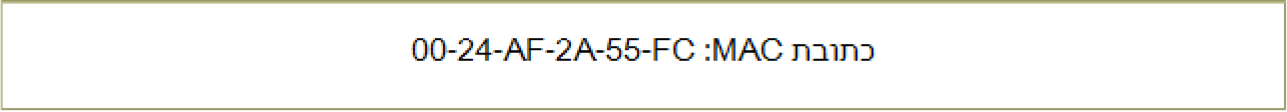
```
<svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 400 400"
     direction="rtl" xml:lang="fa">

  <title direction="ltr" xml:lang="en">Right-to-left Text</title>
  <desc direction="ltr" xml:lang="en">
    A simple example for using the 'direction' property in documents
    that predominantly use right-to-left languages.
  </desc>

  <text x="200" y="200" font-size="20">داستان SVG Tiny 1.2 طولا ني است.</text>

</svg>
```


داستان SVG Tiny 1.2 طولا ني است.

Below is another example, where where implicit bidi reordering is not sufficient:

Example: rtl-complex.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 400 400"
     direction="rtl" xml:lang="he">

  <title direction="ltr" xml:lang="en">Right-to-left Text</title>
  <desc direction="ltr" xml:lang="en">
    An example for using the 'direction' and 'unicode-bidi' properties
    in documents that predominantly use right-to-left languages.
  </desc>

  <text x="200" y="200" font-size="20"> כתובת
        MAC:&#x200F;
            <tspan direction="ltr" unicode-bidi="embed">00-24-AF-2A-55-FC</tspan>
        </text>

</svg>
```


כתובת MAC: 00-24-AF-2A-55-FC

Within text content elements, the alignment of text with regards to the **'text-anchor'** or **'text-align'** properties is determined by the value of the **'direction'** property. For example, given a **'text'** element with a **'text-anchor'** value of **"end"**, for a **'direction'** value of **"ltr"**, the text will extend to the left of the position of the **'text'** element's **'x'** attribute value, while for **'direction'** value of **"rtl"**, the text will extend to the right of the position of the **'text'** element's **'x'** attribute value.

A more complete discussion of bidirectionality can be found in the Text direction section of CSS 2 ([CSS2], section 9.10).

The processing model for bidirectional text is as follows. The user agent processes the characters which are provided in logical order (i.e., the order the characters appear in the original document). 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. After processing the Unicode bidirectional algorithm and properties **'direction'** and **'unicode-bidi'** on each of the independent text blocks, the user agent will have a potentially re-ordered list of characters which are now in left-to-right rendering order. 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.

10.6.3 The **'direction'** property

'direction'

Value:	ltr | rtl | inherit
Initial:	ltr
Applies to:	text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

This property specifies the base writing direction of text and the direction of embeddings and overrides (see **'unicode-bidi'**) for the Unicode bidirectional algorithm. For the **'direction'** property to have any effect on an element that does not by itself establish a new text chunk (such as the **'tspan'** element in SVG 1.2 Tiny), the **'unicode-bidi'** property's value must be **embed** or **bidi-override**.

Note: though this property can be declared on any element for purposes of inheritance, it only affects text content. It does not effect the coordinate system or the positioning of shapes.

Except for any additional information provided in this specification, the normative definition of the property is found in CSS 2 ([CSS2], section 9.10).

10.6.4 The **'unicode-bidi'** property

'unicode-bidi'

Value:	normal | embed | bidi-override | inherit
Initial:	normal
Applies to:	text content elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	no

Except for any additional information provided in this specification, the normative definition of the property is found in CSS 2 ([CSS2], section 9.10).

10.7 Text rendering order

The glyphs associated with the characters within text content block elements are rendered in the logical order of the characters in the original document, independent of any re-ordering necessary for visual display (e.g 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, as they come earlier in logical order.

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 a string of text relative to a given point, along a given axis. This axis of alignment varies by writing mode; for horizontal writing mode (the norm for Latin or Arabic) the axis is horizontal, while for vertical writing mode (often used for Japanese) the axis is vertical. *(Note: SVG Tiny 1.2 does not include support for vertical text.)* The point of orientation depends upon the **'text-anchor'** property value.

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
Animatable:	yes
Computed value:	Specified value, except inherit

Values have the following meanings (where *right*, *left*, *top*, and *bottom* refer to alignment in an untransformed coordinate space):

start
: The rendered characters are aligned such that the start of the resulting rendered text is at the initial current text position. For an element with a **'direction'** property value of **"ltr"** (typical for most European languages), the left side of the text is rendered at the initial text position. For an element with a **'direction'** property value of **"rtl"** (typical for Arabic and Hebrew), the right side of the text is rendered at the initial text position. For an element with a vertical primary text direction (often typical for Asian text), the top side of the text is rendered at the initial text position. *(Note: SVG Tiny 1.2 does not include support for vertical text.)*

middle
: The rendered characters are aligned such that the geometric middle of the resulting rendered text is at the initial current text position.

end
: The rendered characters are aligned such that the end of the resulting rendered text is at the initial current text position. For an element with a **'direction'** property value of **"ltr"** (typical for most European languages), the right side of the text is rendered at the initial text position. For an element with a **'direction'** property value of **"rtl"** (typical for Arabic and Hebrew), the left side of the text is rendered at the initial text position. For an element with a vertical primary text direction (often typical for Asian text), the bottom of the text is rendered at the initial text position. *(Note: SVG Tiny 1.2 does not include support for vertical text.)*

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 XSL 1.1 ([XSL], section 7.9).

'font-family'

Value:	[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>] | inherit
Initial:	depends on user agent
Applies to:	text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	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 XSL 1.1 ([XSL], section 7.9.2).

'font-style'

Value:	normal | italic | oblique | inherit
Initial:	normal

Applies to:	text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified value, except inherit

This property specifies whether the text is to be rendered using a normal, italic or oblique face. The font-style value "backslant" defined in XSL 1.1 is not supported. Except for any additional information provided in this specification, the normative definition of the property is in XSL 1.1 ([XSL], section 7.9.7).

'font-variant'

Value:	normal | small-caps | inherit
Initial:	normal
Applies to:	text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	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 XSL 1.1 ([XSL], section 7.9.8).

'font-weight'

Value:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit
Initial:	normal
Applies to:	text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	one of the legal numeric values, non-numeric values shall be converted to numeric values according to the rules defined below.

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 XSL 1.1 ([XSL], section 7.9.9).

Non-numeric values are interpreted as follows:

normal
Same as "400".

bold
Same as "700".

bolder
Specifies the next weight that is assigned to a font that is darker than the inherited one. If there is no such weight, it simply results in the next darker numerical value (and the font remains unchanged), unless the inherited value was "900", in which case the resulting weight is also "900".

lighter

Specifies the next weight that is assigned to a font that is lighter than the inherited one. If there is no such weight, it simply results in the next lighter numerical value (and the font remains unchanged), unless the inherited value was "100", in which case the resulting weight is also "100".

'font-size'

Value:	<absolute-size> | <relative-size> | <length> | inherit
Initial:	medium
Applies to:	text content elements
Inherited:	yes, the computed value is inherited
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	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. Percentage values are not supported.

Except for any additional information provided in this specification, the normative definition of the property is in XSL 1.1 ([XSL], section 7.9.4).

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 content block element's character data. Note that any child element of a text content block element may also have an **'xml:space'** attribute which will apply to that child element's text content. The SVG user agent has special processing rules associated with this attribute as described below. These are behaviors that occur subsequent to XML parsing [XML11] and do not affect the contents of the Document Object Model (DOM) [DOM3].

Attribute definition:

`xml:space` **= "default" | "preserve"**

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").

Animatable: no.

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).

```
<text xml:space='default'>
  WS example
  indented lines
</text>
```

```
[05]  <text xml:space='preserve'>WS example indented lines</text>
[06]
[07]  <text xml:space='default'>
[08]WS example
[09]non-indented lines
[10]  </text>
[11]  <text xml:space='preserve'>WS examplenon-indented lines</text>
```

The first pair of **'text'** elements above show the effect of indented character data. The attribute **xml:space='default'** in the first **'text'** element instructs the SVG 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 SVG 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 ([XML11], section 2.11).

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.

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.

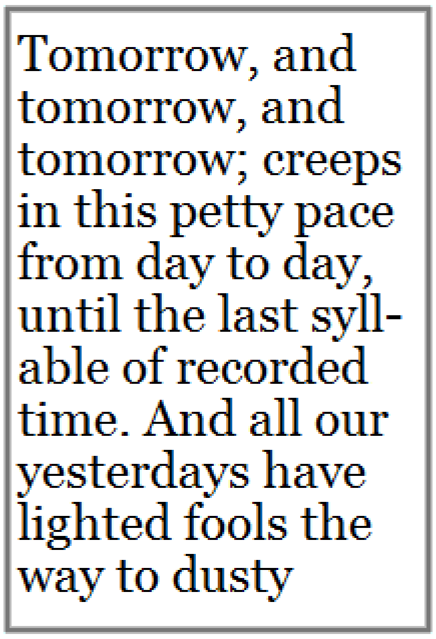
The layout of wrapped text is user agent dependent; thus, content developers need to be aware that there might be different results, particularly with regard to where line breaks occur.

The 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 xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny" viewBox="0 0 220 320">

  <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" fill="none"/>
</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>
      <choice>
        <element name='tspan'>
          <ref name='tspan.AT'/>
          <zeroOrMore>
            <choice>
              <ref name='tbreak'/>
              <ref name='svg.TextCommon.group'/>
            </choice>
          </zeroOrMore>
        </element>
        <ref name='svg.TextCommon.group'/>
      </choice>
    </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. The lacuna value is **'0'**.
Animatable: yes.

y = **"<coordinate>"**
The y-axis coordinate of one corner of the rectangular region into which the text content will be placed. The lacuna value is **'0'**.
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. The lacuna value is **'auto'**.
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. The lacuna value is **'auto'**.
Animatable: yes.

editable = **"none" | "simple"**
This attribute indicates whether the text can be edited. See the definition of the **'editable'** attribute.
Animatable: yes.

focusable = **"true" | "false" | "auto"**
See attribute definition for description.
Animatable: yes

Navigation Attributes
See definition.

If both **'width'** and **'height'** have the value **'auto'**, the text will be rendered in a single line along the direction of the text progression until all the text is rendered, or until a line-breaking element such as **'tbreak'** is encountered, in which case the remaining text is rendered on a new line.

10.11.3 The **'tbreak'** element

The **'tbreak'** element is an empty element that forcibly breaks the current line of text, even if the current line of text is empty (i.e. multiple consecutive **'tbreak'** elements each cause a line break.)

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

The **'tbreak'** element has no attributes aside from the standard core and conditional 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, and to child elements of the **'textArea'** element. The **'line-increment'** property must not have any effect when used on an element which is not, or does not have as an ancestor, a **'textArea'** element.

'line-increment'

Value:	auto | <number> | inherit
Initial:	auto
Applies to:	**'textArea'**, **'tspan'** and **'tbreak'** elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	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, multiplied by some reasonable value for default line spacing. This specification recommends a value of 1.1 for this multiplier.

<number>
Subsequent lines are offset from the previous line by this amount (in user units). Negative values are unsupported.

10.11.5 The **'text-align'** property

Alignment in the inline progression direction in flowing text is provided by the text-align property. It is a modified version of the XSL 1.1 text-align property ([XSL], section 7.16.9).

'text-align'

Value:	start | end | center | inherit
Initial:	start
Applies to:	textArea elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

For details refer to the XSL 1.1 text-align property ([XSL], section 7.16.9). Note that SVG does not require user agents to support the following values for this property: **justify**, **inside**, **outside**, **<string>**, **left**, or **right**. The lacuna value is **start**.

As with the **'text-anchor'** property, the values **start** and **end** are dependent on the value of the **'direction'** property (typically, as appropriate for the writing system being used).

- For left to right horizontal (French, Russian, Thai, etc.): **start** is left and **end** is right
- For right to left horizontal (Hebrew, Arabic, etc.): **start** is right and **end** is left
- For top to bottom vertical (vertical Chinese, etc.): **start** is top and **end** is bottom *(Note: SVG Tiny 1.2 does not include support for vertical text.)*

10.11.6 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'

Value:	auto | before | center | after | inherit
Initial:	auto
Applies to:	**'textArea'**
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	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.

The following sentence is informative: For a better understanding of the **'display-align'** property this diagram from the XSL specification (the final diagram in [XSL], section 4.2.3) illustrates the correspondence between the various edge names for a mixed writing-mode example (western and japanese writing-mode).

10.11.7 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 SVG 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.

SVG 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 [UAX14].
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 BIDI 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: its normal extent, and its last in text area extent. Its 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 hyphen, but does not include the advance of trailing white space or non-spacing 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 SVG user agent may choose to force a break in the line so that at least some text will appear for the given line.)
7. If **xml:space="default"**, any space character that causes a line break in a **'textArea'** element will be consumed by the line break and thus not rendered. However, if **xml:space="preserve"**, any space character that causes a line break and subsequent spaces that will not fit on the line shall be included in the next line.
8. 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'**.
9. 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.
10. 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).
11. 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. Go to step 8.
12. 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 content block 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' svg:animatable='true' svg:inheritable='false'>
      <choice>
        <value>none</value>
        <value>simple</value>
      </choice>
    </attribute>
  </optional>
</define>
```

Attribute definition:

`editable` = **"none" | "simple"**

If set to **'none'** (the lacuna value), SVG editing facilities are not provided for the contents of the text content block elements. If set to **'simple'**, the SVG user agent must provide a way for the user to edit the content of the text content block elements and all contained subelements which are not hidden (with **visibility="hidden"**) or removed from the rendering tree (through the **'switch'** element or **display="none"**).

If a clipboard is supported by the platform, the SVG user agent must also 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.

Whenever the **'editable'** attribute is set to **'simple'**, the **'focusable'** attribute is considered to be set to **'true'**, irrespective of what the actual value is.

Animatable: yes.

SVG user agents should allow for the editing of text in-place. 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 presentation value of the **'editable'** attribute must be **'simple'**, the text content block element must have focus, and it must then be activated, e.g. by using an Enter key or clicking on the text region with a pointer device. When editing text in a text field, all text and key events are dispatched to the SVG user agent, which processes the events for proper handling of text entry.

If a text content block element is editable, then the SVG user agent must not normalize white space in user input when changing the tree according to the input. However, the displayed text must be rendered according to the SVG rules for **'xml:space'**.

For editing in-place 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 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 is 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 an alike behavior native to the platform), firing a single `textInput` 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 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 SVG 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 SVG user agent.

The behavior of edited text while the caret is placed inside a ligature is implementation dependent. SVG 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 [CHARMOD].

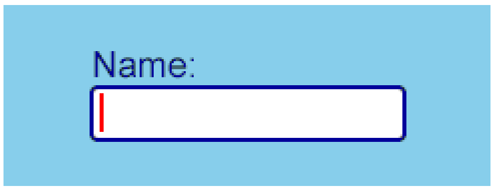
If the text of an editable element is edited, and the element has child elements, the contents of the edited element must first be stripped of all non-**'tbreak'** elements, preserving the contents of each non-**'tbreak'** element in place.

If the editable element does not have text content, it may not be possible to activate the editability with a pointer, since there will be no rendered element to click on. In the case of the **'textArea'** element, which has inherent **'width'** and **'height'** geometry, setting the **'pointer-events'** property value to **boundingBox** will allow the user to initiate the editing (see Example textArea02). This functionality does not exist for the **'text'** element since it has no inherent geometry without text content.

Example textArea02 below shows how to use the **'pointer-events'** property value **boundingBox** to create a declarative input box that can be activated with a pointer device.

Example: textArea02.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.2" baseProfile="tiny" viewBox="0 0 250 100">

  <title>Editable text input</title>
  <desc>Illustrates how to create an editable input box without script</desc>

  <rect x='0' y='0' width='250' height='100' fill='#87ceeb'>
    <title>background rectangle</title>
  </rect>

  <g id="nameInput" transform='translate(50, 20)'>
    <text x="0" y="20" font-size="18" font-family="Arial" fill="#000080">Name:</text>
    <rect x="0" y="25" width="156" height="26" rx="3" ry="3"
          fill="white" stroke-width="2" stroke="#000080"/>
    <textArea x="3" y="27" width="150" height="20" font-size="18" font-family="Arial"
              editable="simple" focusable="true" pointer-events="boundingBox"/>
  </g>

</svg>
```



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 content block 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 should 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 SVG user agent must support text selection in logical order, which will result in discontinuous highlighting of glyphs due to the bidirectional reordering of characters. SVG 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 SVG 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.

In addition to discrete text selection, the SVG user agent should provide facility for the mass selection of entire text passages, for whatever text is in scope. If a text content element has focus, such a select-all operation should include only the contents of that element. If no text content element is in focus, the select-all operation should select all the text in the document. This may be a progressive operation, widening the scope with each subsequent operation. For example, a common idiom is to allow a user to select text with a single click on a word, with first the word selected, then the entire passage with a second click, then the entire document with a third click. For purposes of accessibility, the user agent must allow any such operation to be performed by keyboard as well as pointer device (such as in the `ctrl/command+A` "select-all" keyboard shortcut), and should also expose appropriate accessibility APIs.

10.14 Text search

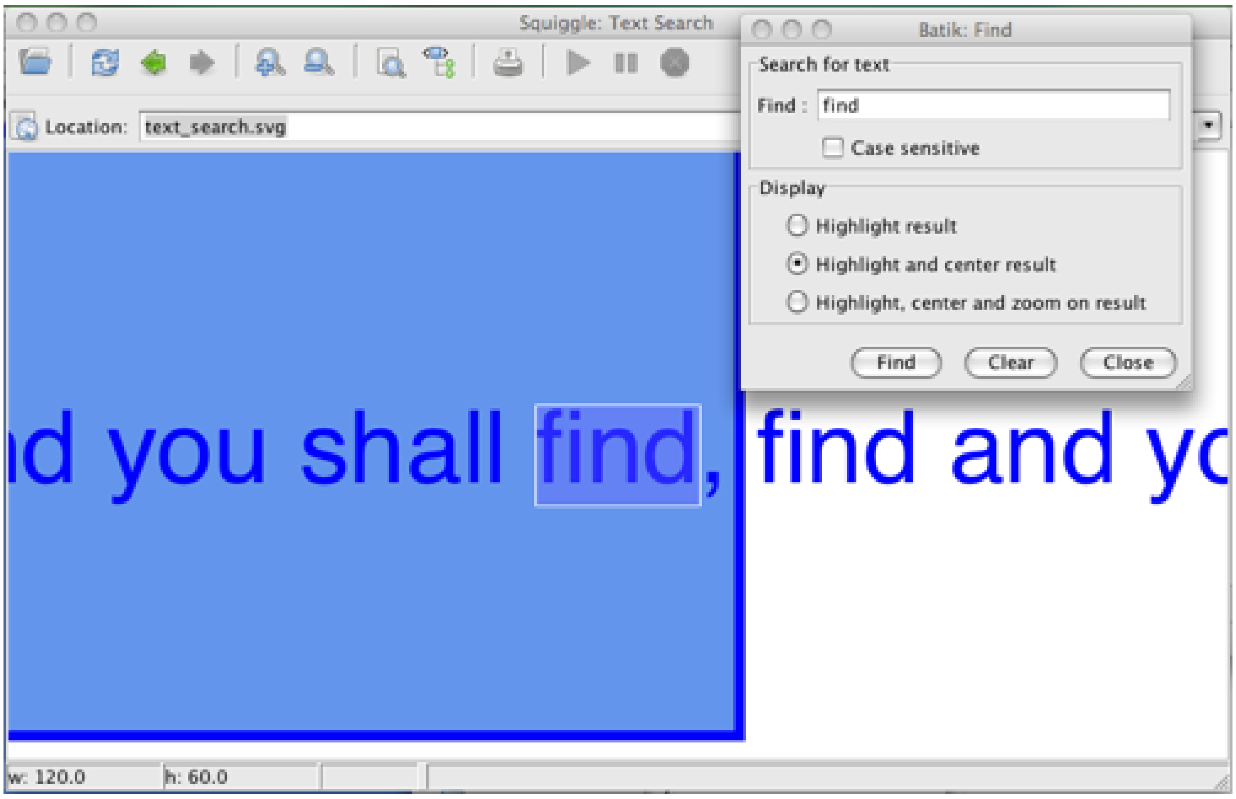
If the user agent supports searching for text strings, then it must support searching for text strings in SVG content as well. An SVG viewer which supports search must allow the user to find all instances of the searched text string within the document that are in the rendering tree (e.g., those with a **'display'** property other than **none**), and must highlight or otherwise indicate each instance. SVG viewers which allow sequential searches for text strings must pan and zoom the viewport, as appropriate, in order to show the text string in context, and are recommended to adjust the viewport as if there had been a fragment identifier link traversal to the element containing the text string.

In other words, if the containing text content element is too large to be enclosed in the viewport, the SVG user agent is recommended to zoom out, but if the text does fit, the user agent is recommended only to pan, and not to zoom. In order to enable maximum usability, authors should create their content accordingly, breaking text into discrete text content elements that fit within the expected viewport at a readable size, while providing sufficient context. Additionally, users must be provided a way to zoom in on text that is too small for the user to read.

Example 'text_search.svg' below contains a long text string which extends outside of the initial viewport, and which needs to be adjusted when searching for one of the words outside the viewport. The image shows the results of a text search using the Batik SVG toolkit.

Example: text_search.svg

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
     width="100%" height="100%" viewBox="0 0 120 60">

  <title>Text Search</title>
  <desc>
    An example of text that extends beyond the viewport,
    requiring the user agent to pan the viewport when
    words are searched for.
  </desc>

  <!--
    This rectangle is the same position and dimensions as the viewBox,
    illustrating the initial area of the viewport.
  -->
  <rect x="0" y="0" width="120" height="60" fill="#6495ed" stroke="blue" />

  <!--
    The contents of this text element extend beyond the initial viewport.
  -->
  <text x="120" y="35" text-anchor="middle" fill="blue"
        font-size="10" font-family="Helvetica">
        Seek and you shall find, find and you will search.
  </text>

</svg>
```


Squiggle: Text Search
Location: text_search.svg
nd you shall find, find and yo
w: 120.0
h: 60.0
Batik: Find
Search for text
Find : find
Case sensitive
Display
Highlight result
Highlight and center result
Highlight, center and zoom on result
Find
Clear
Close

11 Painting: Filling, Stroking, Colors and Paint Servers

Contents

11.1 Introduction 129
11.2 Specifying paint 129
11.3 Fill properties 130
11.4 Stroke properties 132
11.5 Non-scaling stroke 135
11.6 Simple alpha compositing 136
11.6.1 Compositing the currentColor value 137
11.7 The 'viewport-fill' property 137
11.8 The 'viewport-fill-opacity' property 138
11.9 Controlling visibility and rendering 139
11.10 Rendering hints 140
11.10.1 The 'color-rendering' property 140
11.10.2 The 'shape-rendering' property 140
11.10.3 The 'text-rendering' property 141
11.10.4 The 'image-rendering' property 142
11.10.5 The 'buffered-rendering' property 142
11.11 Inheritance of painting properties 143
11.12 Object and group opacity: the 'opacity' property 143
11.13 Color 146
11.13.1 Syntax for color values 146
11.13.2 HTML color keywords 147
11.14 Paint servers 147
11.14.1 System paint servers 147
11.14.2 The 'solidColor' element 148
11.14.3 The 'color' property 150
11.15 Gradients 151
11.15.1 Linear gradients 151
11.15.2 Radial gradients 153
11.15.3 Defining gradient stops: the 'stop' element 154

11.1 Introduction

Graphics elements, including text content elements and 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 color, possibly with some level of transparency
- a gradient (linear or radial)

SVG uses the general notion of a **paint server**. Apart from system paint, paint servers are specified using a local IRI reference on a **'fill'** or **'stroke'** property. Gradients and 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> |
	<FuncIRI> [none | currentColor | <color>] |

<system paint> |
inherit

none
Indicates that no paint shall be applied.

currentColor
Indicates that painting shall be done using the color specified by the current animated value of 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>
the explicit color (in the sRGB color space [SRGB]).

<FuncIRI> [none | currentColor | <color>]
The <FuncIRI> specifies a paint server such as a gradient. The fragment identifier of the <FuncIRI> provides a link to the paint server (e.g., a gradient or **'solidColor'**) that shall be used to paint the current object. SVG Tiny 1.2 user agents are only required to support local IRI references. If the IRI reference is invalid (for example, it points to an object that doesn't exist or the object is not a valid paint server or it is a non-local IRI reference and the viewer does not support it), then the fallback value (if specified) is used; otherwise it must be treated as if **none** was specified.

<system paint>
A system paint server

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:	"none", system paint, specified <color> value or absolute IRI

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'

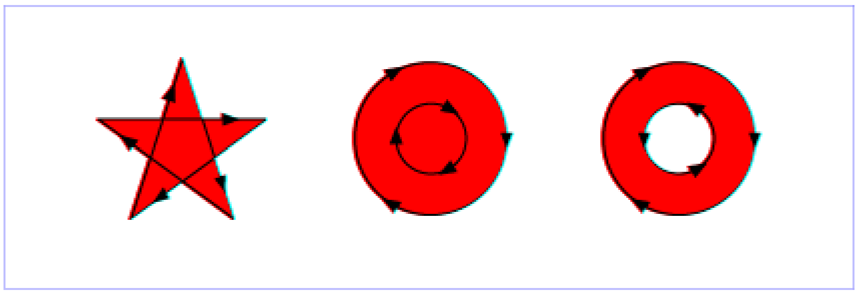
Value:	nonzero | evenodd | inherit
Initial:	nonzero
Applies to:	shapes and text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified value, except inherit

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:

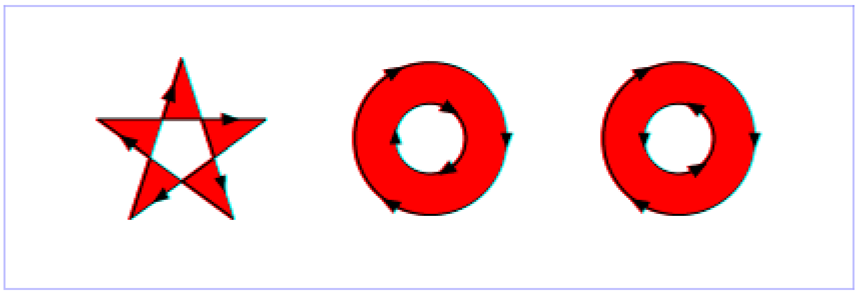
nonzero

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:



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:



(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 is to be used to fill 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: **'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:	"none", system paint, specified <color> value or absolute IRI

The **'stroke'** property shall paint along the outline of the given graphics 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 not be stroked if the **'stroke-linecap'** property has a value of **butt** and shall be stroked if the **'stroke-linecap'** property has a value of **round** or **square**, producing respectively a circle or a square centered at the given point.

This property contributes to an element's decorated bounding box.

'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.

This property contributes to an element's decorated bounding box.

'stroke-linecap'

Value:	butt | round | square | inherit
Initial:	butt
Applies to:	shapes and text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual

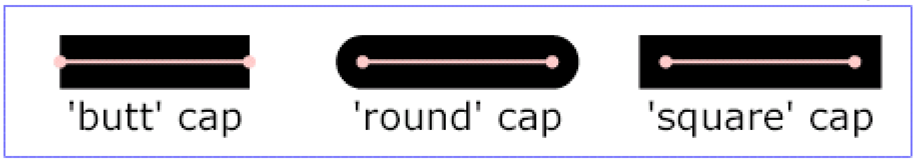
Animatable:	yes
Computed value:	Specified value, except inherit

'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.



This property contributes to an element's decorated bounding box.

'stroke-linejoin'

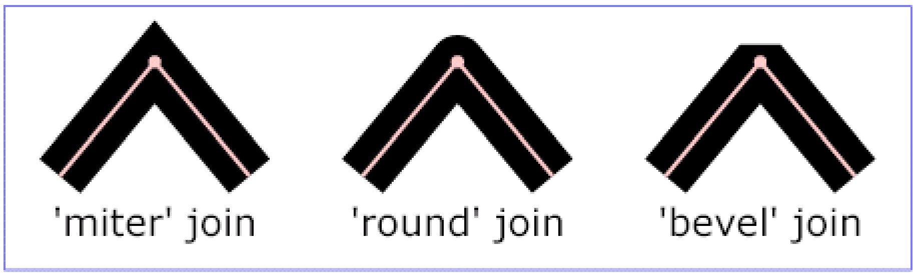
Value:	miter | round | bevel | inherit
Initial:	miter
Applies to:	shapes and text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified value, except inherit

'stroke-linejoin' specifies the shape which shall be used at the corners of shapes when they are stroked.

miter
The shape drawn at the corner of shapes shall be as per the drawing below.

round
The shape drawn at the corner of shapes shall be as per the drawing below.

bevel
The shape drawn at the corner of shapes shall be as per the drawing below.



This property contributes to an element's decorated bounding box.

'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:

```
miterLimit = miterLength / stroke-width = 1 / sin(theta / 2)
```

For example, a miter limit of 1.414 converts miters to bevels for theta less than 90 degrees, a limit of 4.0 converts them for theta less than approximately 29 degrees, and a limit of 10.0 converts them for theta less than approximately 11.5 degrees.

This property contributes to an element's decorated bounding box.

'stroke-dasharray'

Value:	none | <list-of-lengths> | 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. The **<list-of-lengths>** contains the list of <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"**. The computed value of the attribute **'stroke-linecap'** is applied to both sides of each dash. If a dash has zero length, linecaps are still added if the stroke-linecap values **round** and **square** are used.

none

Indicates that no dashing shall be used. If stroked, the line must be drawn solid.

<list-of-lengths>

The list of <length>s that specify the lengths of alternating dashes and gaps that must be used. 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.

Note: Certain cases regarding the behavior of **'stroke-dasharray'** are not fully specified because SVG Tiny implementations often rely on underlying graphics libraries with predetermined behaviors they cannot easily change. Examples include: rendering of **'stroke-linejoin'** and **'stroke-linecap'** in case a dash ends exactly at a corner of two path segments, continuation of stroke-dasharray in subpaths, and others. These cases may be fully specified in version SVG 1.2 Full. Additional attributes, such as dash-caps that can be defined separately from linecaps may be added. Authors are encouraged not to rely on a specific behavior of a specific viewer for **'stroke-dasharray'** regarding these currently unspecified cases.

'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. When rendering a **'path'** element with multiple subpaths, the value of **'stroke-dashoffset'** should start from scratch with the original value of **'stroke-dashoffset'** for each subpath. SVG 1.2 Full may be stricter and also add an additional attribute to change this behavior.

<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 that is to 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. For example, 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, SVG Tiny 1.2 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 being able to specify the non-scaling stroke behavior.

'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

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.

non-scaling-stroke

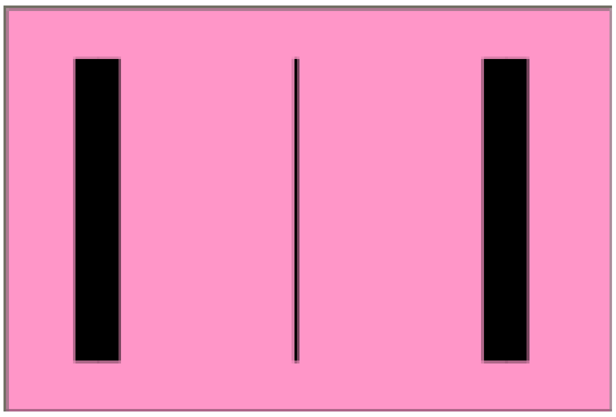
Modifies the way an object is stroked. Normally stroking involves calculating stroke outline of the shape's path in current user space and filling that outline with the stroke paint (color or gradient). With the non-scaling-stroke vector effect, stroke outline shall be calculated in the "host" coordinate space instead of user coordinate space. More precisely: a user agent establishes a host coordinate space which in SVG Tiny 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. The resulting outline is transformed back to the user coordinate system. (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 the host coordinate system.

Below is an example of the non-scaling-stroke **'vector-effect'**.

Example: non-scaling-stroke.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="6cm" height="4cm" viewBox="0 0 600 400"
     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 operations:

- **'color-rendering'**

11.6.1 Compositing the currentColor value

The **currentColor** value 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'

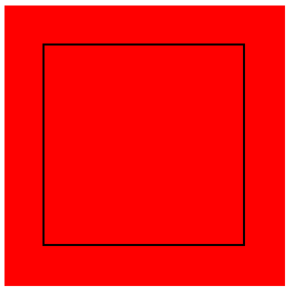
Value:	none | currentColor | <color> | inherit
Initial:	none
Applies to:	viewport-creating elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	"none" or specified <color> value, except inherit

If the value of **'viewport-fill'** is **none**, then no paint operation is applied to the viewport.

Below is an example of **'viewport-fill'**.

Example: 11_02.svg

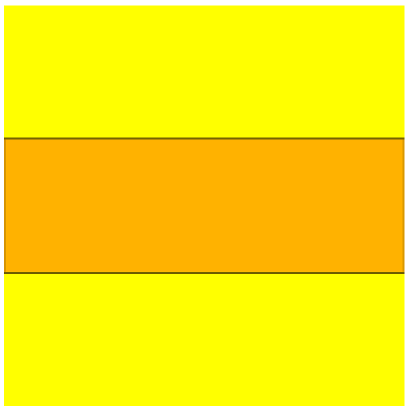
```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     viewport-fill="red">

  <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" baseProfile="tiny"
     viewBox="0 0 300 100" viewport-fill="yellow">

  <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'

Value:	<opacity-value> | inherit
Initial:	1.0
Applies to:	viewport-creating elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified value, except inherit

<opacity-value>

The opacity of the painting operation that is to 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 and rendering

SVG uses two properties, **'display'** and **'visibility'**, to control the rendering of graphics 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 excluded from the rendering tree; 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 visually 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 visually 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 visually rendered on the canvas. Regarding events, if **'display'** is set to **none**, the element receives no events which require the element to be in the rendering tree (for example mouse 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 content block elements), media elements and text sub-elements (e.g., **'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 or made audible (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 or made audible by the SVG user agent.

The **'display'** property only affects the direct rendering or audibility 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 of this property is found in CSS 2 ([CSS2], section 9.2.5).

'visibility'

Value:	visible | hidden | collapse | inherit
Initial:	visible
Applies to:	graphics elements (including the text content block elements), media elements and text sub-elements (e.g., **'tspan'** and **'a'**)
Inherited:	yes

Percentages:	N/A
Media:	all
Animatable:	yes
Computed value:	Specified value, except inherit

visible
The current graphics element or media element shall be visible.

hidden or collapse
The current graphics element or media element shall be invisible (i.e., nothing is painted on the canvas).

Note that, unlike the **'display'** property, the **'visibility'** property does not have any affect on the audibility of any media element. To control the audibility of an element, use the **'display'** or **'audio-level'** properties.

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 of this property is found in CSS 2 ([CSS2], section 11.2).

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 versus 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 content block 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.

11.10.5 The **'buffered-rendering'** property

The creator of SVG content might want to provide a hint to the implementation about how often an element is modified to make speed vs. memory tradeoffs as it performs rendering. The **'buffered-rendering'** property provides a hint to the SVG user agent about how to buffer the rendering of elements:

'buffered-rendering'

Value:	auto | dynamic | static | inherit
Initial:	auto
Applies to:	container elements and graphics elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified value, except inherit

auto
: Indicates that the user agent is expected to use a reasonable compromise between speed of update and resource allocation.

dynamic
: Indicates that the element is expected to be modified often.

static

Indicates that the element is not expected to be modified often. This suggests that user agent may be able to allocate resources, such as an offscreen buffer, that would allow increased performance in redraw. It does not mean that the element will never change. If an element is modified when the value is **'static'**, then redraw might have reduced performance.

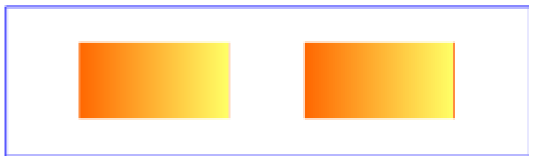
11.11 Inheritance of painting properties

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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="7cm" height="2cm" viewBox="0 0 700 200">

  <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 be set on, and only apply to, the **'image'** element. Note: if the value is set to **inherit**, then the initial value of 1 for the opacity property will be used, meaning full opacity. This is the same as not specifying it at all.

'opacity'

Value:	<opacity-value> | inherit
Initial:	1
Applies to:	**'image'** element
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified value, except inherit

<opacity-value>

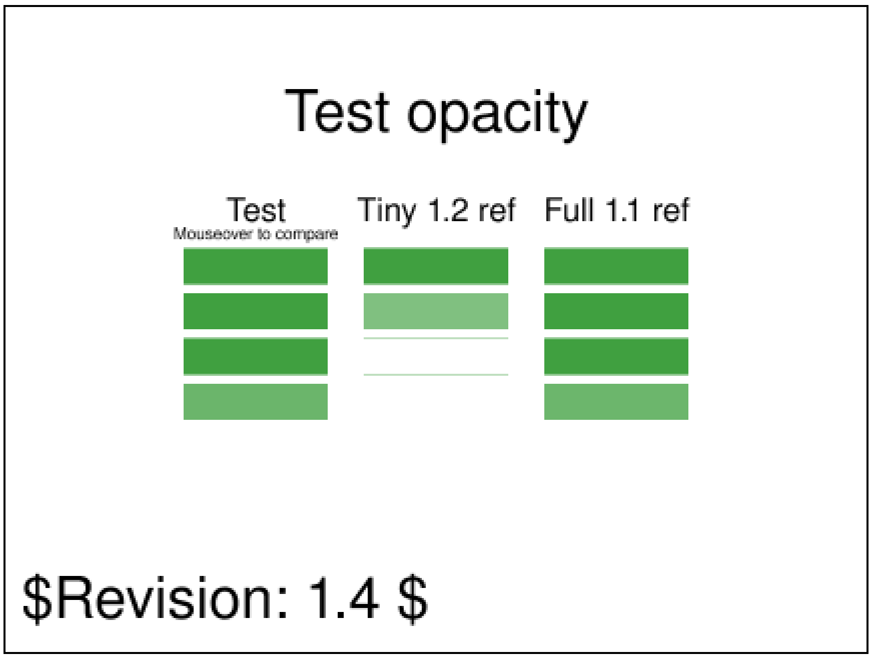
The uniform opacity setting that must applied across an entire object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) must be clamped to this range. (See Clamping values which are restricted to a particular range.)

Below is an example of **'opacity'** which illustrates the difference in behavior between SVG Basic/Full 1.1 and SVG Tiny 1.2.

Example: struct-image-201-t.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.2" baseProfile="tiny" xml:id="svg-root" width="100%" height="100%"
  viewBox="0 0 480 360" xmlns="http://www.w3.org/2000/svg" xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <!--======================================================================-->
  <!--=  Copyright 2007 World Wide Web Consortium, (Massachusetts          =-->
  <!--=  Institute of Technology, European Research Consortium for         =-->
  <!--=  Informatics and Mathematics (ERCIM), Keio University).            =-->
  <!--=  All Rights Reserved.                                              =-->
  <!--=  See http://www.w3.org/Consortium/Legal/.                          =-->
  <!--======================================================================-->
  <metadata>
      <p>
        This test shows the differences in opacity-handling between SVG Tiny 1.2 and SVG Full 1.1.
      </p>
          <p>
                The test has passed if the leftmost column looks like either of the other two columns.
                SVG Tiny 1.2 requires only what is portrayed by the middle column, but does not require
                user agents that implement both SVG 1.1 and 1.2 Tiny to follow the more limited
                opacity-handling in 1.2 Tiny.
          </p>
          <p>
                In SVG Tiny 1.2 the opacity property is only allowed on the image element itself.
                If it's encountered anywhere else it must be treated as an unsupported value.

                NOTE: This test is not valid 1.2 Tiny because it's using opacity on something
                other than the image element.
          </p>
</metadata>
  <title xml:id="test-title">&#36;RCSfile: struct-image-201-t.svg,v $</title>
  <defs>
    <font-face
      font-family="SVGFreeSansASCII"
      unicode-range="U+0-7F">
      <font-face-src>
```

```
        <font-face-uri xlink:href="SVGFreeSans.svg#ascii"/>
      </font-face-src>
    </font-face>
  </defs>
  <g xml:id="test-body-content" font-family="SVGFreeSansASCII,sans-serif" font-size="18">

        <text x="240" y="70" text-anchor="middle" font-size="32">Test opacity</text>

        <g id="test" transform="translate(-100 0)">
                <text x="240" y="120" text-anchor="middle">Test</text>
                <text x="240" y="130" text-anchor="middle" font-size="9">Mouseover to compare</text>
                <rect id="r1" x="200" y="135" height="20" width="80" fill="green"/>
                <rect id="r2" x="200" y="160" height="20" width="80" fill="green"/>
                <rect id="r3" x="200" y="185" height="20" width="80" fill="green"/>
                <rect id="r4" x="200" y="210" height="20" width="80" fill="green"/>

                <g pointer-events="none">
                        <image width="460" height="20" x="10" y="135" xlink:href="1pixelwhite.png"
preserveAspectRatio="none" opacity="0.25"/>
                        <g opacity="0.5">
                                <image width="460" height="20" x="10" y="160" xlink:href="1pixelwhite.png"
preserveAspectRatio="none" opacity="0.5"/>
                                <rect id="r5" x="200" y="185" height="20" width="80" fill="white" opacity="0.5"/>

                        </g>
                        <g opacity="0.75">
                                <g opacity="inherit">
                                        <image width="460" height="20" x="10" y="210" xlink:href="1pixelwhite.png"
preserveAspectRatio="none" opacity="inherit"/>
                                </g>
                        </g>
                </g>

                <ev:listener event="mouseover" observer="r1" handler="#handler"/>
                <ev:listener event="mouseout" observer="r1" handler="#handler"/>
                <ev:listener event="mouseover" observer="r2" handler="#handler"/>
                <ev:listener event="mouseout" observer="r2" handler="#handler"/>
                <ev:listener event="mouseover" observer="r3" handler="#handler"/>
                <ev:listener event="mouseout" observer="r3" handler="#handler"/>
                <ev:listener event="mouseover" observer="r4" handler="#handler"/>
                <ev:listener event="mouseout" observer="r4" handler="#handler"/>
                <handler id="handler">
                        if(event.type == "mouseover")
                        {
                                event.target.setFloatTrait("width", 280);
                                if(event.target.id == "r3")
                                        document.getElementById("r5").setFloatTrait("width", 280);
                        }
                        else
                        {
                                event.target.setFloatTrait("width", 80);
                                if(event.target.id == "r3")
                                        document.getElementById("r5").setFloatTrait("width", 80);
                        }
                </handler>
        </g>

        <g id="tiny12reference">
                <text x="240" y="120" text-anchor="middle">Tiny 1.2 ref</text>
                <rect x="200" y="135" height="20" width="80" fill="green"/>
                <rect x="200" y="160" height="20" width="80" fill="green"/>
                <rect x="200" y="185" height="20" width="80" fill="green"/>
                <rect x="200" y="210" height="20" width="80" fill="green"/>
                <rect x="200" y="135" height="20" width="80" fill="white" fill-opacity="0.25"/>
                <rect x="200" y="160" height="20" width="80" fill="white" fill-opacity="0.5"/>
                <rect x="200" y="185" height="20" width="80" fill="white" fill-opacity="1"/>
                <rect x="200" y="210" height="20" width="80" fill="white" fill-opacity="1"/>
        </g>

        <g id="full11reference" transform="translate(100 0)">
                <text x="240" y="120" text-anchor="middle">Full 1.1 ref</text>
                <rect x="200" y="135" height="20" width="80" fill="green"/>
                <rect x="200" y="160" height="20" width="80" fill="green"/>
                <rect x="200" y="185" height="20" width="80" fill="green"/>
                <rect x="200" y="210" height="20" width="80" fill="green"/>
                <rect x="200" y="135" height="20" width="80" fill="white" fill-opacity="0.25"/>
                <rect x="200" y="160" height="20" width="80" fill="white" fill-opacity="0.25"/>
                <rect x="200" y="185" height="20" width="80" fill="white" fill-opacity="0.25"/>
                <rect x="200" y="210" height="20" width="80" fill="white" fill-opacity="0.421875"/>
        </g>

    </g>
    <g font-family="SVGFreeSansASCII,sans-serif" font-size="32">
    <text xml:id="revision" x="10" y="340" stroke="none"
      fill="black">&#36;Revision: 1.4 $</text>
    </g>
    <rect xml:id="test-frame" x="1" y="1" width="478" height="358" fill="none" stroke="#000"/>
</svg>
```



11.13 Color

In SVG Tiny 1.2, all colors are specified in the sRGB color space [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, and 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** (i.e. 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 white space. 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 white space. 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 below (originally from HTML 4 [HTML4]) must be supported, with the further restriction that they must be lowercase.

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)

11.14 Paint servers

With SVG, you can fill (i.e., paint the interior of) or stroke (i.e., paint the outline of) shapes and text using one of the following:

- color (using <color> or a reference to a **'solidColor'** element)
- a system paint
- 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.

Apart from system paint, paint servers are referenced using a local IRI reference on a **'fill'** or **'stroke'** property.

11.14.1 System paint servers

The following list of system paint servers must be supported. If a paint specification specifies one of the system paint servers, then the user agent must either paint using a system-provided paint server or paint with a substitute paint server, such as a color or gradient. System paint servers often depend on the operating system, user choices, and the implementation. Substitute paint servers should attempt to match the appearance of corresponding user interface elements on the platform, including user color choices. In environments which do not provide adequate system paint server APIs, a conformant user agent may use substitute paint servers which do not necessarily match the environment's system paint servers.

The computed value of a paint specified as a system paint is the specified value.

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.

11.14.2 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>
```

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 <color> value, except inherit

The **'solid-opacity'** property defines the opacity of the **'solidColor'**.

'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).

Properties shall inherit into the **'solidColor'** element from its ancestors; properties shall *not* inherit from the element referencing the **'solidColor'** element.

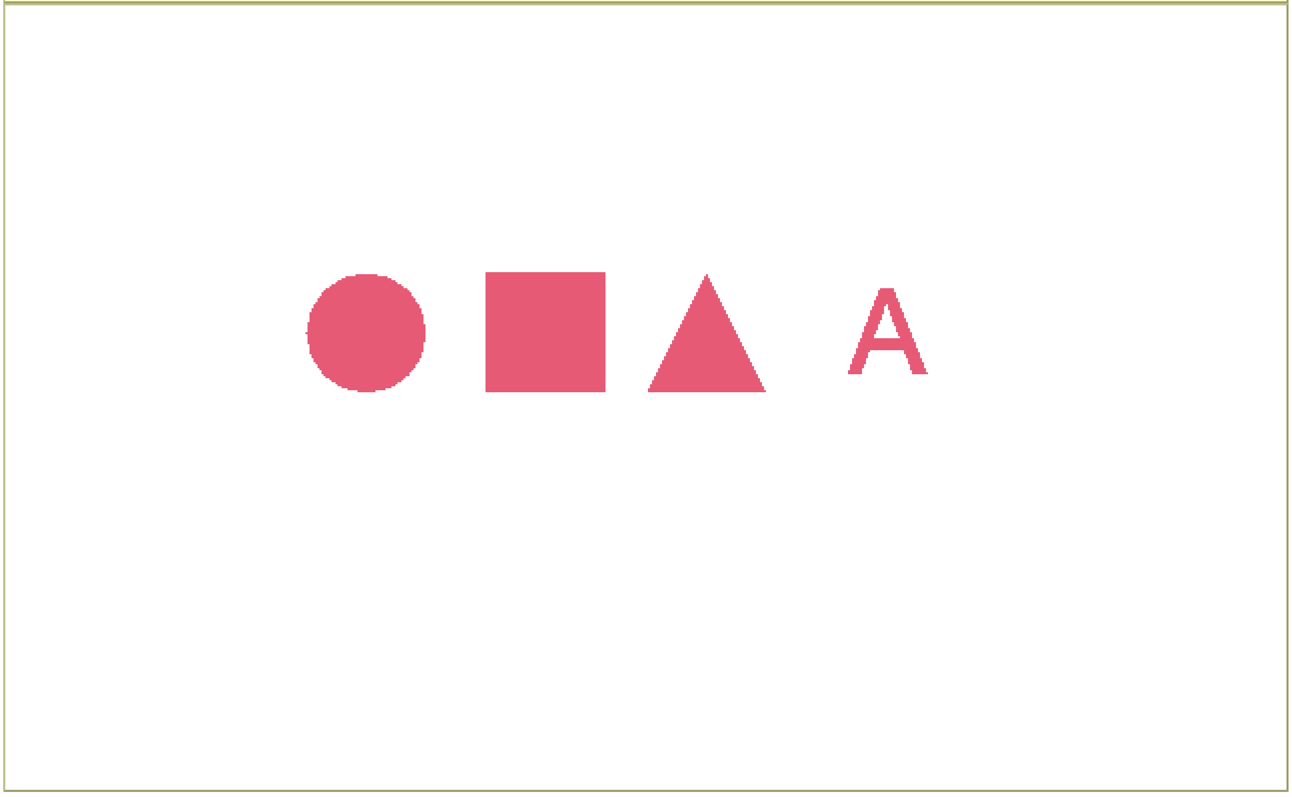
'solidColor' elements are never rendered directly; their only usage is as something that can be referenced using the **'fill'** and **'stroke'** properties. The **'display'** property does not apply to the **'solidColor'** element; thus, **'solidColor'**

elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'solidColor'** elements are available for referencing even when the **'display'** property on the **'solidColor'** element or any of its ancestors is set to **none**.

Below is an example of the **'solidColor'** element:

Example: solidcolor.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="480" height="360" viewBox="0 0 480 360">

  <title>'solidColor' 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>
```



11.14.3 The 'color' property

The **'color'** property, which is 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'

Value:	<color> | inherit
Initial:	depends on user agent
Applies to:	None. Indirectly affects other properties via **currentColor**
Inherited:	yes

Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified <color> value, except inherit

Except for any additional information provided in this specification, the normative definition of the property is found in CSS 2 ([CSS2], section 14.1).

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, **'linearGradient'** and **'radialGradient'**.

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.GradientCommon.attr'/>
  <ref name='svg.Core.attr'/>
  <ref name='svg.X12Y12.attr'/>
</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.

The lacuna value is **'objectBoundingBox'**.

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.

The lacuna value is **'0'**.

Animatable: yes.

y1 = **"<coordinate>"**
See **'x1'**.
The lacuna value is **'0'**.
Animatable: yes.

x2 = **"<coordinate>"**
See **'x1'**.
The lacuna value is **'1'**.
Animatable: yes.

y2 = **" <coordinate>"**
See **'x1'**.
The lacuna value is **'0'**.
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.

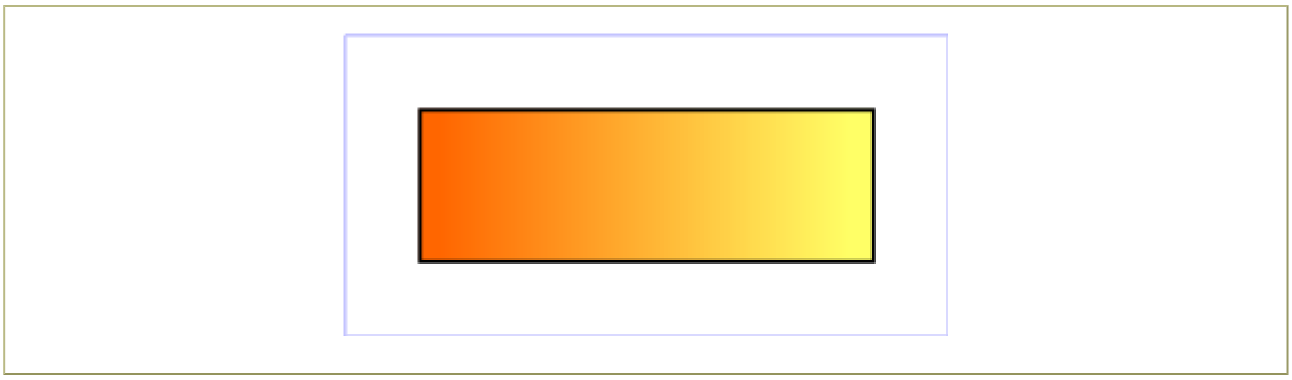
Properties shall inherit into the **'linearGradient'** element from its ancestors; properties shall *not* inherit from the element referencing the **'linearGradient'** element.

'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 13_01 shows how to fill a rectangle by referencing a linear gradient paint server.

Example: 13_01.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="8cm" height="4cm" viewBox="0 0 800 400">

  <desc>Example 13_01 - 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.

Schema: radialGradient

```
    <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.GradientCommon.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.

The lacuna value is **'objectBoundingBox'**.

Animatable: yes.

`cx` = **"<coordinate>"**

'cx', **'cy'** and **'r'** define the largest (i.e., outermost) circle for the radial gradient and the **0** gradient stop is mapped to (**'cx'**, **'cy'**).

The lacuna value is **'0.5'**.

Animatable: yes.

`cy` = **"<coordinate>"**

See **'cx'**.

The lacuna value is **'0.5'**.

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. The lacuna value is **'0.5'**.

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.

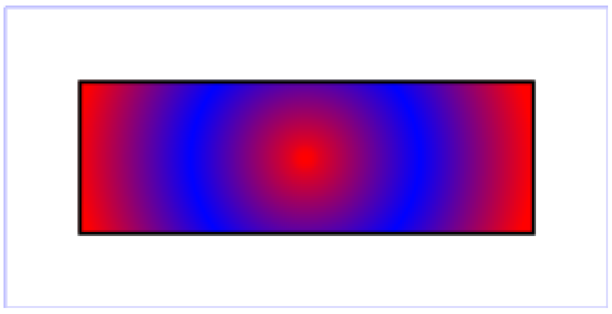
Properties shall inherit into the **'radialGradient'** element from its ancestors; properties shall *not* inherit from the element referencing the **'radialGradient'** element.

'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 13_02 shows how to fill a rectangle by referencing a radial gradient paint server.

Example: 13_02.svg

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="8cm" height="4cm" viewBox="0 0 800 400">

  <desc>Example 13_02 - 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 Defining gradient stops: the 'stop' element

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'/>
  <optional>
    <attribute name='offset' svg:animatable='true' svg:inheritable='false'>
      <ref name='Number.datatype'/>
    </attribute>
  </optional>
</define>
```

Attribute definitions:

`offset` = **"<number>"**

The **'offset'** attribute is a <number> 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.

The lacuna value is **'0'**.

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
Initial:	black
Applies to:	**'stop'** elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes
Computed value:	Specified <color> value, except inherit

The **'stop-opacity'** property specifies the opacity that shall be used for the gradient **'stop'**.

'stop-opacity'

Value:	<opacity-value> | inherit
Initial:	1
Applies to:	**'stop'** elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

Computed value: 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).

<opacity-value>

The opacity of the **'stop-color'** for the current gradient **'stop'**. 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.)

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 **'stop'** elements are specified to have a gradient effect. If no **'stop'** elements are specified, then painting shall occur as if **none** were specified as the paint style. If one **'stop'** is specified, 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=".1999999999" stop-color="red"/>
  <stop offset=".2" stop-color="blue"/>
  <stop offset="1" stop-color="black"/>
  ```

  which is a gradient that goes smoothly from white to red, then abruptly shifts from red to blue, and then goes smoothly from blue to black.
- Colors and opacities are interpolated separately, and the resulting gradient is composited using simple alpha compositing. In particular:

  ```
  <stop offset="0" stop-color="#F00" stop-opacity="0"/>
  <stop offset="1" stop-color="#0F0" stop-opacity="1"/>
  ```

  will produce a gradient from fully transparent red, via partly transparent dark yellow, to fully opaque lime.
- All gradient stops must be converted into the interpolation color space. Interpolation between gradient stop colors must occur in the interpolation color space.
- SVG Tiny 1.2 user agents have the option to interpolate gradients in either sRGB or in linearRGB color space. Both color spaces have the same color gamut (see [SRGB]).
- Other W3C specifications may allow alternative interpolation color spaces to be specified.

12 Multimedia

Contents

12.1 Media elements 157
12.1.1 Media timeline and document timeline 157
12.1.2 Media availability 158
12.1.3 Platform limits 159
12.1.4 Audio mixing for 'audio' and 'video' elements 159
12.1.5 Discrete control of audio and video 159
12.1.6 Controlling media playback through script 159

12.2 The 'audio' element 159
12.3 The 'video' element 161
12.3.1 Restricting the transformation of the 'video' element 164
12.3.2 Restricting compositing of the 'video' element 165
12.3.3 Examples 165

12.4 The 'animation' element 168
12.5 The 'audio-level' property 170
12.6 Attributes for runtime synchronization 170
12.7 The 'initialVisibility' attribute 172

12.1 Media elements

SVG supports media elements similar to the SMIL 2.1 Media Elements ([SMIL21], chapter 7). Media elements define their own timelines within their time container. All SVG media elements support the SVG timing attributes and runtime synchronization attributes. The default event-base element for all SVG media elements is the element itself.

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 xml: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 $t0+t1+t2 \geq 6s$. 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 xml: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'** element is fully parsed and processed, i.e. at time *t0*. The video will also start at document time 0, but since the video will only be processed when document time is *t0+t1+t2*, the video will start displaying the frame at time *t0+t1+t2* 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 values of the **'syncBehavior'** attributes are changed from the default value to **'independent'**.

Example: video-timelineBegin-03.svg

```
<?xml version="1.0"?>
<svg xml: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 xml: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 example 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 ([PNG], section 4.5.2)). In that case, it is an implementation choice to decide whether 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.

12.1.4 Audio mixing for **'audio'** and **'video'** elements

If two or more audio streams from **'audio'** or **'video'** elements 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.

12.1.5 Discrete control of audio and video

Authors may wish to independently control both the visual and auditory aspects of the **'video'** element. Through a combination of the various properties available, all permutations are possible, as described below:

- play both video and audio: this is the default setting, and nothing special needs to be done
- play video with no audio: use the **'audio-level'** property with a value of **0**
- play audio with no video: use the **'visibility'** property with a value of **hidden**
- hide both video and audio: use the **'display'** property with a value of **none**

12.1.6 Controlling media playback through script

In addition to setting fixed timeline attribute values or using declarative animation to control the playback of media elements such as **'audio'**, **'video'**, and **'animation'**, SVG allows scripted control. See the uDOM section on Multimedia control for details.

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.

The **'audio'** element must reference content with an audio stream.

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.ContentTypeAnim.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 invalid IRI reference is an unsupported value. An empty string value (**xlink:href=""**) disables playback of the element. The lacuna value is the empty string.

When the value of this attribute is animated or otherwise modified, if the media timeline can be controlled, then the media timeline is restarted only if the **'syncBehavior'** attribute is set to **independent**. If the media timeline cannot be controlled, then the media timeline is unaffected by such modification.

Animatable: yes.

`type` = **"<content-type>"**

The audio format. Implementations may choose not to fetch audios of formats that they do not support. For optimizing download time by requiring a particular content format authors are encouraged to use **'requiredFormats'**, instead of **'type'**.

Animatable: yes.

`Runtime 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="application/ogg"
        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>
```

This specification does not mandate support for any particular audio format. Content can check for a particular audio codec with the **'requiredFormats'** conditional processing attribute.

12.3 The 'video' element

The **'video'** element specifies a video file which is to be rendered to provide synchronized video. 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.

The **'video'** element must reference content with a video stream.

The **'video'** element produces a rendered result, and thus has **'width'**, **'height'**, **'x'** and **'y'** attributes.

The **'video'** element can have two different transform behaviors, either geometric or pinned, depending on the value of the transformBehavior attribute. If the transform behavior is geometric, the **'video'** element must establish a new viewport for the referenced document as described in Establishing a new viewport. In this case, 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. In the case of pinned transform behavior, a new viewport must not be established. As such, **'viewport-fill'**, **'viewport-fill-opacity'**, **'width'**, **'height'**, or **'preserveAspectRatio'** have no effect.

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.ContentTypeAnim.attr'/>
  <ref name='svg.InitialVisibility.attr'/>
  <optional>
    <attribute name='transformBehavior' 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' 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. The lacuna value is **'0'**.

  If the transform behavior 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.

  Animatable: yes.

`y` = **"<coordinate>"**
: The *y*-axis coordinate of the rectangular region into which the video is placed. The lacuna value is **'0'**.

  If the transform behavior 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'** for the interpretation of this attribute.

  Animatable: yes.

`width` = **"<length>"**
: The width of the rectangular region into which the video is placed. A negative value shall be treated as unsupported. The lacuna value is **'0'**.

  If the transform behavior of the video is geometric, a value of zero shall disable rendering of the element. If it is pinned, this attribute shall have no effect on rendering.

  Animatable: yes.

`height` = **"<length>"**
: The height of the rectangular region into which the video is placed. A negative value shall be treated as unsupported. The lacuna value is **'0'**.

  If the transform behavior of the video is geometric, a value of zero shall disable rendering of the element. If it is pinned, this attribute shall have no effect on rendering.

  Animatable: yes.

`xlink:href` = **"<IRI>"**
: An IRI reference to the video content. An invalid IRI reference is an unsupported value. An empty string value (**xlink:href=""**) disables rendering of the element. The lacuna value is the empty string.

  When the value of this attribute is animated or otherwise modified, if the media timeline can be controlled, then the media timeline is restarted only if the **'syncBehavior'** attribute is set to **independent**. If the media timeline cannot be controlled, then the media timeline is unaffected by such modification.

  Animatable: yes.

`preserveAspectRatio` = **[defer] <align> [<meet>]**
: Indicates whether or not to force uniform scaling. (See the **'preserveAspectRatio'** for the syntax of <align> and the interpretation of this attribute.)

  Animatable: yes.

`type` = **"<content-type>"**
: The video format. Implementations may choose not to fetch videos of formats that they do not support. For optimizing download time by requiring a particular content format authors are encouraged to use **'requiredFormats'**, instead of **'type'**.

  Animatable: yes.

`transformBehavior` = **"geometric" | "pinned" | "pinned90" | "pinned180" | "pinned270"**
: 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.

`Runtime 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 Tiny 1.2 viewer and support for a Windows AVI using Motion JPEG; this is a 3.7M video file).

This specification does not mandate support for any particular video format. Content can check for a particular video codec with the **'requiredFormats'** conditional processing 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. SVG Tiny 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, SVG Tiny 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 the **'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"**

Defines whether a video is transformed/resampled (in essence treated as a geometric rectangle) or pinned/unresampled (i.e., treated as a pin point for a non-geometric blit region).

The attribute can take one of the five following values:

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 is subject to transformation. This is the lacuna value.

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)"**.

If one of the four values **'pinned'**, **'pinned90'**, **'pinned180'** and **'pinned270'** is specified, 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 be 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. The values of the **'width'** and **'height'** attributes in this case shall have no effect on the rendering of the video.

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. SVG Tiny 1.2 therefore introduces the **'overlay'** attribute and a corresponding feature string: **http://www.w3.org/Graphics/SVG/feature/1.2/#ComposedVideo**. A viewer supporting compositing of video must render the **'video'** element according to the SVG painter's model, and thus 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"**

Defines whether a **'video'** is rendered according to the SVG painter's model or if it must be positioned on top of all other SVG elements.

The attribute value can be either of the following:

top

The **'video'** element must not be composited on 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 painter's model. This is the lacuna value.

Animatable: no.

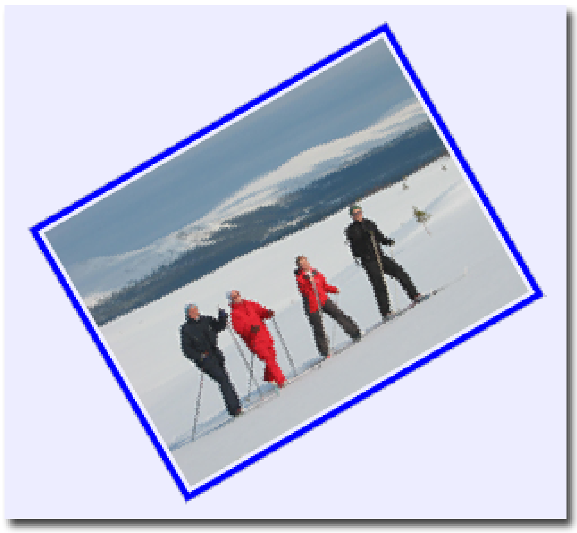
If multiple **'video'** elements have **overlay="top"**, the drawing order of those **'video'** elements follows the typical SVG rendering order.

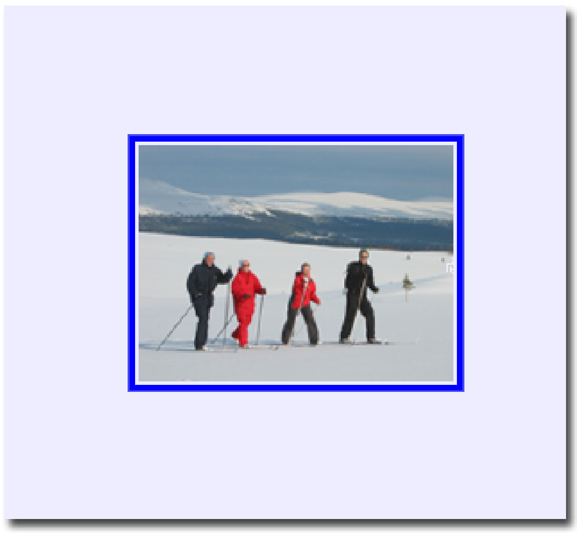
12.3.3 Examples

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 don'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/Graphics/SVG/feature/1.2/#TransformedVideo feature
string</desc>
     <switch>

       <!-- Transformed video group -->
       <g requiredFeatures="http://www.w3.org/Graphics/SVG/feature/1.2/#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="98" y="236"/>
       </g>
     </switch>
</svg>
```


SVGT 1.2 viewer supporting transformed video.


SVGT 1.2 viewer not supporting transformed video.

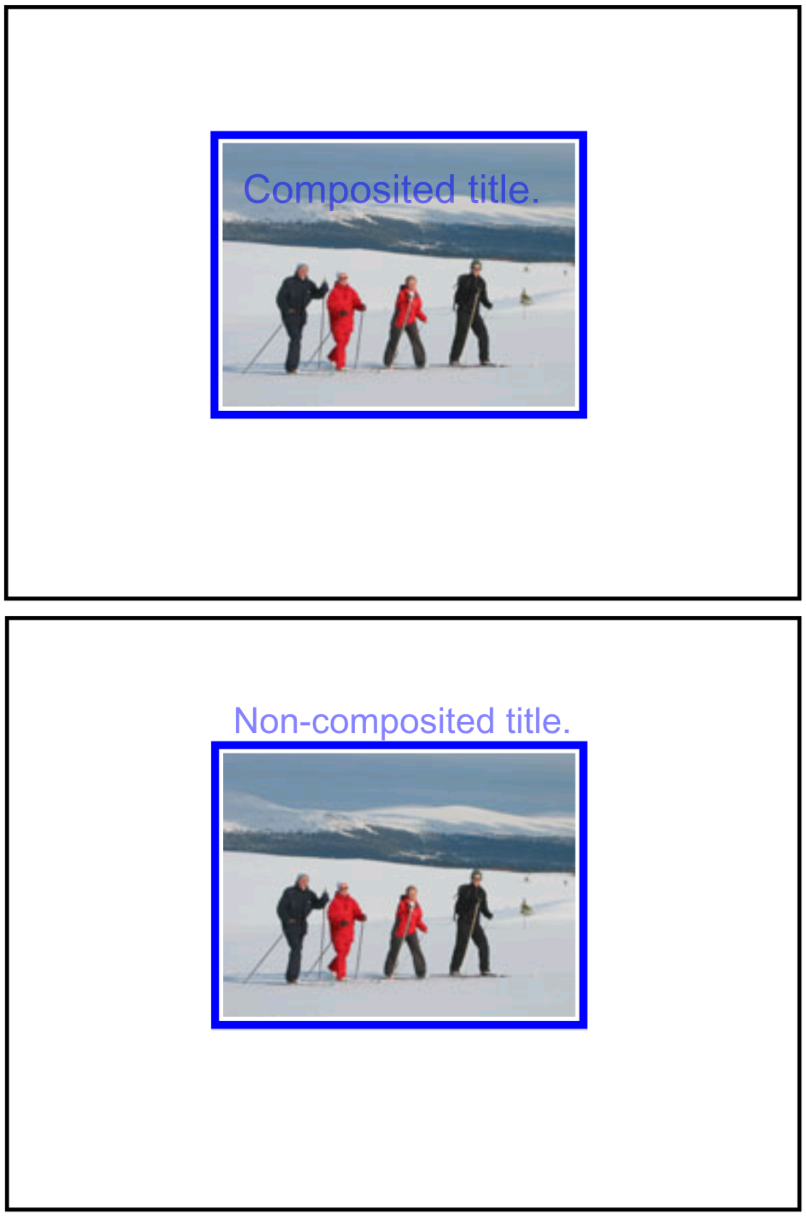
The above images show the rendering of Example media04 in two SVG user agents: the first one supporting transformed video (on the left) and the second one not (on the right).

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/Graphics/SVG/feature/1.2/#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/Graphics/SVG/feature/1.2/#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>

        <!-- Overlayed 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>
```



The above images show the rendering of Example media05 in two SVG user agents: the first one supporting composed video (on the left) and the second one not (on the right).

12.4 The **'animation'** element

The **'animation'** elements 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 synchronization attributes which allow 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 **'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 Processing of external references to 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. The lacuna value is **'0'**.

Animatable: yes.

y = **"<coordinate>"**

The *y*-axis coordinate of one corner of the rectangular region into which the animation is placed. The lacuna value is **'0'**.

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. The lacuna value is **'0'**.

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. The lacuna value is **'0'**.

Animatable: yes.

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

An IRI reference to the animation data. An invalid IRI reference is an unsupported value. An empty attribute value (**xlink:href=""**) disables rendering of the element. The lacuna value is the empty string.

When the value of this attribute is animated or otherwise modified, if the media timeline can be controlled, then the media timeline is restarted only if the **'syncBehavior'** attribute is set to **independent**. If the media timeline cannot be controlled, then the media timeline is unaffected by such modification.

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.

Runtime synchronization attributes

See definition.

SVG timing attributes

See definition.

The example below shows basic usage of the **'animation'** element. For another example, see the use and animation example in the Linking chapter.

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, the **'use'** element, plus container elements such as the **'g'** element.

'audio-level'

Value:	<number> | inherit
Initial:	1.0
Applies to:	media elements, **'use'** and container elements
Inherited:	no
Percentages:	N/A
Media:	visual, audio
Animatable:	yes
Computed value:	Specified value, except inherit

The **'audio-level'** property specifies a value that is used to calculate the volume of a particular element. Values below 1.0 decrease it and a value of zero silences it.

An element's volume is the product of its clamped **'audio-level'** property and either the clamped computed value of its parent, or the initial value (1.0) if it has no parent. Audio level clamping occurs for any values outside the range 0.0 (silent) to 1.0 (system volume). (See Clamping values which are restricted to a particular range.)

This sentence is informative: An element's volume cannot be louder than the volume of its parent.

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)
```

User agents may limit the actual signal level to some maximum, based on user preferences and hardware limitations.

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 runtime synchronization

SVG Tiny 1.2 supports the five attributes listed below from SMIL 2.1 to control runtime synchronization of timed elements. In SVG Tiny 1.2 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.

Attribute definitions:

`syncBehavior` = **"canSlip" | "locked" | "independent" | "default"**
  See the SMIL 2.1 definition of 'syncBehavior' ([SMIL21], section 10.4.1).
  Animatable: no.

`syncBehaviorDefault` = **"canSlip" | "locked" | "independent" | "inherit"**
  See the SMIL 2.1 definition of 'syncBehaviorDefault' ([SMIL21], section 10.4.1).
  Animatable: no.

`syncTolerance` = **"<Clock-value>" | "default"**

See the SMIL 2.1 definition of 'syncTolerance' ([SMIL21], section 10.4.1).

Animatable: no.

`syncToleranceDefault` = **"<Clock-value>" | "inherit"**

See the SMIL 2.1 definition of 'syncToleranceDefault' ([SMIL21], section 10.4.1).

Animatable: no.

`syncMaster` = **"<boolean>"**

See the SMIL 2.1 definition of 'syncMaster' ([SMIL21], section 10.4.1).

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 synchronization attributes.

Example: sync-attr-main.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"
     viewBox="0 0 400 100" height="100%" width="100%" syncBehaviorDefault="locked">

  <title>Sync* Attributes</title>
  <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 xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny"
     viewBox="0 0 400 100" height="100%" width="100%">

  <title>Synchronizing lyrics with video</title>
  <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 xml:id="Text0" display="none">This is some text</text>
    <set xlink:href="#Text0" attributeName="display" to="inline" begin="0" end="1"/>
    <text xml:id="Text10" display="none">simulating some lyrics</text>
    <set xlink:href="#Text10" attributeName="display" to="inline" begin="1.1" end="2"/>
    <text xml:id="Text20" display="none">displayed synchronously</text>
    <set xlink:href="#Text20" attributeName="display" to="inline" begin="2.1" end="3"/>
    <text xml:id="Text30" display="none">with some video</text>
    <set xlink:href="#Text30" attributeName="display" to="inline" begin="3.1" end="4"/>
    <text xml: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 attribute has the value **'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 lacuna value. Indicates that the media object is not displayed, as though the element had **display="none"**, until its first active duration starts.

always

The media element is visible from the initialization of the parent time container (i.e. time 0 of the parent SVG document). During this time, and until the active duration starts, the media element is initialized but remains inactive.

Animatable: no.

13 Interactivity

Contents

13.1 Introduction . . . 173
13.2 Complete list of supported events . . . 173
13.3 User interface events . . . 177
13.4 Pointer events . . . 178
13.5 Text events . . . 178
13.6 Key events . . . 178
13.7 Event flow . . . 179
13.8 Event dispatching . . . 179
13.9 Processing order for user interface events . . . 179
13.10 The 'pointer-events' property . . . 180
13.11 Magnification and panning . . . 182
13.12 Element focus . . . 182
    13.12.1 The 'focusable' attribute . . . 182
13.13 Navigation . . . 183
    13.13.1 Navigation behavior . . . 183
    13.13.2 Specifying navigation . . . 184
    13.13.3 Specifying focus highlighting . . . 187
    13.13.4 Obtaining and listening to focus programmatically . . . 188

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 timed elements to start or stop, scripts to execute or **'listener'** elements to trigger **'handler'** elements.
- The user can initiate hyperlinks to new Web pages (see 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 SVG 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 Linking
- **'script'** and **'handler'** elements are discussed in Scripting
- timed elements are discussed in Animation and Multimedia chapters

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 Type	Description	Animation event name	Bubbles	Cancelable	uDOM interface

DOMFocusIn	Occurs when an element receives focus. See the DOM 2 Events definition of DOMFocusIn ([DOM2EVENTS], section 1.6.1).	focusin	Yes	No	`UIEvent`
DOMFocusOut	Occurs when an element loses focus. See the DOM 2 Events definition of DOMFocusOut ([DOM2EVENTS], section 1.6.1).	focusout	Yes	No	`UIEvent`
DOMActivate	Occurs when an element is activated, for instance, through a mouse click or a keypress. See the DOM 2 Events definition of DOMActivate ([DOM2EVENTS], section 1.6.1).	activate	Yes	Yes	`UIEvent`
click	Occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: `mousedown`, `mouseup`, `click`. If multiple clicks occur at the same screen location, the sequence repeats with the `detail` attribute incrementing with each repetition. See the DOM 2 Events definition of click ([DOM2EVENTS], section 1.6.2).	click	Yes	Yes	`MouseEvent`
mousedown	Occurs when the pointing device button is pressed over an element. See the DOM 2 Events definition of mousedown ([DOM2EVENTS], section 1.6.2).	mousedown	Yes	Yes	`MouseEvent`
mouseup	Occurs when the pointing device button is released over an element. See the DOM 2 Events definition of mouseup ([DOM2EVENTS], section 1.6.2).	mouseup	Yes	Yes	`MouseEvent`
mouseover	Occurs when the pointing device is moved onto an element. See the DOM 2 Events definition of mouseover ([DOM2EVENTS], section 1.6.2).	mouseover	Yes	Yes	`MouseEvent`
mousemove	Occurs when the pointing device is moved while it is over an element. See the DOM 2 Events definition of mousemove ([DOM2EVENTS], section 1.6.2).	mousemove	Yes	Yes	`MouseEvent`

mouseout	Occurs when the pointing device is moved away from an element. See the DOM 2 Events definition of mouseout ([DOM2EVENTS], section 1.6.2).	mouseout	Yes	Yes	`MouseEvent`
mousewheel	Occurs when a rotational input device has been activated. See the description of the `MouseWheelEvent` event for details.	*none*	Yes	Yes	`MouseWheelEvent`
textInput	One or more characters have been entered. See the Text events section below for details.	*none*	Yes	Yes	`TextEvent`
keydown	A key is pressed down. See the Key events section below for details.	*none*	Yes	Yes	`KeyboardEvent`
keyup	A key is released. See the Key events section below for details.	*none*	Yes	Yes	`KeyboardEvent`
load	The event is triggered at the point at which the user agent finishes loading the element and any dependent resources (such as images, style sheets, or scripts). In the case the element references a script, the event will be raised only after an attempt to interpret the script has been made. Dependent resources that fail to load will not prevent this event from firing if the element that referenced them is still in the document tree unless they are designated as externalResourcesRequired. The event is independent of the means by which the element was added to DOM tree.	load	No	No	`Event`
SVGLoad	This event is deprecated and is for backwards compatibility only, see notes below. The This event must be dispatched immediately after the load event is dispatched.	*none*	No	No	`Event`
resize	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.	resize	Yes	No	`Event`

SVGResize	This event is deprecated and is for backwards compatibility only, see notes below. This event must be dispatched immediately after the resize event is dispatched.	*none*	Yes	No	`Event`
scroll	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.	scroll	Yes	No	`Event`
SVGScroll	This event is deprecated and is for backwards compatibility only, see notes below. This event must be dispatched immediately after the scroll event is dispatched.	*none*	Yes	No	`Event`
SVGZoom	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.	zoom	No	No	`Event`
SVGRotate	Occurs when the rotation of a document view is being changed, either through a direct user interaction or any change to the `currentRotate` property available on `SVGSVGElement` interface. This event is only applicable to **'svg'** elements and is dispatched after the rotation modification has taken place. The target of the event is the **'svg'** element.	rotate	No	No	`Event`
beginEvent	Occurs when a timed element begins. See the SMIL 2.1 definition of beginEvent ([DOM2EVENTS], section 10.6.2).	beginEvent	Yes	No	`TimeEvent`
endEvent	Occurs when a timed element ends. See the SMIL 2.1 definition of endEvent ([DOM2EVENTS], section 10.6.2).	endEvent	Yes	No	`TimeEvent`
repeatEvent	Occurs when a timed element repeats. It is raised each time the element repeats, after the first iteration.	repeatEvent	Yes	No	`TimeEvent`

	See the SMIL 2.1 definition of repeatEvent ([DOM2EVENTS], section 10.6.2).				
loadstart	A load operation has begun. See the description of the `ProgressEvent` interface for details on this event.	*none*	No	No	`ProgressEvent`
progress	Progress has occurred in loading a given resource. See the description of the `ProgressEvent` interface for details on this event.	*none*	No	No	`ProgressEvent`
loadend	A load operation has completed. See the description of the `ProgressEvent` interface for details on this event.	*none*	No	No	`ProgressEvent`
SVGTimer	Occurs when the specified timer interval has elapsed for a timer. This event is triggered only by 'running' 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). The target of the event is the `SVGTimer` object itself. The event processing is limited to the at target phase. See the description of the `SVGTimer` interface for more details.	*none*	No	No	`Event`

Note that in order to unify event names with other W3C specifications, SVG Tiny 1.2 deprecates some of the SVG 1.1 event types. (The term "deprecate" in this case means that user agents which are compatible with both SVG 1.1 and SVG Tiny 1.2 must support both the old deprecated event names and the new event names. Content creators who are making content that targets SVG Tiny 1.2 should use the new event types, not the deprecated event types.) Specifically:

- The `"SVGLoad"` event is deprecated in favor of `"load"`
- The `"SVGResize"` event is deprecated in favor of `"resize"`
- The `"SVGScroll"` event is deprecated in favor of `"scroll"`

Details on the values of attributes on the event object passed to event listeners for the event types defined in DOM Level 2 Events can be found in the description for that event in that specification. For other event types, the values of the attributes are are described elsewhere in this specification.

13.3 User interface events

On SVG 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 a `DOMActivate` event to trigger an ECMAScript event handler:

Example: activate.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     width="6cm" height="5cm" viewBox="0 0 600 500">

  <desc>Example: invoke an ECMAScript function from a DOMActivate 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>

  <!-- Act on each DOMActivate event -->
  <circle cx="300" cy="225" r="100" fill="red">
    <handler type="application/ecmascript" ev:event="DOMActivate"> 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>
```

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 joypad. 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 `mouseover` and `mouseout` 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. The event is dispatched whenever a string of Unicode characters is sent to the document and is thus independent of the input device or method 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 flow

DOM Level 2 Events defines the event flow model ([DOM2EVENTS], section 1.2), which defines three phases in which event listeners in the document are triggered: *capture*, *at target* and *bubbling*. An SVG Tiny 1.2 user agent is not required to support the capture phase of the event flow model. If the capture phase is not supported:

- Registering an event listener for the capture phase by passing `true` for the useCapture parameter of `EventTarget::addEventListener()` will result in that listener never being triggered. Since there is no way with the SVG uDOM to determine whether a listener has been registered on a node or not, such calls to `EventTarget::addEventListener()` can be ignored.
- Registering an event listener for the capture phase by specifying **phase="capture"** on a **'listener'** will result in an event listener being registered for the at target and default phases, since a value of **'capture'** will be ignored, resulting in the lacuna value of **'default'** being used. Conforming SVG documents must use **'default'** as the value of the **'phase'** attribute if it is specified.
- Any `keydown` event that corresponds to an **accessKey-value** in an animation timing specifier list will never cause any appropriate listeners to be triggered, since, as described in the definition of the **accessKey-value** syntax, the SVG user agent behaves as if `stopPropagation()` and `preventDefault()` had been invoked on the event object in the capture phase.

13.8 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 graphics 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.

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.

If an event is defined to bubble ([DOM2EVENTS], section 1.2.3), 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.

13.9 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.

The **'use'** element creates shadow content which can be the target of user interface events.

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.

13.10 The **'pointer-events'** property

In different circumstances, authors may want to control under what circumstances particular graphics element 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 graphics 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 **'rect'** 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 **'rect'** 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 a key press
- hyperlinks (see the **'a'** element)

'pointer-events'

Value:	boundingBox | 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

boundingBox
: The given element must be a target element for pointer events when the pointer is over the bounding box of the element.

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 basic (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.

13.11 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.

Attribute definition:

`zoomAndPan` = **"magnify" | "disable"**

Can be specified on the **'svg'** element. The attribute is intended for applications where SVG is used for both the content and for the user interface, e.g. a mapping application. The default zoom might move critical user interface components from view, confusing the user; disabling the default zoom, pan and rotate while providing zoom, pan and rotate controls for a smaller content area would give a better user experience. The effect of **'zoomAndPan'** applies solely to user interface aspects, and must not disable script-initiated zooming and panning on the corresponding element.

The attribute value can be one of the following:

magnify

The lacuna value. If *magnify*, in environments that support user interactivity, the user agent must provide controls to allow the user to perform a "magnify" operation on the document fragment.

disable

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 SVG user agent may provide another mode which continues to allow zoom and pan at user option.

Animatable: no.

13.12 Element focus

13.12.1 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 one 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 lacuna value. Equivalent to **'false'**, except that it must be treated like **'true'** for the following cases:

- The **'a'** element.
- Text content block elements with **'editable'** set to **'simple'**.

- Elements that are the target of an animation whose begin or end lists include an eventbase timing specifier triggered by the following user interface events: `DOMFocusIn`, `DOMFocusOut`, `DOMActivate`.
- Elements that have an event listener registered on one of the following user interface events: `DOMFocusIn`, `DOMFocusOut`, `DOMActivate`.

  Informative note: Event listeners for the listed events can be added to elements that are the **'target'** or **'observer'** of a **'listener'** element, the parent element of a **'handler'** element if it has an **'ev:event'** attribute as well as by using script.

Animatable: yes.

13.13 Navigation

13.13.1 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 is **'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 xml:id="r1" focusable="true" .../>
<g xml:id="g1" focusable="true">
  <circle xml: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 SVG user agent must not navigate to an element which has **display="none"**. (An element which has **display="none"** is not focusable.)
- The SVG user agent must allow navigation to elements which are not visible (i.e. which has a 100% transparency or which is hidden by another element).
- The SVG user agent must allow navigation to elements which are located outside of the current viewport. In this case it is recommended that the SVG user agent 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 SVG user agent.
2. Once the SVG 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 SVG 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 SVG 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 SVG user agent

For stand-alone SVG documents, the SVG user agent must always have a currently focused object. If focus is not held by any object in the document tree, the SVG 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.

User agents should provide a mechanism for a user to escape from a focus ring. When the user activates this mechanism, the user agent should change focus to the user agent, sending the appropriate `focusout` event to the element currently in focus.

13.13.2 Specifying navigation

Navigation order can be specified using the ten navigation attributes defined below.
Attribute definitions:

`nav-next`,
`nav-prev` = **"<FuncIRI>" | "auto" | "self"**

Specifies the next element (when using **'nav-next'**) or previous element (when using **'nav-prev'**) in the focus ring.

The attribute value for **'nav-next'** and **'nav-prev'** can be one of the following:

<FuncIRI>

Specifies the element that must receive focus when navigation in the next direction (for **'nav-next'**) or previous direction (for **'nav-prev'**) is triggered. The specified element must be within the current SVG document fragment.

auto

The lacuna 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`,
`nav-up-right`,
`nav-right`,
`nav-down-right`,
`nav-down`,
`nav-down-left`,
`nav-left`,
`nav-up-left` = **"<FuncIRI>" | "auto" | "self"**

Each of these eight attributes specifies an element to receive focus when navigating in a particular direction. For each of the attributes, the direction for which an element is being specified for navigation is suggested by the name of the attribute. The following table lists these directions explicitly:

Attribute name	Direction
'nav-up'	↑ upward
'nav-up-right'	↗ up-and-rightward
'nav-right'	→ rightward

'nav-down-right'	↘ down-and-rightward
'nav-down'	↓ downward
'nav-down-left'	↙ down-and-leftward
'nav-left'	← leftward
'nav-up-left'	↖ up-and-leftward

The value for each of these attributes can be one of the following:

<FuncIRI>
Specifies the element that must receive focus when navigation in the given direction is triggered. The specified element must be within the current SVG document fragment.

auto
The lacuna value. Means that the behavior is left up to the SVG user agent.

self
The focus must stay on the element itself.

Animatable: yes.

Example: navigation.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny"
     viewBox="0 0 220 40">

  <title>Media Channel Navigation User Interface</title>
  <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.

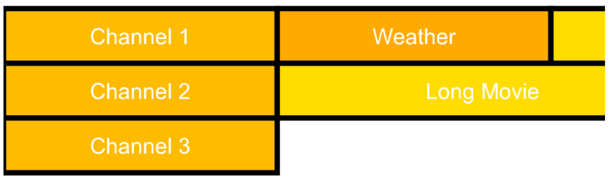
  In order to apply option 3, we could have set **nav-prev="self"** instead.
- Whenever the focus is located on a program which is at the end of the timeline of a given channel, there are 2 options when the user wants to go to the next focusable item (i.e., the user presses the "Tab" key on most desktop 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.

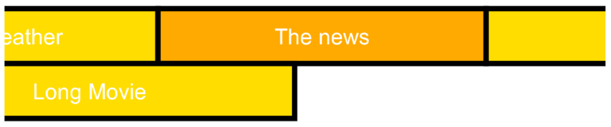
  In order to apply option 1, we could have set **nav-next="url(#Chan2Prog1)"** instead.
- Whenever the focus is located on `"Chan2Prog1"` container, if the user wants to go to the next focusable element, the concept of a focus ring will apply because of value **nav-next="auto"**. Here, according to the focus ring navigation rules, focus will be offered to the SVG 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 SVG user agent 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 navigation 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 SVG user agent to choose which focusable element should receive focus when the user wants to go 'right'.

13.13.3 Specifying focus highlighting

Automated highlighting upon focus can be specified using the **'focusHighlight'** attribute. This hint indicates whether the SVG user agent should highlight an element on focus. The highlighting method is implementation dependent and the SVG 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 SVG user agent should highlight an element on focus.

The attribute value can be one of the following:

auto

The lacuna value. This indicates that the element should be highlighted on focus. The highlighting method is left up to the SVG user agent.

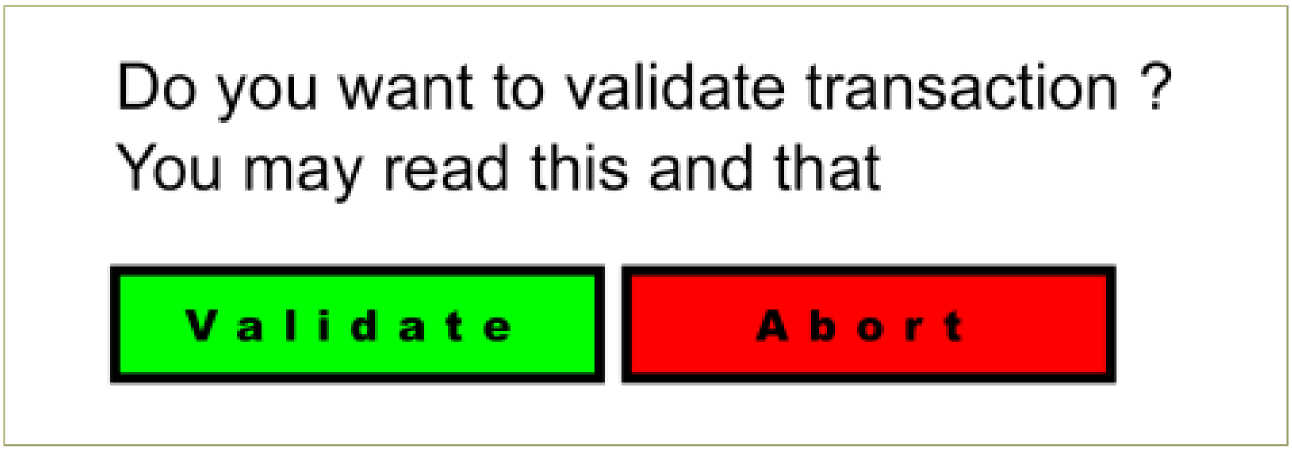
none

The SVG user agent should not highlight this element on focus.

Animatable: no.

Example: focusHighlight.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" viewBox="0 0 210 80">

  <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"
          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
Validate
Abort

In the above SVG example:

- Highlight of the focus on the first two textual links is left up to the SVG user agent (underline the text, highlight of the bounding box, change color of the text, ...) since the lacuna 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 two graphical buttons is designed by the author and therefore the SVG user agent doesn't need to highlight it as well. Therefore, **focusHighlight="none"** is used to disable the default focus highlight behavior.

13.13.4 Obtaining and listening to focus programmatically

When the user agent gives an element focus it receives a `DOMFocusIn` event which has the new focused object as the event target and a `DOMFocusOut` event which has the previously focused object as the event target.

The `SVGSVGElement` interface has a `setFocus` 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`.

SVG 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. SVG 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 navigation 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="myAnimationHandler" type="application/ecmascript"><![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 190
14.1.1 Overview 190
14.1.2 IRIs and URIs 190
14.1.3 Syntactic forms: IRI and FuncIRI 191
14.1.4 Reference restrictions 191
14.1.5 IRI reference attributes 194
14.1.6 Processing of external references to documents 199

14.2 Links out of SVG content: the 'a' element 200
14.2.1 Indicating links 202

14.3 Linking into SVG content: IRI fragments and SVG views 203
14.3.1 Introduction: IRI fragments and SVG views 203
14.3.2 SVG fragment identifiers 203

14.1 References

14.1.1 Overview

On the Internet, resources are identified using IRIs (Internationalized Resource Identifiers). For example, an SVG file called someDrawing.svg located at http://example.com might have the following IRI:

```
http://example.com/someDrawing.svg
```

An IRI can also address a particular element within an XML document by including an IRI fragment identifier as part of the IRI. An IRI which includes an IRI fragment identifier consists of an optional base IRI, followed by a "#" character, followed by the IRI fragment identifier. For example, the following IRI can be used to specify the element whose ID is "Lamppost" within file someDrawing.svg:

```
http://example.com/someDrawing.svg#Lamppost
```

Altering the 'xlink:href' attribute

If the **'xlink:href'** attribute of an element in the tree is altered by any means (e.g. script, declarative animation) such that a new resource is referenced, the new resource must replace the existing resource, and must be rendered as appropriate. For specific effects on the scripting context when a **'script'** element's **'xlink:href'** attribute is altered, see Script processing.

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 [UNICODE]. A URI is constructed from a much more restricted set of characters. All URIs are already conformant IRIs. A mapping from IRIs to URIs is defined by the IRI specification, which means that IRIs can be used instead of URIs in XML documents, to identify resources. IRIs can be converted to URIs for resolution on a network, if the protocol does not support IRIs directly.

Previous versions of SVG, following XLink, defined 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 1.0 specification [XLINK10], and in the W3C XML Schema Part 2: Datatypes specification [SCHEMA2]. This copying introduced the possibility of error and divergence, but was done because the IRI specification was not yet standardized.

In this specification, the correct term IRI is used for this "URI or sequence of characters plus an algorithm" and the escaping method, which turns IRIs into URIs, is defined by reference to the IRI specification [RFC3987], which has since become an IETF Proposed Standard. Other W3C specifications are expected to be revised over time to remove these duplicate descriptions of the escaping procedure and to refer to IRI directly.

14.1.3 Syntactic forms: IRI and FuncIRI

IRIs are used in the **'xlink:href'** attribute. Some attributes allow both IRIs and text strings as content. To disambiguate a text string from a relative IRI, the functional notation <FuncIRI> is used. This is simply an IRI delimited with a functional notation. **Note:** For historical reasons, the delimiters are "url(" and ")", for compatibility with the CSS specifications. The FuncIRI form is used in presentation attributes and navigation attributes.

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

Example: 05_07.xml

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

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

Example: 05_08.xml

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

14.1.4 Reference restrictions

Some of the elements using IRI references have restrictions on them. Which kinds of IRI references that are allowed on each element is listed in the table below. In SVG, IRI references can be categorized as being one (or more) of the following five types:

- **A:** A reference to a fragment within the current document (e.g. **'#someelement'**). If the referenced fragment is not within the current SVG document fragment, then whether the reference is an invalid IRI reference or not is defined by the host language.
- **B:** A reference to a fragment within an external document (e.g. **'afile.svg#anelement'**).
- **C:** A reference to an entire SVG document (e.g. **'afile.svg'**).
- **D:** A reference to a media resource other than SVG, with or without the use fragments (e.g. **'someimage.jpg'** or **'somecontainer#fragment'**). Where applicable, the table shows the supported media types.
- **E:** A `data:` IRI (e.g. **'data:image/jpeg;base64,/9j...'**) [RFC2397]. Note that `data:` IRIs, if XML, resolve to a document that is distinct from the referencing element's owner document, however the data is already loaded as it is part of the IRI itself.

For each of the above five IRI types, A – E, there is a column in the reference restriction table below indicating whether the given attribute is allowed to have a reference of the given form. An IRI reference that does not comply to the restrictions in the table below is an invalid IRI reference.

Element	Referencing attribute	A	B	C	D	E
An animation element	'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

Element	Referencing attribute	A	B	C	D	E
'a'	'xlink:href'	Yes, see Linking into SVG content.	Yes, see Links out of SVG content.	Yes	Yes	Yes
'a'	'xlink:role' 'xlink:arcrole'	Yes	Yes	Yes	Yes	Yes
'use'	'xlink:href'	Yes, but a 'use' element must not reference an 'svg' element.	Yes, but the referenced fragment must not contain scripting, hyperlinking to animations or any externally referenced 'use' or 'animation' elements.	No	No	No
'image'	'xlink:href'	No	No	No	Yes, but the 'image' element must reference content that is a raster image format.	Yes, but the content within the `data:` IRI reference must be a raster image.
'animation'	'xlink:href'	No	No	Yes	No	Yes
'prefetch'	'xlink:href'	Yes	Yes	Yes	Yes	No
'audio'	'xlink:href'	No	No	No	Yes, depending on supported audio formats, indicated by the 'type' attribute.	Yes
'video'	'xlink:href'	No	No	No	Yes, depending on supported video formats, indicated by the 'type' attribute.	Yes
'foreignObject'	'xlink:href'	No	Yes	No	Yes	Yes
'script'	'xlink:href'	No	No	No	Yes, but it must reference an external resource that provides the script content.	Yes
'handler'	'xlink:href'	Yes	Yes	No	Yes, but it must reference an external resource that provides the script content.	Yes
'listener'	'handler'	Yes	No	No	No	No
An element on which	'fill'	Yes, only referencing a	No	No	No	No

Element	Referencing attribute	A	B	C	D	E
paint may be specified		paint server, see Specifying paint.				
An element on which paint may be specified	**'stroke'**	Yes, only referencing a paint server, see Specifying paint.	No	No	No	No
An element on which navigation attributes may be specified	A navigation attribute	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

Additionally, any IRI reference which cannot be resolved is an invalid IRI reference. Examples of reasons for an IRI reference to be unable to be resolved include:

- The resource is an external resource and is not available (for example, the user agent cannot connect to the location on the network which stores the resource, and the resource is not cached locally).
- The IRI reference is to a local element that does not exist (for example, a **'use'** element whose **'xlink:href'** references a non-existent element).
- The IRI reference is to a resource that does not exist (for example, an **'image'** element that references an HTTP resource that results in a 404 response code, even if the response body contains an otherwise supported raster image resource).

Any required processing for an attribute with an invalid IRI reference is described in the attribute definition. Note that when the **'externalResourcesRequired'** attribute has been set to **'true'** on the referencing element or one of its ancestors, then an unresolved external IRI reference will result in special handling (see External resources).

A circular IRI reference is an error. Because SVG user agents may vary on when they first detect and abort a circular reference, conforming SVG document fragments must not rely upon circular references. Examples of circular references include:

- A **'use'** element that directly or indirectly references itself, as in the following SVG document fragment:

```
<svg xmlns='http://www.w3.org/2000/svg'
     xmlns:xlink='http://www.w3.org/1999/xlink'
     version='1.2' baseProfile='tiny'>

  <title>Example of a circular reference with 'use'</title>

  <g id='a'>
    <text>ABC</text>
    <use xlink:href='#b'/>
  </g>
  <g id='b'>
    <text>DEF</text>
    <use xlink:href='#a'/>
  </g>
</svg>
```

- An **'animation'** element that directly or indirectly references the document that contains the current SVG document fragment, as in the following example:

```
<svg xmlns='http://www.w3.org/2000/svg'
     xmlns:xlink='http://www.w3.org/1999/xlink'
     version='1.2' baseProfile='tiny'>

  <title>Example of a circular reference with 'animation'</title>

  <animation xlink:href='#' width='100' height='100'/>
</svg>
```

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.

14.1.5 IRI reference attributes

IRI references are normally specified with an **'href'** attribute in the XLink [XLink] namespace. For example, if the prefix of 'xlink' is used for attributes in the XLink namespace, then the attribute is be specified as **'xlink:href'**. The value of this attribute forms a reference for the desired resource (or secondary resource, if there is a fragment identifier).

The value of the **'href'** attribute must be an Internationalized Resource Identifier.

If the protocol, such as HTTP, does not support IRIs directly, the IRI is converted to a URI by the SVG implementation, as described in section 3.1 of the IRI specification [RFC3987].

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

If the IRI reference is relative, its absolute version must be computed by the method described in XML Base before use [XML-BASE].

Additional XLink attributes can be specified that provide supplemental information regarding the referenced resource.

Schema: xlinkattr

```
    <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'>
      <optional>
        <attribute name='xlink:href' svg:animatable='true' svg:inheritable='false'>
          <ref name='IRI.datatype'/>
        </attribute>
      </optional>
    </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' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>new</value>
        <value>replace</value>
      </choice>
    </attribute>
  </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. In line with the changes proposed in XLink 1.1 [XLINK11], this attribute may be omitted on simple links. 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) [XLINK10].

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) [XLINK10].
  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) [XLINK10].
  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) [XLINK10].
  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) [XLINK10].
  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) [XLINK10].
  Animatable: no.

In all cases, for compliance with either the "Namespaces in XML 1.0" or the "Namespaces in XML 1.1" Recommendation [XML-NS10][XML-NS], an explicit XLink namespace declaration must be provided whenever one of the above XLink attributes is used within SVG content. One simple way to provide such an XLink namespace declaration is to include an **'xmlns'** attribute for the XLink namespace on the **'svg'** element for content that uses XLink attributes.

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"
     xml:id="animationRef" width="150" height="50" viewBox="0 0 150 50" fill="inherit">

    <rect xml: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

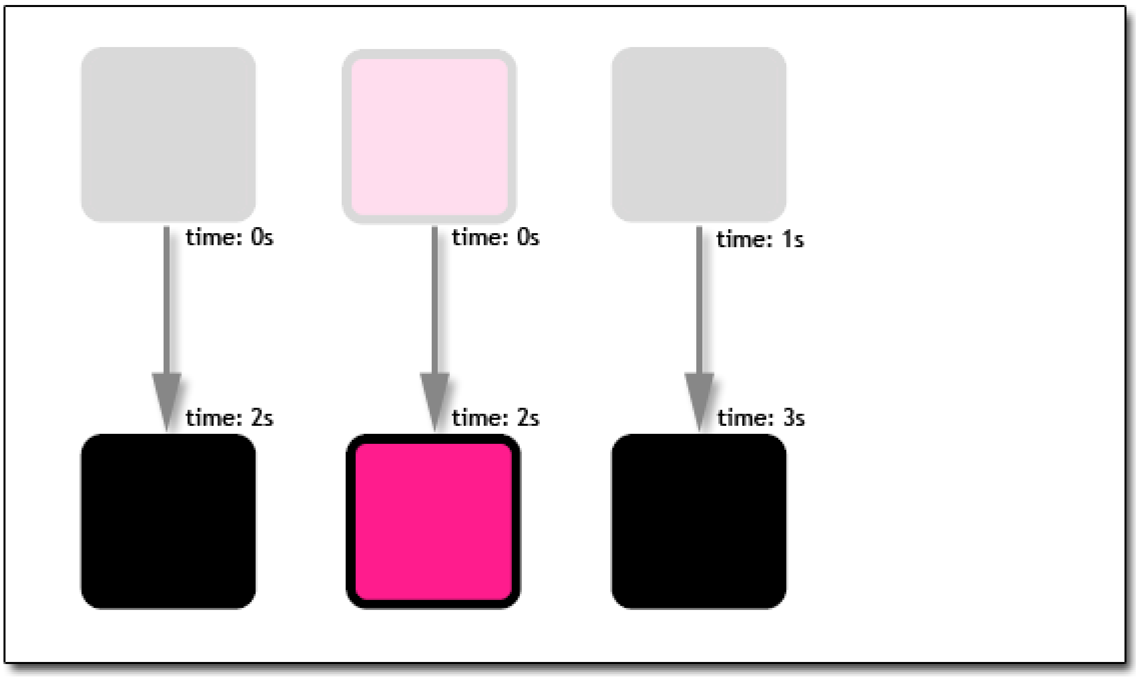
```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny"
     xml:id="animationRef" width="150" height="50" viewBox="0 0 150 50" fill="inherit">

    <rect xml: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).

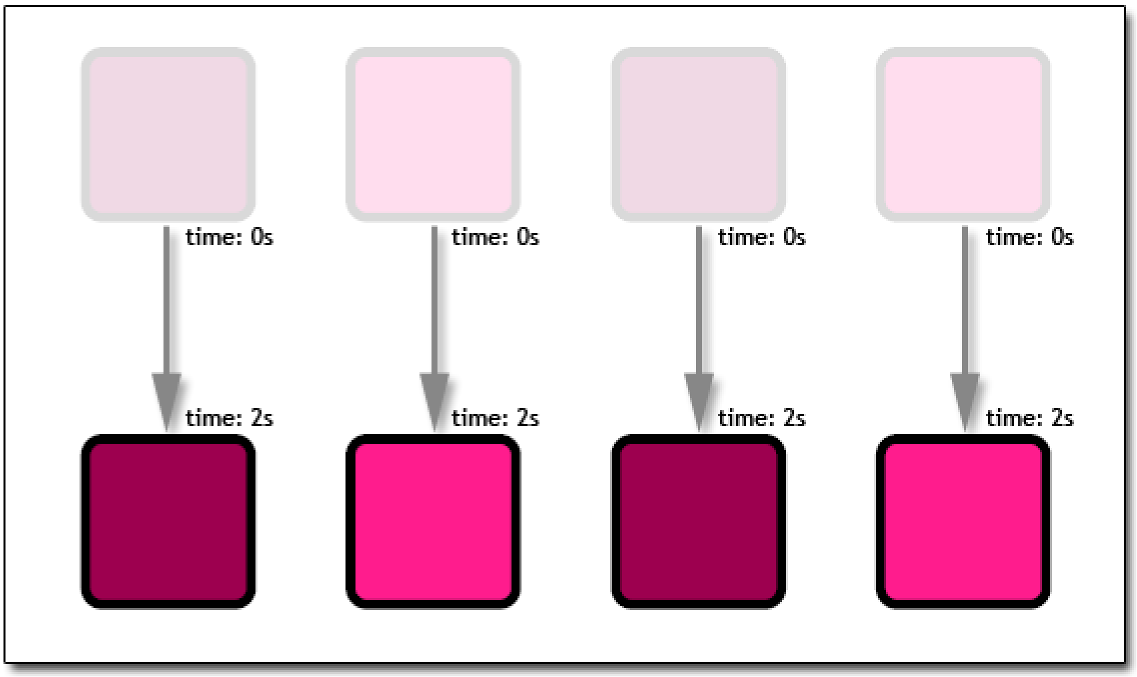


The following example illustrates the different ways SVG content can be referenced from a **'use'** element. Different **'fill'** values are used to show the way properties are inherited on content referenced from the **'use'** element.

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 xml: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).



14.1.6 Processing of external references to documents

When an SVG user agent resolves an external reference to a document, how the document is loaded and processed depends on how the document was referenced. As defined below, a document is classified as either a *primary document* or a *resource document*, and this classification determines the document's processing with respect to loading of external references.

A primary document is one that is to be presented in whole by the user agent. Specifically, the following are classified as primary documents:

- An entire document, be it an SVG stand-alone document or some other document that can contain SVG document fragments, that is loaded into a user agent for presentation, such as when navigating a web browser to an IRI, whether by typing the IRI into the browser's address bar, clicking on a link to that IRI, or having the `Location::assign()` method invoked. (In an HTML 5 user agent, this is when a document is part of a top-level browsing context ([HTML5], section 4.1.1).)
- An entire SVG document that is loaded due to it being referenced by an **'animation'** element.
- A document that is loaded due to it being referenced for inclusion by a parent non-SVG document for presentation, such as using the HTML **'object'** or **'iframe'** elements.

A resource document is a document that has been loaded because parts of it are referenced as resources by an SVG document fragment. Specifically, the following kinds of external references, all of which are references to elements, will cause the loaded document to be classified as a resource document:

- The **'xlink:href'** attribute on a **'use'** element.
- The **'xlink:href'** attribute on a **'font-face-uri'** element.
- A paint server reference in a **'fill'** or **'stroke'** property.

Note that neither a primary document nor a resource document need be a complete SVG document (with the rootmost 'svg' element being the document element). Both may be non-SVG documents that contain SVG document fragments.

Each primary document maintains a dictionary that maps IRIs to resource documents. This dictionary is used whenever a resource document is to be loaded because an SVG document fragment within the primary document (or one of its resource documents) references it. Before loading a resource document, its IRI is first looked up in the primary document's dictionary to determine if it has already been loaded. If so, then that already-loaded document is used instead of creating a separate document instance. Thus, for each primary document, a given resource document is loaded only once. Primary documents, however, are always separate, self-contained document instances, and resource documents are not shared between different primary documents.

The IRI used as the key in the dictionary of resource documents must be the absolute IRI after resolving it against any applicable base IRI, and comparisons of the dictionary keys must be performed using a Simple String Comparison, as defined in section 5.3.1 of *Internationalized Resource Identifiers* [RFC3987].

Whether a document is a primary document or a resource document, its processing once loaded is the same: each SVG document fragment within the document acts as a separate SVG processing context in which events are fired, scripts are executed, an animation timeline is created and animations are run, stylesheets are applied (if supported by the SVG user agent), and so on. Since a resource document is not just a static DOM, any changes to it (be they modifications by script or changing presentation values with animation) will be visible through all references to that resource document.

Note that since IRI references to resources from different primary documents will result in logically separate resource documents being instantiated, an SVG user agent will in general not be able to conserve memory by having only one instance of the resource document in memory. In the case that many primary documents all have references to a single, large, common resource file, this will likely result in a large amount of memory consumed. If the SVG user agent is able to prove that the primary documents will behave exactly the same if a single instance is shared in memory (by using copy-on-write semantics for the resource documents, for example), then such an optimization may of course be performed.

References to any other kinds of document, such as media or external scripts, are not classified as primary or resource documents. Multiple references to media at a particular IRI always result in separate timelines being created.

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 [XLINK10] 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).

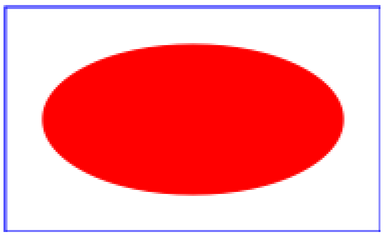
The remote resource (the destination for the link) is defined by an IRI specified by the XLink **'href'** attribute on the **'a'** element. The remote resource may be any Web resource (e.g., an image, a video clip, a sound bite, a program, another SVG document, an HTML document, etc.). By activating these links (by clicking with the mouse, through keyboard input, voice commands, etc.), users may traverse hyperlinks to these resources.

If the IRI identifies an animation element within the current SVG document fragment, then activating the **'a'** element will hyperlink to the animation, as defined in SMIL 2.1 ([SMIL21], section 10.4.3).

Example 17_01 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">

  <title>Example 17_01</title>
  <desc>A simple 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.

The element definition schema and content model for **'a'** is not defined here. It is defined in all the places it can occur.

Schema: a.at

```
<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) [XLINK10].

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) [XLINK10].

Animatable: no.

`xlink:href` = **"<IRI>"**

The location of the referenced object, expressed as an IRI reference.

Animatable: yes.

`target` = **"_replace" | "_self" | "_parent" | "_top" | "_blank" | "<XML-Name>"**

This attribute should be used when there are multiple possible targets for the ending resource, such as when the parent document is a multi-frame HTML or XHTML document. This attribute specifies the name or portion of the target window, frame, pane, tab, or other relevant presentation context (e.g., an HTML or XHTML frame, iframe, or object element) into which a document is to be opened when the link is activated. 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.

_self
: The current SVG image is replaced by the linked content in the same frame as the current SVG image. This is the lacuna value, if the target attribute is not specified.

_parent
: The immediate frameset parent of the SVG image is replaced by the linked content.

_top
: The content of the full window or tab, including any frames, is replaced by the linked content

_blank
: A new un-named window or tab is requested for the display of the linked content. If this fails, the result is the same as **_top**

<XML-Name>
: Specifies the name of the frame, pane, or other relevant presentation context for display of the linked content. If this already exists, it is re-used, replacing the existing content. If it does not exist, it is created (the same as **'_blank'**, except that it now has a name).

Note: The value **'_new'** is *not* a legal value for target (use **'_blank'**).

Animatable: yes.

`focusable` = **"true" | "false" | "auto"**

See attribute definition for description.

Animatable: yes.

`Navigation Attributes`

See definition.

14.2.1 Indicating links

Typically, HTML user agents, by convention, style the content of anchor elements to indicate that they are links, for example by underlining and changing the color of text and creating a colored border around images and other replacement content. Because SVG is a visual language with irregular shapes and complex link structure (e.g. allowing links within other links), and is intended to allow more precise control over the appearance of the content, SVG user agents should not provide default styling to child content of an **'a'** element, instead allowing authors to control the linking conventions.

However, in order to ensure that links are obvious to users and to provide detailed information about each link, SVG user agents should provide a clear indicator when a link is in scope. A link shall be considered to be in scope if one of the child elements of that **'a'** element has a pointer device cursor hovered over it or when that element is the currently focused element. The user agent should change the scope indicator to signal that a link is in scope (e.g.

the cursor may be changed to a pointing hand, or the focus highlight may be color-coded to indicate the status of the link), should indicate the URI of the link (by displaying it in a status bar, or reading it aloud, for example), and should display any author-supplied information about the link (as with a tooltip). Authors should use the **'xlink:title'** attribute appropriately on links, in order to provide information about the link to users.

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., **someDrawing.svg#someView**). 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., **someDrawing.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 that the corresponding attribute has on a **'g'** element has).

An SVG fragment identifier is defined as follows:

```
SVGFragmentIdentifier ::= BareName |
                          SVGViewSpec

BareName ::= NCName
SVGViewSpec ::= 'svgView(' SVGViewAttributes ')'
SVGViewAttributes ::= SVGViewAttribute |
                      SVGViewAttribute ';' SVGViewAttributes

SVGViewAttribute ::= transformSpec
transformSpec ::= 'transform(' TransformList ')'
```

where:

- **NCName** is an <NCName> value.
- **TransformList** corresponds to the TransformList value. For example, **transform(scale(5))**.

An SVG fragment identifier must match the specified grammar. To ensure robust content, authors are recommended to omit spaces between numeric values, or replace these spaces with percent-encoded strings or commas as appropriate.

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 user agent traverses a link to an SVG document fragment, whether from within the same document or from an external source document, then the SVG fragment identifier shall specify the initial view into the SVG document. This applies whether the link is from an SVG **'a'** element, an HTML anchor element [HTML4] (i.e., an `<a href=...>` element in HTML), or any specification using XLink [XLINK10]. The user agent shall take the following steps in determining the effect of the link traversal:

- If no SVG fragment identifier is provided (i.e., the specified IRI did not contain a "#" character, such as **someDrawing.svg**), then the initial view into the SVG document shall be established using the view specification attributes (i.e., viewBox, etc.) on the rootmost 'svg' element.
- If the SVG fragment identifier addresses specific SVG view (e.g., **linking-svgView-102-t.svg#svgView(transform(rotate(30, 150, 150)))**), then the document fragment defined by the closest ancestor **'svg'** element is displayed in the viewport using the SVG view specification provided by the SVG fragment identifier.
- If the SVG fragment identifier addresses any element (e.g., **#rectId** or **someDrawing.svg#rectId**) and the element indicated by the fragment identifier is found, then the current translation of the SVG document's coordinate

system shall be adjusted such that the centerpoint of the decorated bounding box of the identified element is positioned in the center of the viewport. If the element's decorated bounding box is too large to fit within the current viewport, and the **'zoomAndPan'** attribute of the rootmost 'svg' element is not set to **'disable'**, then the viewport shall not only reposition but also have the current scale expanded to accommodate the entire width and height of the element's decorated bounding box. By contrast, if the bounding box of the target element is smaller than the viewport, the viewport shall remain at the preestablished values (i.e., it will not automatically zoom in on the element). If the specified element does not have a decorated bounding box, then the current translate and current scale are not changed from the established values. Regardless of changes to the current translation or scale of the viewport, the current rotation of the current coordinate system shall be preserved (that is, the centerpoint of the target decorated bounding box shall be the centerpoint of the rotation, with a constant rotation angle), and the existing aspect ratio shall not be altered. In the case of traversal from an external link, the viewport shall be established by the values specified in the rootmost 'svg' element, and in the case of an internal link, the initial viewport shall additionally be adjusted by any previous zooming operations (e.g. previously navigated links, user zooming, script alterations of the current coordinate system, etc.) such that any translation or scaling that happens as a result of the traversal shall use the existing coordinate system as a starting state. If the element is not found, or does not have a decorated bounding box, then the viewport does not move or zoom. In all cases of traversal, the view shall be established instantly, with no animated panning or other enhanced transition toward the target element. The viewbox shall not be continually animated to match the animations of a target element's decorated bounding box. Future specifications may allow more customizable behavior for traversal through another mechanism.

- If the SVG fragment identifier addresses any element and the element is not found, the initial view into the SVG document shall be established using the view specification attributes (i.e., viewBox, etc.) on the rootmost 'svg' element, as if no fragment had been specified.

Note: In SVG Tiny 1.2, only a single **'svg'** element is allowed. Thus, the closest ancestor **'svg'** element and the rootmost 'svg' element are the same. This is not true in other profiles of SVG, where the distinction becomes significant.

15 Scripting

Contents

15.1 Specifying the scripting language. 205
15.1.1 Specifying the default scripting language. 205
15.1.2 Local declaration of a scripting language 205
15.2 The 'script' element 205
15.2.1 Script processing 205
15.3 XML Events 208
15.4 The 'listener' element. 210
15.5 The 'handler' element 212
15.5.1 Parameters to 'handler' elements. 214
15.6 Event handling. 214
15.7 Processing inline executable content 214

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 [ECMA-262] 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. If a **'script'** element has both an **'xlink:href'** attribute and child character data, the executable content for the script is retrieved from the IRI of the **'xlink:href'** attribute, and the child content is not added to the scripting context.

When the executable content is inlined, it must be processed as described in Processing inline executable content.

Some scripting languages such as ECMAScript 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 implement the `SVGGlobal` interface. In addition to being implemented on the global ECMAScript object, the `SVGGlobal` object can also be obtained through the `DocumentView::defaultView` attribute on the `Document` object. 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.

15.2.1 Script processing

Execution of a given **'script'** element occurs at most once. There is a conceptual flag associated with each **'script'** element (referred to here as the "already processed" flag) that enforces this behavior. When a **'script'** element is executed depends on the method by which the element was inserted into the document.

One way for a **'script'** element to be inserted into the document is if it was inserted while parsing the document. As mentioned in Progressive rendering, as the document is parsed if a **'script'** element is encountered then it will be processed just after its *end element* event occurs, but before any more of the document is parsed and further nodes inserted into the document. (See below for a description of what it means for a **'script'** element to be processed.) Once processed, parsing of the document resumes.

The other way a **'script'** element can be inserted into the document is if it was inserted by something other than the parser (such as by other script executing). In this case, as soon as one or more **'script'** elements are inserted into the document, they must be processed one by one in document order.

A **'script'** element is processed as follows:

1. If the **'script'** element's "already processed" flag is true or if the element is not in the document tree, then no action is performed and these steps are ended.
2. If the **'script'** element references external script content, then the external script content using the current value of the **'xlink:href'** attribute is fetched. Further processing of the **'script'** element is dependent on the external script content, and will block here until the resource has been fetched or is determined to be an invalid IRI reference.
3. The **'script'** element's "already processed" flag is set to true.
4. If the script content is inline, or if it is external and was fetched successfully, then the script is executed. Note that at this point, these steps may be re-entrant if the execution of the script results in further **'script'** elements being inserted into the document.

Note that a `load` event is dispatched on a **'script'** element once it has been processed, unless it referenced external script content with an invalid IRI reference and **'externalResourcesRequired'** was set to **'true'**.

Modifying or removing the **'script'** element (or content) after the script has started its execution must have no effect on the script execution.

Modifying a **'script'** element's **'xlink:href'** attribute after its "already processed" flag is set to true will not cause any new script content to be fetched or executed.

What it means to execute some script content depends on the script content 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 must 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.

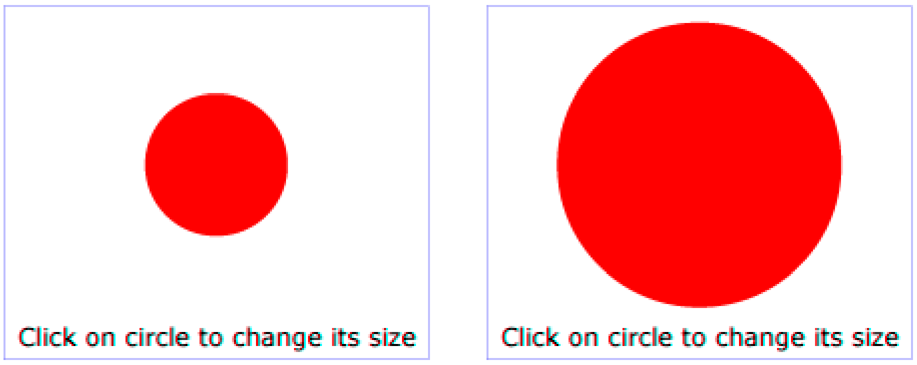
Other language bindings are encouraged to adopt a similar approach to either of the two described above.

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 `DOMActivate` event should be used instead of the `click` event.

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="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

Here the same script is invoked, this time in an external file.

Example: 18_02.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 external ECMAScript function from an click event
  </desc>
  <!-- ECMAScript to change the radius with each click -->
  <script type="application/ecmascript" xlink:href="sample.es"/>

  <!-- 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="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>
```

Schema: script

```
<define name='script'>
  <element name='script'>
    <ref name='script.AT'/>
    <ref name='script.ATCM'/>
  </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.ATCM'>
  <interleave>
    <choice>
      <group>
        <ref name='svg.XLinkRequired.attr'/>
      </group>
      <text/>
    </choice>
    <ref name='svg.Desc.group'/>
  </interleave>
</define>
```

Attribute definitions:

`type` = **"<content-type>"**

Identifies the programming language for the **'script'** element. The **"<content-type>"** value specifies a media type, per *Multipurpose Internet Mail Extensions (MIME) Part Two* [RFC2046]. If **'type'** is not specified, the value of **'contentScriptType'** on the **'svg'** element shall be used, which in turn has a lacuna value of **'application/ecmascript'** [RFC4329]. If a **'script'** element is not inside an SVG document fragment, **'type'** must default to **'application/ecmascript'**. This can happen for example if the **'script'** element is a child of some arbitrary non-SVG markup.

Animatable: no.

`xlink:href` = **"<IRI>"**

An IRI reference to an external resource containing the script code. If the attribute contains an invalid IRI reference, the **'script'** element will not execute any script.

Animatable: no.

15.3 XML Events

XML Events [XML-EVENTS] is an XML syntax for integrating event listeners and handlers with *DOM Level 2 Events* [DOM2EVENTS]. 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, through the inclusion of the **'listener'** and **'handler'** elements to provide the ability to specify the event listener separately from the graphical content.

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"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny">

  <rect x="10" y="20" width="10" height="20" fill="red">
    <handler type="application/ecmascript" ev:event="click">
      var theRect = evt.target;
      var width = theRect.getFloatTrait("width");
      theRect.setFloatTrait("width", width + 10);
    </handler>
  </rect>

</svg>
```

In this method the **'handler'** element is a child element of the observer ([XML-EVENTS], section 3.1). 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 (`click`, in this case) 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="theRect1" x="10" y="20" width="10" height="20" fill="red"/>
  <rect xml:id="theRect2" x="10" y="40" width="10" height="20" fill="green"/>

  <ev:listener event="click" observer="theRect1" handler="#theClickHandler"/>
  <ev:listener event="click" observer="theRect2" handler="#theClickHandler"/>

  <handler xml:id="theClickHandler" type="application/ecmascript">
    var theRect = evt.target;
    var width = theRect.getFloatTrait("width");
    theRect.setFloatTrait("width", (width+10));
  </handler>

</svg>
```

In the above example, the **'listener'** element registers that the "theClickHandler" element should be invoked whenever a `click` event happens on "theRect1" or "theRect2".

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="theRect" 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/theJar.jar"/>

  <!-- register a listener for a theRect.click event -->
  <ev:listener event="click" observer="theRect" handler="#theClickHandler" />

  <handler xml:id="theClickHandler" 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 `MyListenerInitializer` class referenced by the SVG-Handler-Class entry of the `theJar.jar` manifest has the following definition:

MyListenerInitializer

```
package com.example;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.events.Event;
import org.w3c.dom.events.EventListener;
import org.w3c.dom.svg.EventListenerInitializer2;

public class MyListenerInitializer 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 theRect = document.getElementById("theRect");
                float width = Float.parseFloat(theRect.getAttributeNS(null, "width"));
                float offset = Float.parseFloat(handlerElement.getAttributeNS("http://www.example.com/foo",
"offset");
                theRect.setAttributeNS(null, "width", "" + (width + offset));
            }
        };
    }
}
```

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 the attributes of a listener element are modified, the corresponding event listener is removed and a new one is created. When listener elements are added or removed, the event listener is added or removed respectively.

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='XML-NMTOKEN.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' 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='IDREF.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='target' svg:animatable='false' svg:inheritable='false'>
          <ref name='IDREF.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` = **"<XML-NMTOKEN>"**

See the XML Events **'event'** attribute definition. The **'event'** attribute must be a valid SVG Tiny 1.2 event identifier as defined in the list of supported events.

Animatable: no.

`observer` = **"<IDREF>"**

See the XML Events **'observer'** attribute definition. Note that if the **'observer'** attribute is not present, the observer is the parent of the **'listener'** element.

Animatable: no.

`target` = **"<IDREF>"**

See the XML Events **'target'** attribute definition.

Animatable: no.

`handler` = **"<IRI>"**
: See the XML Events **'handler'** attribute definition. This attribute is an IRI reference. Restrictions specified in this chapter as to which IRIs are acceptable must be enforced.
: *Animatable: no.*

`phase` = **"default"**
: See the XML Events **'phase'** attribute definition. Support for the capture phase is not required in SVG Tiny 1.2, and implementations that do not support it must process this attribute as if it had not been specified.
: *Animatable: no.*

`propagate` = **"stop" | "continue"**
: See the XML Events **'propagate'** attribute definition.
: *Animatable: no.*

`defaultAction` = **"cancel" | "perform"**
: See the XML Events **'defaultAction'** attribute definition.
: *Animatable: no.*

15.5 The **'handler'** element

The **'handler'** element is similar to the **'script'** element: its contents, either included inline or referenced, is code that is to be executed by a scripting engine used by user agent.

However, where the **'script'** element executes its contents when it 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 ([SVG11], section 18.4).

When the executable content is inlined, it must be processed as described in Processing inline executable content.

For example, consider the following SVG 1.1 document:

Example: nohandler.svg

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect id="theRect" x="10" y="20" width="200" height="300" fill="red"
        onclick="evt.target.width.baseVal.value += 10"/>
</svg>
```

The above example must be rewritten, as described below, to use the **'handler'** element and XML Events as shown:

Example: handler2.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny">

  <desc>handler element example</desc>

  <rect xml:id="theRect" x="10" y="20" width="200" height="300" fill="red">
    <handler type="application/ecmascript" ev:event="click">
      var theRect = evt.target;
      var width = theRect.getFloatTrait("width");
      theRect.setFloatTrait("width", width + 10);
    </handler>
  </rect>

</svg>
```

Whenever the **'type'** or **ev:event** attributes of a **'handler'** element are modified, the corresponding event listener is removed and a new one is created. When the **'xlink:href'** attribute is modified or the content of the **'handler'** element is modified, the existing event listener is preserved, but the user agent must execute the updated handler logic. When **'handler'** elements are added or removed, the corresponding event listener is added or removed respectively.

In ECMAScript, the contents of the **'handler'** element behave conceptually as if they are the contents of a new Function object, created as shown:

```
    function(event) {
        var evt = event;
        //contents of handler
    }
```

The 'event' parameter shown above is an Event object corresponding to the event that has triggered the **'handler'**. An 'evt' variable can be used instead of 'event' ('evt' is an alias to 'event').

Schema: handler

```
      <define name='handler'>
        <element name='handler'>
          <ref name='handler.AT'/>
          <ref name='handler.ATCM'/>
        </element>
      </define>

      <define name='handler.AT' combine='interleave'>
        <ref name='svg.CorePreserve.attr'/>
        <ref name='svg.External.attr'/>
        <optional>
          <attribute name='ev:event' svg:animatable='false' svg:inheritable='false'>
            <ref name='XML-NMTOKEN.datatype'/>
          </attribute>
        </optional>
        <ref name='svg.ContentType.attr'/>
      </define>

      <define name='handler.ATCM'>
        <interleave>
          <choice>
            <group>
              <ref name='svg.XLinkRequired.attr'/>
            </group>
            <text/>
          </choice>
          <ref name='svg.Desc.group'/>
        </interleave>
      </define>
```

Attribute definitions:

`type` = **"<content-type>"**

Identifies the programming language for the **'handler'** element. The **"<content-type>"** value specifies a media type, per *Multipurpose Internet Mail Extensions (MIME) Part Two* [RFC2046]. If **'type'** is not specified, the value of **'contentScriptType'** on the **'svg'** element shall be used, which in turn has a lacuna value of **'application/ecmascript'** [RFC4329].

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 executed.

Animatable: no.

`ev:event` = **"<XML-NMTOKEN>"**

The name of the event to handle. This attribute is in the XML Events namespace, `http://www.w3.org/2001/xml-events`. See event list for a list of all supported events and the XML Events specification for the definition of the **'ev:event' attribute** ([XML-EVENTS], section 3.1).

Animatable: no.

For script handlers in Java, 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 in 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"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     xmlns:foo="http://www.example.com/foo"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.2" baseProfile="tiny">

  <desc>An example of parameters on the handler element.</desc>

  <rect xml:id="theRect" 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/theJar.jar"/>

  <!-- register a listener for a theRect.click event -->
  <ev:listener event="click" observer="theRect" handler="#theClickHandler" />

  <handler xml:id="theClickHandler" 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 `theJar.jar` manifest has its `createEventListener` method called and the returned `EventListener` registered. Whenever a `click` event on the "theRect" element is observed, the `handleEvent` method of the listener is called. The object can then access the **'handler'** element in order to obtain its parameters from elements in the `http://www.example.com/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 *DOM Level 2 Events* [DOM2EVENTS] are defined which are invoked when a given event happens on a given object.

Related sections of the spec:

- User interface events describes how an SVG user agent handles events such as pointer movements events (e.g., mouse movement) and activation events (e.g., mouse click).

15.7 Processing inline executable content

When executable content is inlined inside an executable element such as **'script'** or **'handler'** elements, it must be processed as follows before it is parsed and executed.

If the type of the content, obtained either through the **'type'** attribute, the **'contentScriptType'** or its lacuna value, attribute, or the default is not known by the user agent, then the user agent must ignore the content and no further processing must be performed.

If the type of the content is an XML media type [RFC3023], then the entire subtree of the executable element must be passed on untouched to the script engine.

Otherwise, the content that the user agent's script engine obtains must be that which is obtained through the following steps:

1. Remove all descendant elements of the executable element.
2. Then, use the text content of the executable element (as defined by `Node::textContent` in DOM Level 3 Core ([DOM3], section 1.4) of the executable element.

16 Animation

Contents

16.1 Introduction 216
16.2 Animation elements. 216
16.2.1 Overview 216
16.2.2 Relationship to SMIL 2.1 Animation 216
16.2.3 Animation elements example 217
16.2.4 Attributes to identify the target element for an animation. 218
16.2.5 Attributes to identify the target attribute or property for an animation. 219
16.2.6 Animation with namespaces 220
16.2.7 Paced animation and complex types 221
16.2.8 Attributes to control the timing of the animation 222
16.2.9 Attributes that define animation values over time. 226
16.2.10 Attributes that control whether animations are additive 230
16.2.11 Inheritance 231
16.2.12 The 'animate' element 231
16.2.13 The 'set' element. 232
16.2.14 The 'animateMotion' element 232
16.2.15 The 'mpath' element. 236
16.2.16 The 'animateColor' element 237
16.2.17 The 'animateTransform' element 237
16.2.18 Attributes and properties that can be animated. 239
16.3 Animation using the SVG DOM 240
16.4 Animation and the bounding box. 241

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) 2.1 Specification [SMIL21].

SVG incorporates the animation features defined in the SMIL 2.1 specification and provides some SVG-specific extensions.

For an introduction to the approach and features available in any language that supports SMIL 2.1 animation, see the Introduction and animation sandwich model sections of the SMIL 2.1 Animation Modules ([SMIL21], sections 3.2 and 3.3.3). For the list of animation features which go beyond SMIL Animation, see SVG extensions to SMIL 2.1 Animation below.

16.2.2 Relationship to SMIL 2.1 Animation

SVG is a host language in terms of SMIL 2.1 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 those in the SMIL 2.1 Animation Modules and the SMIL 2.1 Timing and Synchronization chapter ([SMIL21], sections 3 and 10). Note that the SMIL timing engine is orthogonal to the rendering tree.

SVG supports the following four animation elements which are defined in the SMIL 2.1 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 2.1:

'animateTransform'	modifies one of SVG's transformation attributes over time, such as the **'transform'** attribute
'path' attribute	allows any feature from SVG's path data syntax to be specified in a **'path'** attribute to the **'animateMotion'** element (SMIL 2.1 Animation only allows a subset of SVG's path data syntax within a **'path'** attribute)
'mpath' element	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	adds a **'keyPoints'** attribute to the **'animateMotion'** to provide precise control of the velocity of motion path animations
'rotate' attribute	adds a **'rotate'** attribute to the **'animateMotion'** to control whether an object is automatically rotated so that its x-axis points in the same direction (or opposite direction) as the directional tangent vector of the motion path

For compatibility with other aspects of the language, SVG uses IRI references via an **'xlink:href'** attribute to identify the elements which are to be targets of the animations.

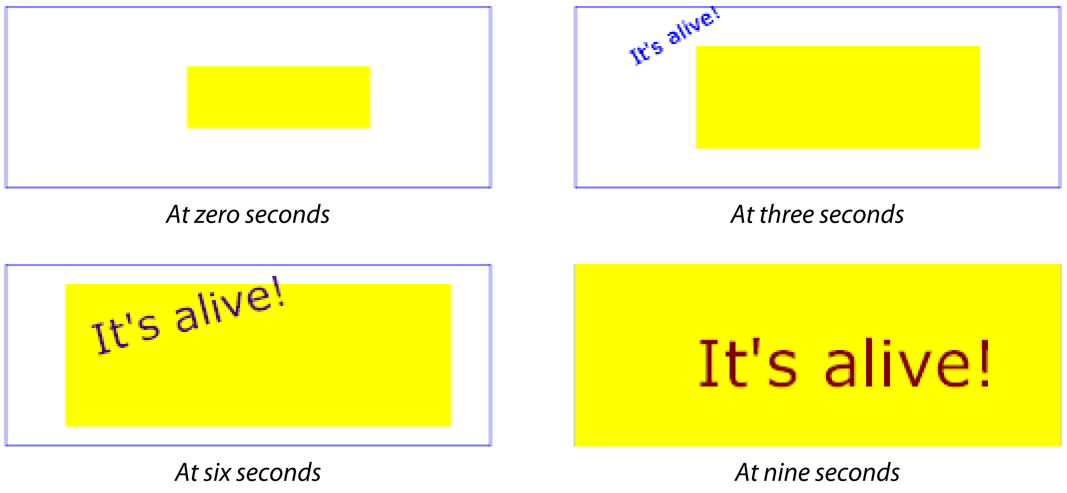
SMIL 2.1 Animation requires that the host language define the meaning for document begin and document end, and these are defined in the Definitions section.

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>
```



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 has no effect.

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'/>
      <ref name='svg.Conditional.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 or the IRI reference is otherwise an invalid IRI reference, then the animation has no effect. This means the animation timing will still run but no animation effect will be applied to the target attribute.

If the **'xlink:href'** attribute is not provided, then the target element will be the immediate parent element of the current animation element.

'xlink:href' must point to exactly one target element. If **'xlink:href'** 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 **xlink:href=""** (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 found in the Specifying the animation target section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.1).

Animatable: no.

If the **'xml:id'** or **'id'** attribute of an element that is the target of an animation is changed, then whether this affects the animation is undefined, and implementation dependent. Future specifications will describe in more detail the result of modifying elements that affect animation elements which are already in the document tree.

Resolution of the animation target occurs when the animation element is inserted into the document tree or at the document begin time, whichever is later. (See also the **'timelineBegin'** attribute, which controls when the document begin time occurs.)

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: animateattributecommon

```
<define name='svg.AnimateAttributeCommon.attr'>
  <optional>
    <attribute name='attributeName' svg:animatable='false' svg:inheritable='false'>
      <ref name='QName.datatype'/>
    </attribute>
  </optional>
  <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` **= "<QName>"**

Specifies the name of the target attribute. A qualified name, if used, indicates the XML namespace for the attribute. The prefix will be resolved to a namespace name in the scope of the current (i.e. the referencing) animation element. Note that while the **'attributeName'** attribute is optional, there is no lacuna value. When no value is specified the animation still runs, and events related to the animation element (e.g. `beginEvent`) are still fired, but the animation doesn't apply to any attribute.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the Specifying the animation target section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.1).

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 property defined as animatable in this specification.

XML

This specifies that the value of **'attributeName'** is the name of an XML attribute on the target element. If the value for **'attributeName'** has a prefix, it is resolved as described above. If not, then the target attribute is in no namespace. The attribute must be defined as animatable in this specification.

auto

The user agent should match the **'attributeName'** to an animatable property or attribute applicable to the target element. If the user agent supports CSS, it must first search through its list of supported CSS properties (including those not defined by SVG) for a matching property name that is defined to be animatable for the target element. If no suitable property is found, the user agent must then search for a suitable animatable attibute, using the appropriate namespace. If the **'attributeName'** value has no prefix, then the attribute searched for is in no namespace. If it does have a prefix, then that prefix is resolved as described above and the attribute searched for in the associated namespace. If no suitable property or attribute is found, the animation has no effect.

Note that not all properties are applicable to all SVG elements, even if an attribute of the same name is applicable to the element. In particular, the CSS **'width'** and **'height'** properties are not applicable to those SVG elements which have **'width'** and **'height'** attributes.

The lacuna value is **'auto'**.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the Specifying the animation target section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.1).

In the absence of style sheets (external style sheets, **'style'** elements or **'style'** attributes, as may be available in other profiles of SVG) 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"**.

Animatable: no.

16.2.6 Animation with namespaces

Example animns01 below shows a namespace prefix being resolved to a namespace name in the scope of the referencing element, and that namespace name being used (regardless of the prefix which happens to be used in the target scope) to identify the attribute being animated.

Example: animns01.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">
    <title>Demonstration of the resolution of namespaces for animation</title>
    <!-- at the point of definition, the QName a:href resolves to the namespace
        name "http://www.w3.org/1999/xlink" and the local name "href" -->
    <g xmlns:a="http://www.w3.org/1999/xlink">
        <animate attributeName="a:href" xlink:href="#foo" dur="2s" to="two.png" fill="freeze"/>
    </g>
    <!-- at the point of use, the namespace name "http://www.w3.org/1999/xlink"
      happens to be bound to the namespace prefix 'b'  while the prefix
      'xlink' is bound to a different namespace name-->
    <g xmlns:b="http://www.w3.org/1999/xlink" xmlns:xlink="http://example.net/bar">
        <image xml:id="foo" b:href="one.png" x="35" y="50" width="410" height="160"/>
    </g>
</svg>
```

16.2.7 Paced animation and complex types

Paced animations assume a notion of distance between the various animation values defined by the **'to'**, **'from'**, **'by'** and **'values'** attributes. Distance is defined only for scalar types (such as <length>), colors and the subset of transformation types that are supported by **'animateTransform'**. In the list of distance functions below, V_a and V_b represent the two values the distance between which is being calculated.

Since paced animation is intended to produce an animation with an even pace of change, it does not make sense to define distance functions for all data types. Distance can be usefully defined for types whose values are *n*-dimensional vectors (including scalars, which are 1-dimensional vectors). For example, a <length> value is a scalar value, and a <color> value is a 3-dimensional vector. Thus attributes of these types can have paced animation applied to them. On the other hand, a <list-of-length> (as used by **'stroke-dasharray'**) is a list of scalars (1-dimensional vectors), and <points-data> (as used by the **'points'** attribute on a **'polygon'**) is a list of 2-dimensional vectors. Therefore, these types do not have a distance function defined and cannot have paced animation applied to them.

The distance functions for types that support paced animation are as follows:

<coordinate>, <integer>, <length> and <number>

distance(V_a, V_b) = $|V_a - V_b|$

Examples: animating the **'x'** attribute on a **'rect'**, or the **'stroke-width'** property on a **'circle'**.

<color>

distance(V_a, V_b) = sqrt((V_a.red – V_b.red)^2 + (V_a.green – V_b.green)^2 + (V_a.blue – V_b.blue)^2), where:

V_i.red is the red component of the V_i color value,

V_i.green is the green component of the V_i color value, and

V_i.blue is the blue component of the V_i color value.

Each of the color component values is usually in the range [0, 1], where 0 represents none of that color component, and 1 represents the maximum amount of that color component, in the sRGB gamut [SRGB]. Since <color> values may specify colors outside of the sRGB gamut, these component values may lie outside the range [0, 1].

Example: animating the **'fill'** property on an **'ellipse'**.

Transform definitions of type 'translate'

distance(V_a, V_b) = sqrt((V_a.tx – V_b.tx)^2 + (V_a.ty – V_b.ty)^2), where:

V_i.tx is the *x* component of the V_i translation transform value, and

V_i.ty is the *y* component of the V_i translation transform value.

Example (for all transform definition types): animating the **'transform'** attribute on a **'g'** using **'animateTransform'**.

Transform definitions of type 'scale'

distance(V_a, V_b) = sqrt((V_a.sx – V_b.sx)^2 + (V_a.sy – V_b.sy)^2), where:

V_i.sx is the *x* component of the V_i scale transform value, and

V_i.sy is the *y* component of the V_i scale transform value.

Note that, as when specifying scale transformations in a <transform-list>, if the *y* component of the scale is omitted it is implicitly equal to the *x* component.

Transform definitions of type 'rotate', 'skewX' and 'skewY'

distance(V_a, V_b) = sqrt((V_a.angle – V_b.angle)^2), where:

V_i.angle is the angle component of the V_i rotation or skew transform value.

Since the distance function for rotations is not in terms of the rotation center point components, a paced animation that changes the rotation center point may not appear to have a paced movement when the animation is applied.

Distance functions for all other data types are not defined. If **calcMode="paced"** is used on an animation of an attribute or property whose type is not one of those listed above, the animation effect is undefined. SVG user agents may choose to perform the animation as if **calcMode="linear"**, but this is not required. Authors are recommended not to specify paced animation on types not listed above.

16.2.8 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.

These timing attributes also apply to media elements.

Schema: animatetiming

```
    <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>

    <define name='svg.AnimateTimingNoMinMax.attr' combine='interleave'>
      <ref name='svg.AnimateTimingNoFillNoMinMax.attr'/>
      <optional>
        <attribute name='fill' svg:animatable='false' svg:inheritable='false'>
          <choice>
            <value>remove</value>
            <value>freeze</value>
          </choice>
        </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' svg:animatable='false' svg:inheritable='false'>
          <choice>
            <value>always</value>
            <value>never</value>
            <value>whenNotActive</value>
          </choice>
        </attribute>
      </optional>
    </define>

    <define name='svg.AnimateBegin.attr' combine='interleave'>
      <optional><attribute name='begin' 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 timing specifier values.

The definition of ***begin-value-list*** is as follows:

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 2.1 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 time.

offset-value ::= (S? "+" | "-" S?)? (<Clock-value>)

For SMIL 2.1, 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 dispatched to the event-base element. The list of event names available for use as the **event-ref** value can be found in the "Animation event name" column of the Complete list of supported events table. Details of event-based timing are described in the Unifying event based and scheduled timing section of SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.11).

repeat-value ::= (Id-value ".")? "repeat(" <integer> ")" (S? ("+" | "-") S? <Clock-value>)?

Describes a qualified repeat event and an optional offset that will begin the element. The element begin is defined relative to the time that the `repeatEvent` event is dispatched to the event-base element with the iteration value specified by the integer, which must be positive.

accessKey-value ::= "accessKey(" character ")" (S? ("+" | "-") S? <Clock-value>)?

Describes a key press event and an optional offset that will begin the element. The element begin is defined relative to the time of the `keydown` event corresponding to the specified key. From a formal processing model perspective, an accessKey-value corresponds to a `keydown` event listener on the document which behaves as if `stopPropagation()` and `preventDefault()` have both been invoked in the capture phase. (See Event flow for consequences of this behavior.)

The **character** value can be one of two types. It can be any of the keyboard event identifier strings listed in the key identifiers set. Alternatively, this value can be a single Unicode character [UNICODE]. In this case, the single character is mapped to a keyboard identifier for the purpose of processing the accessKey-value event listener. For example, **'accessKey(Q)'** will translate into a `keydown` event listener using "U+0051" as the target keyboard identifier string, as if **'accessKey(U+0051)'** were used.

"indefinite"

The begin of the animation will be determined by a `beginElement()` method call or a hyperlink targeted to the element. (Hyperlink-based timing is described in SMIL 2.1 Timing and Synchronization: Hyperlinks and timing.)

In SVG, if no **'begin'** is specified, the default timing of the time container is equivalent to an offset value of **'0'**. If the **'begin'** attribute is syntactically invalid, in the list itself or in any of the individual list values, it is equivalent to a single **'begin'** value of **'indefinite'**.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the basic timing support section of SMIL 2.1 Timing and Synchronization. See also the special parsing rules for **Id-value** and **event-ref** as described in the Parsing timing specifiers ([SMIL21], section 10.4.1).

Animatable: no.

dur **= "<Clock-value>" | "media" | "indefinite"**

Specifies the simple duration. The attribute value can one of the following:

<Clock-value>

Specifies the length of the simple duration in document 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 found in the Dur value semantics section of SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.4.1).

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 an `endElement()` method call.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the basic timing support section of SMIL 2.1 Timing and Synchronization. See also the special parsing rules for **Id-value** and **event-ref** as described in the Parsing timing specifiers ([SMIL21], section 10.4.1).

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.

The 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 lacuna 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 found in the More control over the active duration section of SMIL 2.1 Timing and Synchronization. ([SMIL21], section 10.4.1).

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.

The 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 lacuna value for **'max'**. If omitted, the active duration is not constrained at all.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the More control over the active duration section of SMIL 2.1 Timing and Synchronization. ([SMIL21], section 10.4.1).

Animatable: no.

`restart` = **"always" | "whenNotActive" | "never"**

always
: The animation can be restarted at any time. This is the lacuna 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, 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 found in The restart attribute section of SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.4.1).

Animatable: no.

`repeatCount` = **"<number>" | "indefinite"**

Specifies the number of iterations of the animation function. It can have the following attribute values:

<number>
: 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 ([SMIL21], section 3.5.2). 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 found in the Repeating elements section of SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.4.1).

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 document time to repeat the animation function `f(t)` ([SMIL21], section 3.3.1).

"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 found in the Repeating elements section of SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.4.1).

Animatable: no.

`fill` = **"freeze" | "remove"**

This attribute can have the following values:

freeze
: The animation effect `f(t)` ([SMIL21], section 3.3.1) is defined to freeze the effect value at the last value of the active duration. Note that in the case of discrete animation, the frozen value that is used is the value of the animation just before the end of the active duration. The animation effect is "frozen" for the remainder of the document duration (or until the animation is restarted — see The restart attribute ([SMIL21], section 10.4.1).

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 The restart attribute ([SMIL21], section 10.4.1).

This is the lacuna value.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the Extending an element section of SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.4.1).

Animatable: no.

SMIL 2.1 defines the detailed processing rules associated with the above attributes.

event-base element

This section is informative.

For a normative definition of event-base elements, see the SMIL 2.1 Animation Modules ([SMIL21], section 3) and the Multimedia section of this specification. For animation elements and the **'discard'** element, the default event-base element is the animation target, which for elements with an **'xlink:href'** attribute is the target IRI, and is otherwise the parent element. The default event-base element for all SVG media elements (e.g. **'audio'**, **'video'** and **'animation'**) is the element itself. For all event-values that are prefaced with an **Id-value**, the event-base element is the element indicated by that ID.

Authoring note: non-rendered elements such as the **'audio'** element cannot receive user-initiated pointer events, so it is recommended that authors specify a rendered element as the event-base element for such cases when using user interface events.

Clock values

Clock values have a subsetted syntax of the clock values syntax defined in SMIL 2.1 ([SMIL21], section 10.4.1):

```
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 document 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
```

16.2.9 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 to be used between particular 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 lacuna value is **'linear'**, except on **'animateMotion'** elements where it is **'paced'**. However if the target attribute does not support interpolation (e.g. when animating an attribute whose type is <string>), the **'calcMode'** attribute is ignored and discrete animation 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.

paced

Defines interpolation to produce an even pace of change across the animation. This is only supported for the data types for which there is an appropriate distance function defined, which includes only scalar numeric types plus the types listed in Paced animation and complex types. If **'paced'** is specified, any **'keyTimes'** or **'keySplines'** will be ignored. Authors are discouraged from using paced animation on types that do not have a distance function defined, due to its unpredictable behavior in some user agents.

spline

Interpolates from one value in the **'values'** list to the next according to a time function defined by a cubic Bézier spline. The points of the spline are specified in the **'keyTimes'** attribute, and the control points for each interval are specified in the **'keySplines'** attribute.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the Simple animation function attributes section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.2).

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 found in the Simple animation function attributes section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.2). Note that, per the SMIL specification, leading and trailing white space, and white space before and after semi-colon separators, is allowed and will be ignored.

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:

- 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 **'keyTimes'** value associated with each value defines when the value is set; values are interpolated between the **'keyTimes'**.
- For discrete animation, the first time value in the list must be 0. The time associated with each value defines when the value is set; the animation function uses that value until the next time defined in **'keyTimes'**.

If the interpolation mode is **'paced'**, the **'keyTimes'** attribute is ignored.

If the **'keyTimes'** attribute has a value that doesn't conform to the above requirements 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.

Because paced animation interpolation is unspecified for some value types, authors are encouraged to use **'linear'** animation interpolation with calculated **'keyTimes'** to achieve particular interpolation behavior for these types.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the Calculation mode attributes section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.8.1).

Animatable: no.

`keySplines` = **"<list>"**

A set of Bézier control points associated with the **'keyTimes'** list, defining a cubic Bézier function that controls interval pacing. The attribute value is a semicolon separated list of control point descriptions. Each control point description is a set of four values: `x1 y1 x2 y2`, describing the Bézier control points for one time segment. Note SMIL 2.1 allows these values to be separated either by commas with optional white space, or by white space alone. The **'keyTimes'** values that define the associated segment are the Bézier "anchor points", and the **'keySplines'** values are the control points. Thus, there must be one fewer set 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 the **'keySplines'** attribute has a value that doesn't conform to the above requirements 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 found in the Calculation mode attributes section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.8.1).

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 found in the Simple animation functions specified by from, to and by section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.4).

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 found in the Simple animation functions specified by from, to and by section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.4).

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 found in the Simple animation functions specified by from, to and by section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.4).

Animatable: no.

SMIL 2.1 defines the detailed processing rules associated with the above attributes.

The animation values specified in the **'values'**, **'from'**, **'to'** and **'by'** attributes must be legal values for the specified target attribute. Leading and trailing white space, and white space before and after semicolon separators, will be ignored. If any values are not legal, they are considered to be unsupported and is processed as if they had not been specified.

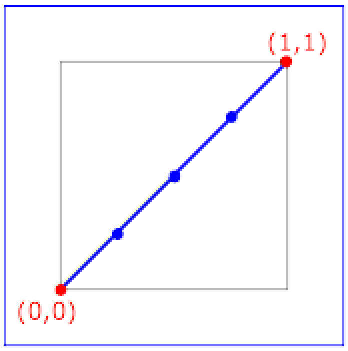
If a list of **'values'** is used, the animation will apply the values in order over the course of the animation. If the **'values'** attribute is specified, any **'from'**, **'to'** or **'by'** attribute values are ignored.

The processing rules for the variants of from/to/by animations, including which of the **'from'**, **'to'** and **'by'** attributes take precedence over the others, is described in Simple animation functions specified by from, to and by ([SMIL21], section 3.5.4).

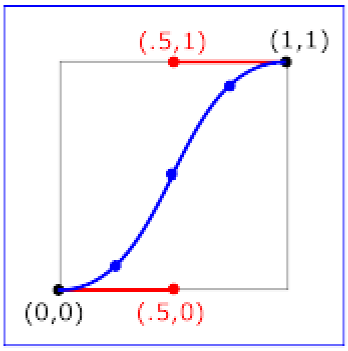
When animating properties, interpolation is performed on computed values. Thus, keywords such as **inherit** which yield a numeric computed value may be included in the values list for an interpolated animation. For example, a linear animation of the **'fill'** property using **values="red; inherit; blue"** is possible as long as the **inherit** value does not compute to a paint server reference, a system paint or the **none** value.

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 Timing and real-world clock times section in SMIL 2.1 Timing and Synchronization ([SMIL21], section 10.4.3).

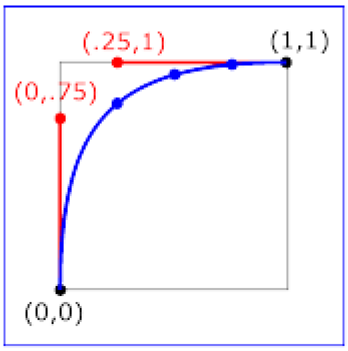
Examples of 'keySplines'



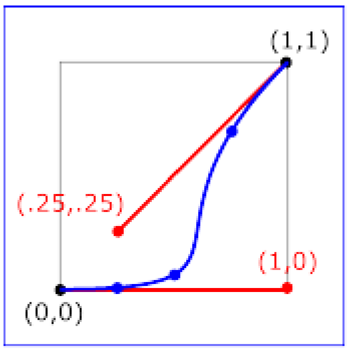
keySplines="0 0 1 1"
(the default)



keySplines=".5 0 .5 1"



keySplines="0 .75 .25 1"



keySplines="1 0 .25 .25"

To illustrate the calculations, consider the simple example:

```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
     calcMode="spline" keySplines="{as in table}"/>
```

Using the **'keySplines'** values for each of the four cases above, the approximate interpolated values as the animation proceeds are:

keySplines values	Initial value	After 1s	After 2s	After 3s	Final value
0 0 1 1	10.0	12.5	15.0	17.5	20.0
.5 0 .5 1	10.0	11.0	15.0	19.0	20.0
0 .75 .25 1	10.0	18.0	19.3	19.8	20.0
1 0 .25 .25	10.0	10.1	10.6	16.9	20.0

For a formal definition of Bézier spline calculation, see [FOLEY-VANDAM], pp. 488-491.

16.2.10 Attributes that control whether animations are additive

It is frequently useful to define animation as an offset or delta to an attribute's value, rather than as absolute values. A simple "grow" animation can increase the width of an object by 10 units:

```
<rect width="20" ...>
  <animate attributeName="width" from="0" to="10" 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="20" ...>
  <animate attributeName="width" from="0" to="10" dur="10s"
           additive="sum" accumulate="sum" repeatCount="5"/>
</rect>
```

At the end of the first repetition, the rectangle has a width of 30 units. At the end of the second repetition, the rectangle has a width of 40 units. At the end of the fifth repetition, the rectangle has a width of 70 units.

For more information about additive animations, see Additive animation ([SMIL21], section 3.3.6) and for more information on cumulative animations, see Cumulative animation ([SMIL21], section 3.3.5).

The following attributes are common to elements **'animate'**, **'animateMotion'**, **'animateColor'** and **'animateTransform'**.

Schema: animateaddition

```
<define name='svg.AnimateAdditionCommon.attr'>
  <optional>
    <attribute name='additive' svg:animatable='false' svg:inheritable='false'>
      <choice>
        <value>replace</value>
        <value>sum</value>
      </choice>
```

```
        </attribute>
      </optional>
      <optional>
        <attribute name='accumulate' 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 lacuna value, however the behavior is also affected by the animation value attributes **'by'** and **'to'**, as described in Simple animation functions specified by from, to and by ([SMIL21], section 3.5.4).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is found in the Animation effect function attributes section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.3).
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 lacuna value.

This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.
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 found in the Animation effect function attributes section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.3).
Animatable: no.

16.2.11 Inheritance

SVG allows both attributes and properties to be animated. If a given attribute or property is inheritable by descendants, then animations on a parent element such as a **'g'** element has the effect of propagating the attribute or property animation values to descendant elements as the animation proceeds; thus, descendant elements can inherit animated attributes and properties from their ancestors.

16.2.12 The **'animate'** element

The **'animate'** element is used to animate a single attribute or property over time. For example, to make a rectangle repeatedly fade away over 5 seconds, you can specify:

```
<rect>
  <animate attributeType="CSS" attributeName="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 found in The animate element section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.6.1).

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.

In the case where an **'animate'** element is animating a color value, the same calculation method as the **'animateColor'** element is used. That is, **'animate'** and **'animateColor'** have identical behavior when animating colors.

16.2.13 The **'set'** element

The **'set'** element provides a simple means of just setting the value of an attribute for a specified duration. It supports most attribute types, including those that cannot be interpolated, such as string and boolean values. The **'set'** element cannot perform additive or cumulative animation; the **'additive'** and **'accumulate'** attributes will be ignored if specified.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is found in The set element section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.6.2).

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 value must be of an appropriate type according to the target attribute being animated.

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.

16.2.14 The **'animateMotion'** element

The **'animateMotion'** element causes its target element to move along a motion path.

Any element that can take a **'transform'** attribute may have motion animations applied to it. See the transform attribute in the attribute table appendix for a list of these elements.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is found in The animateMotion element section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.6.3).

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>

    <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'/>
      <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 lacuna value of the **'calcMode'** attribute for **'animateMotion'** is **'paced'**.

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 post-multiply a supplemental transformation matrix onto the CTM for the target element which 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>"**

Takes a semicolon-separated list of floating point values between 0 and 1 and indicates how far along the motion path the target element 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 the **'keyPoints'** attribute has a value that doesn't conform to the above requirements, the attribute has an unsupported value and the animation element is processed as if the attribute had not been specified.

Animatable: no.

`rotate` = **"auto" | "auto-reverse" | "<number>"**

The **'rotate'** attribute post-multiplies a supplemental transformation matrix onto the CTM of the target element to apply a rotation transformation about the origin of the current user coordinate system. The rotation transformation is applied after the supplemental translation transformation that is computed due to the **'path'** attribute.

auto

Indicates that the target element is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path.

auto-reverse

Indicates that the target element is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path plus 180 degrees.

<number>

Indicates that the target element has a constant rotation transformation applied to it, where the rotation angle is the specified number of degrees.

The lacuna value is **'0'**.

Animatable: no.

`origin` = **"default"**

See the SMIL 2.1 definition of 'origin' ([SMIL21], section 3.6.3). It has no effect in SVG, and if specified, must take the value **'default'**.

Animatable: no.

For **'animateMotion'**, the specified values for **'from'**, **'by'**, **'to'** and **'values'** consist 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 consist 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"** specifies a list of three coordinate pairs. Each coordinate is represented by a <length>. Attributes **'from'**, **'by'**, **'to'** and **'values'** specify a shape in the target element's user coordinate system which represents the motion path.

Two options are available to define a motion path using SVG path data commands:

- a **'path'** attribute, which defines a motion path directly on an **'animateMotion'** element using SVG path data commands
- an **'mpath'** child element of the **'animateMotion'** element, which provides the ability to reference an external **'path'** element as the definition of the motion path

Note that any styling on an **'animateMotion'** or **'mpath'** element does not affect the defined path. For example, specifying a dashed stroke will not cause the animation to jump from dash to dash.

The various (*x*, *y*) points of the shape provide a supplemental transformation matrix to be post-multiplied onto the CTM of the target element, 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 target element 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 targetting that 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 interpolation mode 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 an **'animateMotion'** element has a **'calcMode'** of **'discrete'**, **'linear'** or **'spline'**, and if the **'keyPoints'** attribute is not specified, 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 data. A "move to" command within the path, i.e. other than as the first path data command, does not count as an additional point when dividing up the duration, or when associating **'keyTimes'**, **'keySplines'** and **'keyPoints'** values. When **'calcMode'** is **'paced'**, all "move to" commands are considered to have zero length (i.e. they always happen instantaneously), and are not considered when 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 motion path.

The override rules for **'animateMotion'** are as follows. Regarding the definition of the motion path, the **'mpath'** element overrides the **'path'** attribute, which overrides **'values'**, which overrides **'from'**/**'by'**/**'to'**. Regarding determining the points that correspond to the **'keyTimes'** attribute, the **'keyPoints'** attribute overrides **'path'** and **'mpath'**, which overrides **'values'**, which overrides **'from'**/**'by'**/**'to'**.

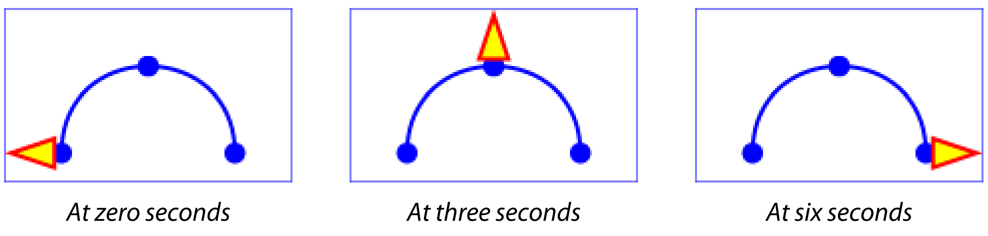
At any time *t* within a motion path animation of duration *dur*, the computed coordinate pair (*x*, *y*) along the motion path is determined by finding the point (*x*, *y*) which is (*t* / *dur*) distance along the motion path using the user agent's distance along a path algorithm.

The following example demonstrates the supplemental transformation matrices that are computed during a motion path animation.

Example 19_02 shows a triangle moving along a motion path.

Example: 19_02.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" width="5cm" height="3cm" viewBox="0 0 500 300">

  <desc>Example 19_02 - 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>
```



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)

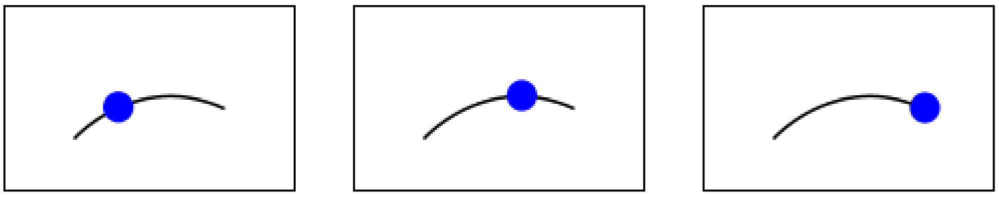
16.2.15 The 'mpath' element

The **'mpath'** element is used to reference an existing **'path'** element for use as a motion animation path. It may appear only as the child of an **'animateMotion'** element to which it provides the path.

Example mpath01 demonstrates the use of an **'mpath'** element that references an existing **'path'** element that is rendered. Without **'mpath'**, the path data would have to be repeated in a **'path'** attribute on the **'animateMotion'** element.

Example: mpath01.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" viewBox="0 0 80 60">

  <desc>Example demonstrating the use of the 'mpath' element.</desc>

  <path xml:id="mpathRef" d="M15,43 C15,43 36,20 65,33" fill="none" stroke="black" stroke-width="1"/>
  <circle r="5" fill="blue">
    <animateMotion begin="0s" dur="6s" calcMode="linear" fill="freeze">
      <mpath xlink:href="#mpathRef"/>
    </animateMotion>
  </circle>
</svg>
```


At two seconds *At four seconds* *At six seconds*

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>"**

An IRI reference to the **'path'** element which defines the motion path. An invalid IRI reference is an unsupported value. The lacuna value is the empty string. An empty attribute value results in the animation timing still running but no motion animation will be applied to the target element.

Animatable: no.

16.2.16 The 'animateColor' element

The **'animateColor'** element specifies a color transition over time.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is found in The animateColor element section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.6.4).

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 same syntax as is used by the **'solid-color'** property (a <color>, **currentColor** or **inherit**), or the additional value **none**. The **'values'** attribute takes a semicolon-separated list of color values using that same syntax.

Interpolated animation (that is, an animation where **'calcMode'** is **'linear'**, **'paced'** or **'spline'**) can be performed for color animations if none of the values involved is **'none'**. Otherwise, only discrete animation can be performed.

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.

16.2.17 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'**.

The animation effect for **'animateTransform'** is post-multiplied to the underlying value for additive **'animateTransform'** animations (see below) instead of added to the underlying value, due to the specific behavior of **'animateTransform'**.

From-to, *from-by* and *by animations* are defined by SMIL to be equivalent to a corresponding *values animations*. See the Simple animation functions specified by from, to and by section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.4). However, *to animations* are a mixture of additive and non-additive behavior, as described in the To animation section of the SMIL 2.1 Animation Modules ([SMIL21], section 3.5.4). *To animations* provide specific functionality to get a smooth change from the underlying value to the **'to'** attribute value, which conflicts mathematically with the requirement for additive transform animations to be post-multiplied. As a consequence, in SVG Tiny 1.2 the behavior of *to animations* for **'animateTransform'** is undefined. Authors are suggested to use *from-to*, *from-by*, *by* or *values animations* to achieve any desired transform animation.

If **'calcMode'** has the value **'paced'**, then the "distance" for the transformation is calculated as further described in Paced animations and complex types.

When an animation is active, the effect of a 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 an additive animation (i.e. **additive="sum"**) is to post-multiply the transformation matrix corresponding to the transformation defined by this **'animateTransform'** to the base set of transformations representing the underlying value. 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 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 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.

16.2.18 Attributes and properties that can be animated

Each attribute or property within this specification indicates whether or not it can be animated by 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 table below indicates which animation elements can be used to animate each of the basic data types.

If the computed value of a given attribute or property and the values specified in an animation can all be converted to numeric values, then additive animations are possible; however, if the animation or base value uses one or more keyword values which cannot be converted to a numeric value, additive animation is not possible.

In the example below, the **'fill'** property has an underlying value of **currentColor** and is animated from **red** to **#DDF**. The value of the **'color'** property is **yellow**.

```
<rect color="yellow" fill="currentColor">
  <animateColor attributeName="fill" from="red" to="#DDF"
                begin="1s" dur="5s" fill="freeze" additive="sum"/>
</rect>
```

The animation can be additive, because the computed value of the **'fill'** property is **yellow**, which can be converted to an RGB color which is a triple of numeric values (255, 255, 0); the keyword **red** can likewise be converted to an RGB color which is a triple of numeric values (255, 0, 0), as can the value **#DDF** which corresponds to (221, 221, 255). Had any of these values been keywords which could not be converted to a numeric representation, such as **none** or **url(#foo)**, then the animation could not have been additive.

Data type	Additive?	'animate'	'set'	'animate Color'	'animate Transform'	Notes
<color>	yes	yes	yes	yes	no	Only additive if each value can be converted to an RGB color.
<coordinate>	yes	yes	yes	no	no	
<integer>	yes	yes	yes	no	no	
<length>	yes	yes	yes	no	no	
<list-of-*Ts*>	no	yes	yes	no	no	
<number>	yes	yes	yes	no	no	
<paint>	yes	yes	yes	yes	no	Only additive if each value can be converted to an RGB color.
<transform>	yes	no	no	no	yes	Additive means that a transformation is post-multiplied to the base set of transformations representing the underlying value.
<IRI>	no	yes	yes	no	no	
All other data types used in animatable attributes and properties	no	yes	yes	no	no	

Any type listed in the table as being able to participate in an additive animation can also participate in a cumulative animation.

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.

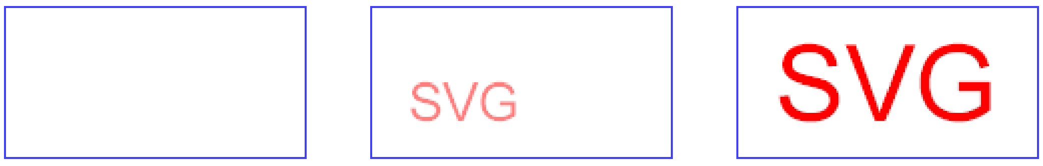
If an attribute or property value is modified while an additive animation element is animating the same attribute or property, the animation must adjust dynamically to the new value.

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 xmlns="http://www.w3.org/2000/svg"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny"
     width="4cm" height="2cm" viewBox="0 0 400 200">

  <desc>A simple animation using the uDOM and the Timer 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;

    // A listener for the timer
    function someListener(evt) {
      showAndGrowElement();
    }

    function init() {
      textElement = document.getElementById("svgtext");
      svgRoot = document.documentElement;

      launchTimer();
    }

    function launchTimer() {
      // Fire timer event as soon as possible, initialInterval = 0
      // Timer event must be sent every 50 ms, repeatInterval = 50
      someTimer = createTimer(0, 50);

      // Add a listener for the "SVGTimer" event
      someTimer.addEventListener("SVGTimer", someListener, false);

      // Start the timer, which will fire the first event immediately as initialInterval is 0
      someTimer.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) {
        someTimer.stop();
      }
    }
  ]]></script>
  <handler type="application/ecmascript" ev:event="load">
    init();
  </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:

- A unique **'handler'** is declared in the example to handle the `load` event:

  ```
      <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 drives the animation. 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()` function. `showAndGrowElement()` sets the **'transform'** attribute and **'fill-opacity** property on the **'text'** element to new values each time it is called.
- The `someListener` function is used as an `EventListener` object. In the `launchTimer()` function, `someListener` is passed as a parameter to the `addEventListener()` method of the timer object so that it is called each time an `SVGTimer` event is dispatched. (The `someTimer` object is the only target of this event). This way, it is possible to register a listener on a timer object using scripting.
- Because an `SVGTimer` event is dispatched regularly, `showAndGrowElement()` is called every 50 milliseconds and changes the text attributes thereby animating the text. `showAndGrowElement()` checks each time it is invoked whether the maximum duration of the animation has been reached and, if so, calls the `stop()` method on the timer object to stop the animation.

16.4 Animation and the bounding box

The effects of animation may change the bounding box of animated elements. For details on this, see the definition of the bounding box in the Coordinate Systems, Transformations and Units chapter.

17 Fonts

Contents

17.1 Introduction 242
17.1.1 Describing fonts available to SVG 242
17.1.2 Defining fonts in SVG 242

17.2 Overview of SVG fonts 243
17.3 The 'font' element 244
17.4 The 'glyph' element 244
17.5 The 'missing-glyph' element 248
17.6 Glyph selection rules 248
17.7 The 'hkern' element 249
17.8 Describing a font 250
17.8.1 Overview of font descriptions 250
17.8.2 The 'font-face' element 250
17.8.3 The 'font-face-src' element 255
17.8.4 The 'font-face-uri' element 255

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.

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>
      <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>
```

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.)

The lacuna value is **'0'**.

Animatable: no.

`horiz-adv-x` = **"<number>"**

The default horizontal advance after rendering a glyph in horizontal orientation. Glyph widths are required to be non-negative, even if the glyph is typically rendered right-to-left, as in Hebrew and Arabic scripts.

The lacuna value is **'0'**.

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'/>
      <optional>
        <attribute name='unicode' svg:animatable='false' svg:inheritable='false'><text/></attribute>
      </optional>
      <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='LanguageIDs.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 characters indicating the sequence of characters which corresponds to this glyph. If a single character is provided, then this glyph corresponds to the given Unicode character. If multiple characters are provided, then this glyph corresponds to the given sequence of Unicode characters. One use of a sequence of characters is ligatures. For example, if **unicode="ffl"**, then the given glyph will be used to render the sequence of characters "f", "f", and "l".

It is often useful to refer to characters using XML character references expressed in hexadecimal notation or decimal notation. For example, **unicode="ffl"** could be expressed as XML character references in hexadecimal notation as **unicode="&#x66;&#x66;&#x6c;"** or in decimal notation as **unicode="&#102;&#102;&#108;"**.

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` = **"<list-of-strings>"**

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` = **"<list-of-language-ids>"**

The attribute value is a comma-separated list of language tags as defined in IETF Best Current Practice 47 [BCP47]. For XML content, 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.

For example, a glyph suitable for French (whose lang attribute includes the language tag 'fr') is suitable for all types of French text (for example, Canadian French, **xml:lang="fr-ca"**). A glyph whose lang attribute is specific for Traditional Chinese (lang includes zh-Hant) is not suitable for Simplified Chinese (**xml:lang="zh-Hanc"**) *or* generic Chinese (**xml:lang="zh"**).

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 content 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 and is used by permission.

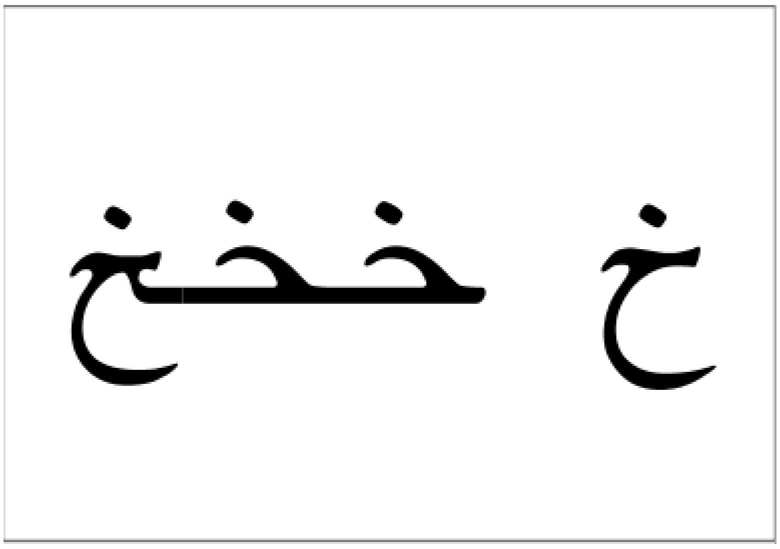
Example: font01.svg

```
<?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" xml: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 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 خ - 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 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 "ffl" 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 (sequences and/or ranges 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>
  <optional>
    <attribute name='k' svg:animatable='false' svg:inheritable='false'>
      <ref name='Number.datatype'/>
    </attribute>
  </optional>
</define>

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

Attribute definitions:

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

A sequence (comma-separated) of 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 = **"<list-of-strings>"**

A sequence 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 = **"[<Char> | <urange>] [, [<Char> | <urange>]]* "**

Same as the **'u1'** attribute, except that **'u2'** specifies possible second glyphs in the kerning pair.

Animatable: no.

g2 = **"<list-of-strings>"**

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. The lacuna value is **'0'**.

Animatable: no.

At least one of **'u1'** and **'g1'** must be provided, and at least one of **'u2'** and **'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 so that it can be used to select the relevant font data. The font description contains descriptors such as the location of the font data on the Web, and characterizations of that font data. The font descriptors are also needed to match the font properties to particular font data. The level of detail of a font description can vary from just the name of the font up to a list of glyph widths.

For more about font descriptions, refer to the Fonts chapter in the CSS2 specification [CSS2].

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>
      </optional>
      <optional><attribute name='font-style' svg:animatable='false'
svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='font-weight' svg:animatable='false'
svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='font-variant' svg:animatable='false'
svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='font-stretch' svg:animatable='false'
svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='unicode-range' svg:animatable='false'
svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='panose-1' svg:animatable='false' svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='widths' svg:animatable='false' svg:inheritable='false'><text/></attribute>
      </optional>
      <optional><attribute name='bbox' svg:animatable='false' svg:inheritable='false'><text/></attribute>
      </optional>
      <optional>
        <attribute name='units-per-em' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='stemv' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='stemh' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='slope' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='cap-height' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='x-height' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='accent-height' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='ascent' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='descent' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='ideographic' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='alphabetic' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='mathematical' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='hanging' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='underline-position' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </attribute>
      </optional>
      <optional>
        <attribute name='underline-thickness' svg:animatable='false' svg:inheritable='false'>
          <ref name='Number.datatype'/>
        </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.

The lacuna value is **'all'**.

Animatable: no.

`font-weight` = **"all | [normal | bold |100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900] [, [normal | bold |100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900]]*"**

Same syntax and semantics as the **'font-weight'** descriptor within an @font-face rule.

The weight of a face relative to others in the same font family. Takes on the same values as the 'font-weight' property with three exceptions:

- relative keywords (bolder, lighter) must not be used
- a comma-separated list of values may be used, for fonts that contain multiple weights
- 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.

The lacuna value is **'all'**.

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].

The lacuna value is **'U+0-10FFFF'**.

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.

The lacuna value is **'1000'**.

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 white space. Except for any additional information provided in this specification, the normative definition of the attribute is in [CSS2].

The lacuna value is **'0 0 0 0 0 0 0 0 0 0'**.

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].

The lacuna value is **'0'**.

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'** 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 CSS 2 specification ([CSS2], sections 15.3.1 and 15.3.5).

A **'font-face-src'** element contains a **'font-face-uri'** 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'/>
    </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'/>
  <ref name='svg.External.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 to the the font.

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>
```

SVG

18 Metadata

Contents

18.1 Introduction 258
18.2 The 'metadata' element 258
18.2.1 A 'metadata' element example 259

18.3 Extensible metadata attributes 259

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 RDF (Resource Description Framework). The set of six specifications for RDF can be found starting with the 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.).

A popular usage of RDF metadata in the SVG community is the inclusion of license terms using the Creative Commons [CC] license framework. The popular open-source SVG authoring tool Inkscape can automatically include this license information, and inclusion of such a license is required by many collaborative Web sites. This indicates the potential and impact of metadata.

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.

SVG provides two mechanisms for adding metadata directly to a document: the **'metadata'** element, and several metadata attributes. These different mechanisms may be used independently, or in concert. If both are being used for RDF (that is, RDF and RDFa), then an RDF processor should combine them into the same graph. (Note: metadata attributes should not be used directly on RDF elements.)

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 either the *Namespaces in XML 1.0* or *Namespaces in XML 1.1* Recommendations [XML-NS10][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.

If a **'metadata'** element is placed as a child of a non-root element with the intent to apply directly to that element, it is recommended that the author indicate this explicitly in the content of the **'metadata'** element, if that content provides a way to do so. For example, when using RDF, the target element should be given an **'id'** attribute, and the **'about'** attribute of the RDF should reference that **'id'**.

Schema: metadata

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

18.2.1 A 'metadata' element example

Here is an example of how metadata can be included in an SVG document. The example uses the Creative Commons schema to indicate the usage license of a work of art. (Other XML-compatible metadata languages, including ones not based on RDF, can be used also.)

Example: metadata-license.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.2" baseProfile="tiny"
     width="8in" height="7in">

  <title>Shiny Donkey</title>

  <desc>
    A drawing of a brown donkey with a white nose, sitting on
    his haunches.  Made with vector paths enhanced with filter
    effects to add a rounded contour.
  </desc>

  <metadata id="license"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:cc="http://creativecommons.org/ns#">
    <rdf:RDF>
      <cc:Work rdf:about="">
        <dc:format>image/svg+xml</dc:format>
        <dc:type rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
        <cc:license rdf:resource="http://creativecommons.org/licenses/by-sa/3.0/" />
      </cc:Work>
      <cc:License rdf:about="http://creativecommons.org/licenses/by-sa/3.0/">
        <cc:permits rdf:resource="http://creativecommons.org/ns#Reproduction" />
        <cc:permits rdf:resource="http://creativecommons.org/ns#Distribution" />
        <cc:requires rdf:resource="http://creativecommons.org/ns#Notice" />
        <cc:requires rdf:resource="http://creativecommons.org/ns#Attribution" />
        <cc:permits rdf:resource="http://creativecommons.org/ns#DerivativeWorks" />
        <cc:requires rdf:resource="http://creativecommons.org/ns#ShareAlike" />
      </cc:License>
    </rdf:RDF>
  </metadata>

  <!-- graphical elements -->

</svg>
```

18.3 Extensible metadata attributes

In addition to the **'metadata'** element, which allows for the direct inclusion of metadata document fragments from other namespaces, SVG includes several attributes that may be placed on any element, for the use of attribute-based metadata formats. These include the **'class'**, **'role'**, **'rel'**, **'rev'**, **'about'**, **'content'**, **'datatype'**, **'property'**, **'resource'**, and **'typeof'** attributes. SVG makes no specific requirements about the values for these attributes, other than their particular value data types, such as a string or a space-separated lists of strings. Other specifications, such as RDFa [RDFA], Microformats patterns [MF], or ARIA ontologies [ARIA], may impose stricter requirements in order to conform to that particular language, which should be expressed as an additional schema for purposes of validation. A few informative examples of such restrictions include:

- The **'class'** attribute, when used for CSS selector names, allow only specific characters and cannot start with a digit, but allow any other combination of those characters, and has no keyword restrictions; when used with

Microformats, only particular keywords will be recognized; for other purposes, neither of these restrictions may apply.

- The **'role'** attribute, to be used with WAI-ARIA [ARIA], must match one or more values designated as role names in that specification. In this case, WAI-ARIA defines additional processing for the elements bearing these **'role'** values, including requirements for the use of further WAI-ARIA-defined attributes with value pertinent to the designated roles. Note that this processing does not interfere with host language processing, but supplements it with regard to the mapping of content to accessibility APIs.
- For the **'rel'** and **'rev'** attributes, both RDFa and Microformats require that the values must contain one of a set of predefined reserved keywords, though these keywords are defined in different ontologies. RDFa enforces the further requirement that each value string must be a **CURIE** (i.e., a reserved keyword that may or may not be a prefixed string, such as **'cc:license'** to indicate a Creative Commons license; [RDFA], section 7), while Microformats does not use CURIEs or namespace-prefixed values. Since SVG requires only that the attribute value is a space-separated list of strings, there is no conflict with either of these formats, and they may be used independently, or in combination provided that the keywords do not clash and that their processors are able to ignore unknown values.
- Specific keywords that are part of no formal language or format, such as the **'rel'** value **'nofollow'** intended for processing by search engines, require only that the keyword be included.

SVG does not mandate or require support for any of these languages, formats, or ontologies, but it includes these metadata attributes to enable their use as the author desires. Note that this specification's description of these complementary formats is intended for illustrative purposes only, and does not impose additional restrictions on those formats.

In order to maintain consistency and simplicity of implementation, and prevent conflict or ambiguity of use, if an author or a non-RDFa format reuses the **'about'**, **'content'**, **'datatype'**, **'property'**, **'resource'**, or **'typeof'** attributes, it is recommended that this is done in a manner consistent with the RDFa Syntax Processing Rules [RDFA].

Currently, many SVG documents contain RDF metadata that provides a title in the Dublin Core [DCORE] namespace, which is useful for certain types of processing, but which is not treated as a title for the document by many user agents. It is recommended that authoring tools which embed RDF metadata that contains a title also provide at least a document-level **'title'** element. The following example shows how the **'property'** attribute may be combined with the descriptive elements to supply both a human- and machine-readable title, in the manner of RDFa.

Example: metadata-dc-title.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:dc="http://purl.org/dc/elements/1.1/">

  <title property="dc:title">Metamorphosis I-IV</title>
  <desc property="dc:creator">M.C. Escher</desc>

  <!-- graphical elements -->

</svg>
```

19 Extensibility

Contents

19.1 Foreign namespaces and private data . . . 261
19.2 Embedding foreign object types . . . 261
19.2.1 The 'foreignObject' element . . . 262
19.2.2 Examples of 'foreignObject' . . . 263

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 the Embedding foreign object types section, below.)

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 xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="4in" height="3in">

  <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 [MATHML].

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.

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.

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.

The **'foreignObject'** element has two ways of including foreign content. One is to reference external content by using the **'xlink:href'** attribute, the other is to include child content of the **'foreignObject'** element. When the **'xlink:href'** attribute is specified the child content of the **'foreignObject'** element must not be displayed.

All mouse events 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.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.FocusHighlight.attr'/>
  <ref name='svg.Transform.attr'/>
  <ref name='svg.XYWH.attr'/>
</define>

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

  The lacuna value is **'0'**.

  Animatable: yes.

`y` = **"<coordinate>"**
: The y-axis coordinate of one corner of the rectangular region into which the referenced document is placed.

  The lacuna value is **'0'**.

  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 visual rendering of the element. The lacuna value is **'0'**.

  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 visual rendering of the element. The lacuna value is **'0'**.

  Animatable: yes.

`xlink:href` = **"<IRI>"**
: An IRI reference. If this attribute is present, then the foreign content must be loaded from this resource and what child content the **'foreignObject'** element may have must not be displayed. If this attribute is not present then the **'foreignObject'** child content must be displayed if the user agent is capable of handling it.

  Animatable: yes.

`focusable` = **"true" | "false" | "auto"**
: See attribute definition for description.

  Animatable: yes.

`Navigation Attributes`
: See definition.

19.2.2 Examples of 'foreignObject'

Here are several examples using a switch and the **'foreignObject'** element.

This is an example of an arbitrary XML language, with comments to explain what happens at each step:

Example: requiredExtensions-foreignObject.svg

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

  <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.  The text wraps within the confines
      of the element's dimensions.
    </textArea>

  </switch>
</svg>
```

This is an example of MathML in SVG:

Example: requiredExtensions-MathML.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 600 500">

  <title>Quadratic Equation</title>
  <desc>
    A sample of MathML in SVG, using the 'foreignObject' element
    to represent a quadratic equation, with a graphical SVG
    representation for fallback.
  </desc>

  <switch>
    <foreignObject x="20" y="20" width="600" height="500"
                   requiredExtensions="http://www.w3.org/1998/Math/MathML">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mrow>
          <mrow>
            <mi>f</mi>
            <mfenced>
                <mi>x</mi>
            </mfenced>
          </mrow>
          <mo>=</mo>
          <mrow>
            <msup>
              <mi>x</mi>
              <mn>2</mn>
            </msup>
            <mo>+</mo>
            <mrow>
              <mn>4</mn>
              <mi>x</mi>
            </mrow>
            <mo>-</mo>
            <mrow>
              <mn>3</mn>
            </mrow>
          </mrow>
        </mrow>
      </math>
    </foreignObject>

    <g fill="gray" transform="translate(300,250)">
      <rect x="-300" y="-250" width="600" height="500" fill="white" stroke="gray" />
      <g id="axes" font-family="monospace" text-anchor="middle">
        <line id="x-axis" x1="-300" y1="0" x2="300" y2="0" stroke="gray"/>
        <line id="x-axis-markers" x1="-300" y1="0" x2="300" y2="0"
              stroke="gray" stroke-width="7" stroke-dasharray="1,99"/>
        <line id="y-axis" x1="0" y1="-250" x2="0" y2="250" stroke="gray"/>
        <line id="y-axis-markers" x1="0" y1="-200" x2="0" y2="250"
              stroke="gray" stroke-width="7" stroke-dasharray="1,99"/>

        <text x="-200" y="20" font-size="10">-4</text>
        <text x="-100" y="20" font-size="10">-2</text>
        <text x="100" y="20" font-size="10">2</text>
        <text x="200" y="20" font-size="10">4</text>

        <text x="15" y="-198" font-size="10">4</text>
        <text x="15" y="-98" font-size="10">2</text>
        <text x="15" y="102" font-size="10">-2</text>
        <text x="15" y="202" font-size="10">-4</text>
      </g>

      <path id="graph" stroke-width="1" stroke="blue" fill="none"
            d="M-200,-250 Q-50,650 100,-250"/>

      <circle id="vertex" cx="-50" cy="200" r="2" fill="blue" />
      <circle id="y-intercept-1" cx="0" cy="150" r="2" fill="red" />
      <circle id="x-intercept-1" cx="-150" cy="0" r="2" fill="red" />
      <circle id="x-intercept-2" cx="50" cy="0" r="2" fill="red" />
    </g>
  </switch>
</svg>
```

This is an example of XHTML in SVG:

Example: requiredExtensions-XHTML.svg

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 300 140">

  <title>Chinese-English Unicode Table</title>
  <desc>
    A sample of XHTML in SVG, using the 'foreignObject' element
    to include an XHTML 'table' with some Chinese-to-English
    correspondances, with an ad-hoc SVG representation for
    fallback.
  </desc>

  <switch>
    <foreignObject width="300" height="140"
                   requiredExtensions="http://www.w3.org/1999/xhtml">
      <table xmlns="http://www.w3.org/1999/xhtml">
         <caption>Using Chinese Characters in SVG</caption>
         <tr>
           <th>English</th>
           <th>Chinese</th>
         </tr>
         <tr y="75">
           <td>moon</td>
           <td>&#x6708;</td>
           <td>6708</td>
         </tr>
         <tr y="100">
           <td>tree</td>
           <td>&#x6728;</td>
           <td>6728</td>
         </tr>
         <tr y="125">
           <td>water</td>
           <td>&#x6c34;</td>
           <td>6c34</td>
         </tr>
       </table>
    </foreignObject>

    <text font-size="18" fill="crimson" text-anchor="middle">
      <tspan x="150" y="25" font-weight="bold">Using Chinese Characters in SVG</tspan>
      <tspan y="50">
```

```
        <tspan x="50">English</tspan>
        <tspan x="150">Chinese</tspan>
        <tspan x="250">Unicode</tspan>
      </tspan>
      <tspan y="75">
        <tspan x="50">moon</tspan>
        <tspan x="150">&#x6708;</tspan>
        <tspan x="250">6708</tspan>
      </tspan>
      <tspan y="100">
        <tspan x="50">tree</tspan>
        <tspan x="150">&#x6728;</tspan>
        <tspan x="250">6728</tspan>
      </tspan>
      <tspan y="125">
        <tspan x="50">water</tspan>
        <tspan x="150">&#x6c34;</tspan>
        <tspan x="250">6c34</tspan>
      </tspan>
    </text>
  </switch>
</svg>
```

A The SVG Micro DOM (uDOM)

Contents

A.1 Introduction 268
A.2 Overview of the SVG uDOM. . . . 268
A.2.1 Document access 269
A.2.2 Tree navigation 269
A.2.3 Element creation. . . . 269
A.2.4 Element insertion 269
A.2.5 Element removal. . . . 269
A.2.6 Attribute and property access 270
A.2.7 Event listener registration and removal 270
A.2.8 Animation 271
A.2.9 Multimedia control 271
A.2.10 Java package naming 271

A.3 Conforming to the SVG uDOM 271
A.3.1 Float values. . . . 271
A.3.2 Attribute/property normalization. . . . 271
A.3.3 Text content access 272

A.4 Module: dom. . . . 273
A.4.1 DOMException 273
A.4.2 Node 274
A.4.3 Element 275
A.4.4 Document 275
A.4.5 ElementTraversal 276
A.4.6 Location 276
A.4.7 Window 277

A.5 Module: views. . . . 278
A.5.1 AbstractView 278
A.5.2 DocumentView 278

A.6 Module: events. . . . 279
A.6.1 EventTarget 279
A.6.2 EventListener 279
A.6.3 Event 280
A.6.4 MouseEvent 280
A.6.5 MouseWheelEvent 281
A.6.6 TextEvent. . . . 281
A.6.7 KeyboardEvent 282
A.6.8 UIEvent. . . . 286
A.6.9 ProgressEvent 287

A.7 Module: smil 288
A.7.1 ElementTimeControl 288
A.7.2 TimeEvent 289

A.8 Module: svg. . . . 290
A.8.1 SVGException. . . . 290
A.8.2 SVGDocument 290
A.8.3 SVGUseElement. . . . 290
A.8.4 SVGElementInstance 291
A.8.5 SVGSVGElement 292

A.8.6 SVGRGBColor 296
A.8.7 SVGRect 297
A.8.8 SVGPoint 297
A.8.9 SVGPath 298
A.8.10 SVGMatrix 300
A.8.11 SVGLocatable. 302
A.8.12 SVGLocatableElement 306
A.8.13 TraitAccess 306
A.8.14 Additional accessing rules 322
A.8.15 SVGElement 323
A.8.16 SVGTimedElement. 324
A.8.17 SVGAnimationElement. 324
A.8.18 SVGVisualMediaElement 324
A.8.19 SVGTimer. 325
A.8.20 SVGGlobal 326
A.8.21 AsyncStatusCallback 329
A.8.22 AsyncURLStatus 329
A.8.23 EventListenerInitializer2 330

This appendix is normative.

A.1 Introduction

During the later stages of development of the SVG Mobile 1.1 specification [SVGM11] it became obvious that there was a requirement to subset the SVG and XML DOM in order to reduce the burden on implementations. SVG Tiny 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 compared to the traditional SVG DOM, 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). A subset of the uDOM (corresponding to SVG Tiny 1.1) is already successfully implemented by various implementations of *JSR 226: Scalable 2D Vector Graphics API for J2ME*, compatibility with which is another goal of the uDOM [JSR226].

The uDOM makes normative reference to DOM Level 2 Events [DOM2EVENTS], and informative reference to DOM Level 3 Events [DOM3EVENTS]. A minimal subset of DOM Level 3 Events was included in the uDOM to specify functionality as currently implemented on mobile devices, since DOM Level 3 Events was not yet a Recommendation at the time of publication. It is anticipated that DOM Level 3 Events may change to reflect the needs of the current Web environment, and any conflicting changes will supersede the functionality specified here for later SVG specifications.

The IDL definition for the uDOM is provided.

This appendix consists of the following parts:

- Overview of the SVG uDOM — an informative introduction to the uDOM, including a summary of supported features and descriptions by topic of the key features and constraints together with examples.
- Conforming to the SVG uDOM — a normative section that states conformance criteria for the SVG uDOM as well as descriptions of key features.
- A normative definition of all the interfaces in the SVG uDOM (DOM Core APIs, DOM Views APIs, SMIL DOM APIs, DOM Events APIs and SVG DOM APIs).

A.2 Overview of the SVG uDOM

The following sections provides an informative 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 contains ECMAScript and Java language examples, the SVG uDOM is compatible with other programming languages.

A.2.1 Document access

The SVG uDOM offers 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. In some languages, such as Java, the `Document` object can be obtained by implementing the `EventListenerInitializer2` interface. The SVG user agent will invoke the implementation's `initializeEventListeners` method once the script has been loaded and is ready to bind to the document. The `Document` object is sometimes accessible through other means, for example through the `AbstractView::document` member which is available on the global object in ECMAScript.

A.2.2 Tree navigation

The SVG uDOM only allows navigation of the document node and the element nodes in the DOM tree. Two options are available for navigating the hierarchy of elements:

- Individual element nodes with an ID value are accessible directly via the `getElementById` method on the `Document` interface.
- The hierarchy of element nodes are 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 [ET]. These traversal mechanisms skip over intervening nodes between element nodes, such as text nodes which might only contain white space.

A.2.3 Element creation

The SVG uDOM allows the creation of new elements using the `createElementNS` method of the `Document` interface.

Example: Element creation (Java)

```
String svgNS = "http://www.w3.org/2000/svg";
Element myRect = document.createElementNS(svgNS, "rect");
```

A.2.4 Element insertion

Elements can be inserted into the document tree by calling the `appendChild` or `insertBefore` methods on the `Node` that is to be the parent.

Example: Element insertion (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

An element can be removed from the document tree by calling the `removeChild` method on its parent `Node`.

Example: Element removal (ECMAScript)

```
var myRect = ...;
var myGroup = document.getElementById("myGroup");
myGroup.appendChild(myRect);
```

```
...
myGroup.removeChild(myRect);
```

A.2.6 Attribute and property access

The SVG Tiny 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 the typed value (e.g. a number, not just a string), associated with an element by an XML attribute or a CSS property. 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`, in contrast with the `getAttributeNS` method which always returns a string. The trait facilities in the SVG uDOM are available on the `TraitAccess` interface, which is implemented by all DOM objects representing SVG elements.

Example: Trait Access (Java)

```
float width = myRect.getFloatTrait("width");
width += 10;
myRect.setFloatTrait("width", width);
```

An important difference between `getTraitNS` (along with all other trait getter 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/property normalization).

Example: Difference between `getTraitNS` and `getAttributeNS`

```
<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>
```

In the above example:

- `r1.getTraitNS(null, "fill")` returns `"red"` (or an equivalent normalized form, see Attribute/property normalization).
- `r2.getTraitNS(null, "fill")` returns `"red"` (or an equivalent normalized form, see Attribute/property normalization).
- `r1.getAttributeNS(null, "fill")` returns `""`.
- `r2.getAttributeNS(null, "fill")` returns `"inherit"`.

Traits may also be animated, by animating the underlying XML attribute or property. To access the animated value of a trait, the `getPresentationTrait`, along with the other similarly named presentation trait getter methods on the `TraitAccess` interface, can be used.

A.2.7 Event listener registration and removal

The SVG uDOM utilizes DOM Level 2 Events, using the `EventTarget` interface, to support the ability to add and remove event listeners to nodes in a document.

Example: Event Listeners (Java)

```
class MyEventListener implements EventListener {
    public void handleEvent(Event evt) {
        // Do whatever is needed here
    }
}
...

// Create a listener
EventListener listen1 = new MyEventListener();

// Listen to click events, during the bubbling phase
SVGElement myRect = (SVGElement)document.getElementById("myRect");
myRect.addEventListener("click", listen1, false);
```

```
...

// Remove the click listener
myRect.removeEventListener("click", listen1, false);
```

A.2.8 Animation

Animation elements can be started and stopped using the methods available on the `ElementTimeControl` interface.

Example: animation (ECMAScript)

```
var animateColor = document.getElementById("myAnimation");

// Start the animation 2.5 seconds from now.
animateColor.beginElementAt(2.5);
```

A.2.9 Multimedia control

Control of multimedia elements, such as the **'audio'**, **'video'**, and **'animation'** elements is available through a combination of the `ElementTimeControl` and `SVGTimedElement` interfaces. Some common controls, and the interface methods to access them, are listed below:

- **play:** beginElement method of the `ElementTimeControl` interface
- **stop:** endElement method of the `ElementTimeControl` interface
- **pause:** pauseElement method of the `SVGTimedElement` interface
- **unpause:** resumeElement method of the `SVGTimedElement` interface
- **seek:** beginElementAt method of the `ElementTimeControl` interface (for example, a line graphic could be used to represent the timeline of a video, and a click event on a certain point on the line could serve as the offset for the new begin time of the video).

Note that SVG 1.2 Tiny does not define controlling the rate of playback (such as fast-forward or reverse) for time container elements. This functionality may be included in a future specification.

A.2.10 Java package naming

The SVG uDOM uses the same Java package names as the upcoming 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 Conforming to the SVG uDOM

This section and all the following are normative. Conforming SVG Viewers must support all constants, attributes and methods of all the interfaces defined in the SVG uDOM unless an interface explicitly allows for exceptions to this rule.

A.3.1 Float values

The SVG uDOM uses IEEE-754 single precision floating point values to represent `float` values in the IDL [IEEE-754]. While such values support a number of non-finite values — a set of NaN (Not a Number) values and positive & negative infinity — these values are never used by the uDOM. Thus, unless otherwise specified in the prose for an operation or attribute, a `DOMException` with error code NOT_SUPPORTED_ERR must be thrown if a non-finite value is passed as an operation argument, or assigned to an attribute, whose type is `float`, or if a list of floating point values containing a non-finite value is passed as an operation argument, or assigned to an attribute, whose type is `sequence<float>`.

In addition, none of the operations or attributes in the uDOM distinguish between positive and negative zero. A negative zero must be treated as a positive zero when passed as an operation argument, or assigned to an attribute, whose type is `float` or `sequence<float>`.

Operations and attributes in the uDOM will never return a non-finite or negative zero value from an operation or attribute.

A.3.2 Attribute/property normalization

A viewer implementing the uDOM is allowed to return normalized attribute values ([DOM3], section 1.4) from `getAttributeNS` and the various trait getter methods (`getTrait`, `getTraitNS`, `getFloatTrait`, etc.) and trait presentation value getter methods (`getPresentationTrait`, `getPresentationTraitNS`, `getFloatPresentationTrait`, etc.). The following is a list of possible attribute normalizations:

Color normalization
: **"red"** may be returned as **"rgb(255,0,0)"**, **"#ff0000"**, or another 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 another semantically identical form.

Whitespace normalization
: **" 3.0 "** may be returned as **"3.0"**. Whitespace normalization also includes unquoted font names in the **'font-family'** property. Font family names containing whitespace should be quoted. If quoting is omitted, any whitespace characters before and after the font name may be ignored and any sequence of whitespace characters inside the font name may be converted to a single space.

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(2,0,0,2,0,0)"**, and **"scale(2,2) translate(10,5) rotate(45)"** may be returned as **"matrix(1.4142, 1.4142, -2.5857, 1.4142, 20, 10)"**.

Path normalization
: The full set of path data comamnds as used by **'d'** and **'path'** may be mapped down to a smaller set of commands.
- Relative commands (c, h, l, m, q, s, t, v, and z) 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 C.
- A command Z is always normalized to command Z, even in the cases where an implicit lineto is added before the path is joined.

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 SVG Tiny 1.2 viewer. For example, **"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 SVG Tiny 1.2 viewer intended for use in a multiple namespace environment is strongly recommended to keep the full range of **'display'** values.

A.3.3 Text content access

In the SVG uDOM, there are two alternative ways to access an element's textual content. 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 [JSR226]) and is still available in order to keep backward compability. 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 text content, **'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 child **'tspan'** elements (i.e. **'tspan'** elements that are children of the element which text content is retrieved) will be lost if a text string is retrieved from an element and then set back again.

The `#text` trait is included for compatibility with the JSR 226 specification [JSR226]. 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.4 Module: dom

A.4.1 DOMException

An exception that occurred due to a DOM operation, as defined in the Fundamental Interfaces: Core Module section of *DOM Level 3 Core* ([DOM3], section 1.4). Note that since the SVG uDOM is a subset of DOM Level 3 Core, some of the exception codes defined for this exception may never occur (such as `INUSE_ATTRIBUTE_ERR`, and `VALIDATION_ERR`). However, in the interest of facilitating implementations that support both the uDOM and the complete DOM Level 3 Core, none of the exception codes are removed.

IDL Definition

```
exception DOMException
{
        unsigned short code;
};

// ExceptionCode
const unsigned short      INDEX_SIZE_ERR                 = 1;
const unsigned short      DOMSTRING_SIZE_ERR             = 2;
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_DATA_ALLOWED_ERR            = 6;
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      INUSE_ATTRIBUTE_ERR            = 10;
const unsigned short      INVALID_STATE_ERR              = 11;
const unsigned short      SYNTAX_ERR                     = 12;
const unsigned short      INVALID_MODIFICATION_ERR       = 13;
const unsigned short      NAMESPACE_ERR                  = 14;
const unsigned short      INVALID_ACCESS_ERR             = 15;
const unsigned short      VALIDATION_ERR                 = 16;
const unsigned short      TYPE_MISMATCH_ERR              = 17;
```

Constants

INDEX_SIZE_ERR
See `INDEX_SIZE_ERR`.

DOMSTRING_SIZE_ERR
See `DOMSTRING_SIZE_ERR`.

HIERARCHY_REQUEST_ERR
See `HIERARCHY_REQUEST_ERR`.

WRONG_DOCUMENT_ERR
See `WRONG_DOCUMENT_ERR`.

INVALID_CHARACTER_ERR
See `INVALID_CHARACTER_ERR`.

NO_DATA_ALLOWED_ERR
See `NO_DATA_ALLOWED_ERR`.

NO_MODIFICATION_ALLOWED_ERR
See `NO_MODIFICATION_ALLOWED_ERR`.

NOT_FOUND_ERR
See `NOT_FOUND_ERR`.

NOT_SUPPORTED_ERR
See NOT_SUPPORTED_ERR.
INUSE_ATTRIBUTE_ERR
See INUSE_ATTRIBUTE_ERR.
INVALID_STATE_ERR
See INVALID_STATE_ERR.
SYNTAX_ERR
See SYNTAX_ERR.
INVALID_MODIFICATION_ERR
See INVALID_MODIFICATION_ERR.
NAMESPACE_ERR
See NAMESPACE_ERR.
INVALID_ACCESS_ERR
See INVALID_ACCESS_ERR.
VALIDATION_ERR
See VALIDATION_ERR.
TYPE_MISMATCH_ERR
See TYPE_MISMATCH_ERR.

No defined attributes
No defined methods

A.4.2 Node

The Node interface describes generic nodes in an SVG document tree.
This interface is a subset of the Node interface defined in DOM Level 3 Core ([DOM3], section 1.4). Node types that must be supported in the uDOM are Element nodes and Document nodes.

This subset does not support the NodeType and DocumentPosition definition groups, since the nodeType field and the compareDocumentPosition method are not members of the subsetted interface.

Concerning textContent, there is no requirement to create a Text node on setting since this subset has no interface representing Text nodes. However, the behaviour of textContent must be as if the Text node described in the the definition of textContent had indeed been created.

An alternate way of accessing text content on elements defined within the SVG specification is with the use of the #text trait.

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
See namespaceURI.
localName
See localName.

parentNode
See `parentNode`.

ownerDocument
See `ownerDocument`.

textContent
See `textContent`.

Methods

appendChild
See `appendChild`.

insertBefore
See `insertBefore`.

removeChild
See `removeChild`.

cloneNode
See `cloneNode`.

A.4.3 Element

The `Element` interface describes generic elements in an SVG document tree.
This interface is a subset of the `Element` interface defined in DOM Level 3 Core ([DOM3], section 1.4).

Concerning `setAttributeNS`, there is no requirement to take the `prefix` into account since neither the `prefix` field nor the `Attr` interface are supported.

IDL Definition

```
interface Element : Node, ElementTraversal
{
        DOMString getAttributeNS(in DOMString namespaceURI, in DOMString localName)
                raises(DOMException);
        void setAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName, in DOMString value)
                raises(DOMException);
        DOMString getAttribute(in DOMString name);
        void setAttribute(in DOMString name, in DOMString value)
                raises(DOMException);
};
```

No defined constants

No defined attributes

Methods

getAttributeNS
See `getAttributeNS`.

setAttributeNS
See `setAttributeNS`.

getAttribute
See `getAttribute`.

setAttribute
See `setAttribute`.

A.4.4 Document

The `Document` interface represents XML documents.
This interface is a subset of the `Document` interface defined in DOM Level 3 Core ([DOM3], section 1.4).

IDL Definition

```
interface Document : Node
{
        Element createElementNS(in DOMString namespaceURI, in DOMString qualifiedName)
                raises(DOMException);
        readonly attribute Element documentElement;
        Element getElementById(in DOMString elementId);
};
```

No defined constants

Attributes

documentElement

See `documentElement`.

Methods

createElementNS

See `createElementNS`.

getElementById

See `getElementById`.

A.4.5 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, including elements in foreign namespaces. For the normative definition of this interface see the ElementTraversal specification [ET]; it is only repeated informatively below.

IDL Definition

```
interface ElementTraversal
{
        readonly attribute Element firstElementChild;
        readonly attribute Element lastElementChild;
        readonly attribute Element nextElementSibling;
        readonly attribute Element previousElementSibling;
        readonly attribute unsigned long childElementCount;
};
```

No defined constants

Attributes

firstElementChild

See `firstElementChild`.

lastElementChild

See `lastElementChild`.

nextElementSibling

See `nextElementSibling`.

previousElementSibling

See `previousElementSibling`.

childElementCount

See `childElementCount`.

No defined methods

A.4.6 Location

Location objects provide a representation of their document's address.

IDL Definition

```
interface Location
{
        void assign(in DOMString iri);
        void reload();
};
```

No defined constants
No defined attributes
Methods

assign

When this method is invoked, the user agent must navigate to the given IRI. The result of the traversal must be identical to the traversal caused by an **'a'** hyperlink with the **'target'** attribute set to **'_replace'**. The difference is that the **'a'** hyperlink is activated on user interaction but `assign` is activated from script. The current document location is the IRI of the `Document` object pointed to by the `AbstractView.document` field. Relative IRI references are resolved based on the base IRI of the current document. If the base IRI differs from that of the current document, the current document is discarded, and loading and parsing of the document at the specified IRI then begins. If the previous step resulted in loading of a new document, the timeline is restarted and a new load event is fired. Note: For HTTP, a pragma:no-cache ([RFC2616], section 14.32) is not issued and thus a fresh copy from the server is not forced if there is a cache.

Parameters

in `DOMString` iri	The IRI to be traversed.

No return value
No exceptions

reload

When this method is invoked, the user agent is forced to reload the resource identified by the `Location`. The current document location is the IRI of the `Document` object pointed to by the `AbstractView.document` field.

No parameters
No return value
No exceptions

A.4.7 Window

This is a subset of the de facto standard Window interface that many browsers implement. See *Window Object 1.0* and The default view in *HTML 5* for ongoing standardization efforts in this area at the time of writing [WINDOW, HTML5].

The `Window` interface must be implemented by the object that represents the default view of the document ([DOM2VIEWS], section 1.1). This object also implements `AbstractView`. Thus, in the ECMAScript language binding, the global script object implements `Window`. The `Window` object for a document can also be obtained through `DocumentView::defaultView`.

IDL Definition

```
interface Window
{
        readonly attribute Window parent;
        readonly attribute Location location;
};
```

No defined constants

Attributes

parent

The `Window` object that is the parent view of this document's default view. If the `Window` has no notion of parent (e.g. if the document is displayed as the top level document in a viewer), then the value of this attribute is `null`.

location

The `Location` object that is for that Window object's active document.

No defined methods

A.5 Module: views

SVG Tiny 1.2 requires complete *DOM Level 2 Views* support, which includes the `AbstractView` and `DocumentView` interfaces [DOM2VIEWS].

The SVG Tiny 1.2 uDOM does not provide access to any views of the document other than the default view. The default view is accessible through `DocumentView::defaultView`. Note that the default view is required to also implement the `SVGGlobal` interface. In the ECMAScript language binding, the global script object must also be the object that represents the default view.

A.5.1 AbstractView

This interface is a copy of the AbstractView interface from *DOM Level 2 Views* ([DOM2VIEWS], section 1.2), and must be implemented by the object that represents the the default view of the document. In the ECMAScript language binding, the global script object must implement this interface.

IDL Definition

```
interface AbstractView
{
        readonly attribute DocumentView document;
};
```

No defined constants

Attributes

document

The document that this `SVGGlobal` is associated with, as a `DocumentView`. Note that this object is also a `Document`. See `AbstractView::document` in *DOM Level 2 Views* ([DOM2VIEWS], section 1.2).

No defined methods

A.5.2 DocumentView

This interface is a copy of the DocumentView interface from *DOM Level 2 Views* ([DOM2VIEWS], section 1.2), and must be implemented by all `Document` objects.

IDL Definition

```
interface DocumentView
{
        readonly attribute AbstractView defaultView;
};
```

No defined constants

Attributes

defaultView

The default `AbstractView` for this `Document`, or `null` if none available. The value of this attribute is the `SVGGlobal` object associated with the document. See `DocumentView::defaultView` in *DOM Level 2 Views* ([DOM2VIEWS], section 1.2).

No defined methods

A.6 Module: events

A.6.1 EventTarget

The `EventTarget` interface is implemented by objects that can notify listeners about events and allows registration and removal of `EventListener` objects.

This interface is a subset of the `EventTarget` interface defined in *DOM Level 2 Events* ([DOM2EVENTS], section 1.3.1).

Please note that SVG Tiny 1.2 user agents are not required to support the capture phase, and conformant SVG Tiny 1.2 content must not make use of it. If an attempt to specify event operations on the capture phase is made an SVG Tiny user agent that does not support it must ignore them as if `addEventListener` had not been called. (See Event flow for details.)

As indicated in the *DOM Level 2 Events* definition for `EventTarget`, this interface is implemented by all `Node`s.

Refer to the *DOM Events Level 2* specification [DOM2EVENTS] or the *XML Events* [XML-EVENTS] specification introduction for an explanation of the SVG event flow and the meaning of event targets, event current target, bubble and capture.

IDL Definition

```
interface EventTarget
{
        void addEventListener(in DOMString type, in EventListener listener, in boolean useCapture);
        void removeEventListener(in DOMString type, in EventListener listener, in boolean useCapture);
};
```

No defined constants

No defined attributes

Methods

addEventListener

See `addEventListener`.

removeEventListener

See `removeEventListener`.

A.6.2 EventListener

The `EventListener` interface is implemented by script to handle an event. The interface can be implemented in ECMAScript by using a Function object (or by using an object with a `handleEvent` property), and in Java by implementing the interface directly. The `EventListener` object can then be registered as a listener using `EventTarget::addEventListener`.

This interface is identical to the `EventListener` interface defined in *DOM Level 2 Events* ([DOM2EVENTS], section 1.3.1).

IDL Definition

```
interface EventListener
{
        void handleEvent(in Event evt);
};
```

No defined constants

No defined attributes

Methods

handleEvent

See `handleEvent`.

A.6.3 Event

The `Event` interface is used to provide contextual information about an event to the handler processing the event. This interface is a subset of the `Event` interface defined in *DOM Level 2 Events* ([DOM2EVENTS, section 1.4), with one addition: the `defaultPrevented` attribute. This subset does not support the PhaseType definition group.

For a list of supported event types see the Complete list of supported events section of the Interactivity chapter.

IDL Definition

```
interface Event
{
        readonly attribute EventTarget target;
        readonly attribute EventTarget currentTarget;
        readonly attribute DOMString type;
        readonly attribute boolean cancelable;
        readonly attribute boolean defaultPrevented;
        void stopPropagation();
        void preventDefault();
};
```

No defined constants

Attributes

target

See `target`.

currentTarget

See `currentTarget`.

type

See `type`.

cancelable

See `cancelable`.

defaultPrevented

Used to indicate whether `Event.preventDefault()` has been called for this event.

Methods

stopPropagation

See `stopPropagation`.

preventDefault

See `preventDefault`.

A.6.4 MouseEvent

`Event` that provides specific contextual information associated with pointing device events.

`Event` types that are `MouseEvent`s: click, mousedown, mouseup, mouseover, mousemove, mouseout.

This interface is a subset of the `MouseEvent` interface defined in *DOM Level 2 Events* ([DOM2EVENTS, section 1.6.2).

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

See `screenX`.

screenY

See `screenY`.

clientX

See `clientX`.

clientY

See `clientY`.

button

See `button`.

No defined methods

A.6.5 MouseWheelEvent

`Event` that provides specific contextual information associated with mouse wheel events.

`Event` types that are `MouseWheelEvent`s: mousewheel.

This interface is a subset of the `MouseWheelEvent` interface defined in *DOM Level 3 Events* ([DOM3EVENTS], section 1.7.6), and inherits attributes from the `MouseEvent` interface defined in *DOM Level 2 Events* ([DOM2EVENTS], section 1.6.2).

IDL Definition

```
interface MouseWheelEvent : MouseEvent
{
        readonly attribute long wheelDelta;
};
```

No defined constants

Attributes

wheelDelta

The distance the wheel has rotated around the y-axis. A positive value shall indicate that the wheel has been rotated away from the user on vertically-aligned devices or in a left-hand manner on horizontally aligned devices, and a negative value shall indicate that the wheel has been rotated towards the user on vertically-aligned devices or in a right-hand manner on horizontally-aligned devices.

No defined methods

A.6.6 TextEvent

`Event` type that is a `TextEvent`: `textInput`.

This interface is a subset of the `TextEvent` interface defined in *DOM Level 3 Events* ([DOM3EVENTS], section 1.7.2).

IDL Definition

```
interface TextEvent : UIEvent
{
        readonly attribute DOMString data;
};
```

No defined constants

Attributes

data

`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 [UNICODE]. Characters should be normalized to Unicode normalization form *NFC*, defined in *Unicode Normalization Forms* [UAX15]. This attribute will not be `null` or contain an empty string.

No defined methods

A.6.7 KeyboardEvent

Provides specific contextual information associated with keyboard devices. Each `KeyboardEvent` references a key using an identifier.

`Event` types that are `KeyboardEvents`: `keydown`, `keyup`.

This interface is a subset of the `KeyboardEvent` interface defined in *DOM Level 3 Events* ([DOM3EVENTS], section 1.7.3).

IDL Definition

```
interface KeyboardEvent : UIEvent
{
        readonly attribute DOMString keyIdentifier;
};
```

No defined constants

Attributes

keyIdentifier

`keyIdentifier` holds the identifier of the key. The key identifiers are defined in the Key identifiers set, below. Implementations that are unable to identify a key must use the key identifier `"Unidentified"`.

No defined methods

Key identifiers set

This is a subset of the key identifiers defined in *DOM Level 3 Events*, and defines a snapshot of functionality currently implemented on mobile devices ([DOM3EVENTS], section A.2).

The list of key identifiers contained in this section is not exhaustive and input devices may have to define their own key identifiers. It is expected that DOM Level 3 Events will define an algorithm to determine which key identifier to use. Future SVG specifications will defer to DOM Level 3 Events for a definitive treatment of keyboard events and key identifiers.

`"U+0000"`, `"U+0001"`, ..., `"U+10FFFF"` are Unicode-based key identifiers [UNICODE]. A user agent may treat string literal characters in content as Unicode codepoints for the purpose of key identification.

"Accept"
The Accept (Commit, OK) key.

"Again"
The Again key.

"AllCandidates"
The All Candidates key.

"Alphanumeric"
The Alphanumeric key.

"Alt"
The Alt (Menu) key.

"AltGraph"
The Alt-Graph key.

"Apps"
The Application key.

"Attn"
The ATTN key.

"BrowserBack"
The Browser Back key.

"BrowserFavorites"
The Browser Favorites key.

"BrowserForward"
: The Browser Forward key.

"BrowserHome"
: The Browser Home key.

"BrowserRefresh"
: The Browser Refresh key.

"BrowserSearch"
: The Browser Search key.

"BrowserStop"
: The Browser Stop key.

"CapsLock"
: The Caps Lock (Capital) key.

"Clear"
: The Clear key.

"CodeInput"
: The Code Input key.

"Compose"
: The Compose key.

"Control"
: The Control (Ctrl) key.

"Crsel"
: The Crsel key.

"Convert"
: The Convert key.

"Copy"
: The Copy key.

"Cut"
: The Cut key.

"Down"
: The Down Arrow key.

"DownLeft"
: The diagonal Down-Left Arrow key.

"DownRight"
: The diagonal Down-Right Arrow key.

"End"
: The End key.

"Enter"
: The Enter key. ***Note:*** *This key identifier is also used for the Return (Macintosh numpad) key.*

"EraseEof"
: The Erase EOF key.

"Execute"
: The Execute key.

"Exsel"
: The Exsel key.

"F1"
: The F1 key.

"F2"
: The F2 key.

"F3"
: The F3 key.

"F4"
: The F4 key.

"F5"
: The F5 key.

"F6"
: The F6 key.

"F7"
: The F7 key.

"F8"
: The F8 key.

"F9"
: The F9 key.

"F10"
: The F10 key.

"F11"
: The F11 key.

"F12"
: The F12 key.

"F13"
: The F13 key.

"F14"
: The F14 key.

"F15"
: The F15 key.

"F16"
: The F16 key.

"F17"
: The F17 key.

"F18"
: The F18 key.

"F19"
: The F19 key.

"F20"
: The F20 key.

"F21"
: The F21 key.

"F22"
: The F22 key.

"F23"
: The F23 key.

"F24"
: The F24 key.

"FinalMode"
: The Final Mode (Final) key used on some asian keyboards.

"Find"
: The Find key.

"FullWidth"
: The Full-Width Characters key.

"HalfWidth"
: The Half-Width Characters key.

"HangulMode"
: The Hangul (Korean characters) Mode key.

"HanjaMode"
: The Hanja (Korean characters) Mode key.

"Help"
: The Help key.

"Hiragana"
: The Hiragana (Japanese Kana characters) key.

"Home"
The Home key.
"Insert"
The Insert (Ins) key.
"JapaneseHiragana"
The Japanese-Hiragana key.
"JapaneseKatakana"
The Japanese-Katakana key.
"JapaneseRomaji"
The Japanese-Romaji key.
"JunjaMode"
The Junja Mode key.
"KanaMode"
The Kana Mode (Kana Lock) key.
"KanjiMode"
The Kanji (Japanese name for ideographic characters of Chinese origin) Mode key.
"Katakana"
The Katakana (Japanese Kana characters) key.
"LaunchApplication1"
The Start Application One key.
"LaunchApplication2"
The Start Application Two key.
"LaunchMail"
The Start Mail key.
"Left"
The Left Arrow key.
"Menu"
The Menu key.
"Meta"
The Meta key.
"MediaNextTrack"
The Media Next Track key.
"MediaPlayPause"
The Media Play Pause key.
"MediaPreviousTrack"
The Media Previous Track key.
"MediaStop"
The Media Stop key.
"ModeChange"
The Mode Change key.
"Nonconvert"
The Nonconvert (Don't Convert) key.
"NumLock"
The Number Lock key.
"PageDown"
The Page Down (Next) key.
"PageUp"
The Page Up key.
"Paste"
The Paste key.
"Pause"
The Pause key.
"Play"
The Play key.
"PreviousCandidate"
The Previous Candidate function key.
"PrintScreen"
The Print Screen (PrintScrn, SnapShot) key.
"Process"
The Process key.
"Props"
The Props key.
"Right"
The Right Arrow key.
"RomanCharacters"
The Roman Characters function key.
"Scroll"
The Scroll Lock key.
"Select"
The Select key.
"SelectMedia"
The Select Media key.
"Shift"
The Shift key.
"Soft1"
The Soft1 key.
"Soft2"
The Soft2 key.
"Soft3"
The Soft3 key.
"Soft4"
The Soft4 key.
"Stop"
The Stop key.
"Up"
The Up Arrow key.
"UpLeft"
The diagonal Up-Left Arrow key.
"UpRight"
The diagonal Up-Right Arrow key.
"Undo"
The Undo key.
"VolumeDown"
The Volume Down key.
"VolumeMute"
The Volume Mute key.
"VolumeUp"
The Volume Up key.
"Win"
The Windows Logo key.
"Zoom"
The Zoom key.
"U+0008"
The Backspace (Back) key.
"U+0009"
The Horizontal Tabulation (Tab) key.
"U+0018"
The Cancel key.
"U+001B"
The Escape (Esc) key.

"U+0020"
The Space (Spacebar) key.
"U+0021"
The Exclamation Mark (Factorial, Bang) key (!).
"U+0022"
The Quotation Mark (Quote Double) key (").
"U+0023"
The Number Sign (Pound Sign, Hash, Crosshatch, Octothorpe) key (#).
"U+0024"
The Dollar Sign (milreis, escudo) key ($).
"U+0026"
The Ampersand key (&).
"U+0027"
The Apostrophe (Apostrophe-Quote, APL Quote) key (').
"U+0028"
The Left Parenthesis (Opening Parenthesis) key (().
"U+0029"
The Right Parenthesis (Closing Parenthesis) key ()).
"U+002A"
The Asterisk (Star) key (*).
"U+002B"
The Plus Sign (Plus) key (+).
"U+0025"
The Percent Sign (Percent) key (+).
"U+002C"
The Comma (decimal separator) sign key (,).
"U+002D"
The Hyphen-minus (hyphen or minus sign) key (-).
"U+002E"
The Full Stop (period, dot, decimal point) key (.).
"U+002F"
The Solidus (slash, virgule, shilling) key (/).
"U+0030"
The Digit Zero key (0).
"U+0031"
The Digit One key (1).
"U+0032"
The Digit Two key (2).
"U+0033"
The Digit Three key (3).
"U+0034"
The Digit Four key (4).
"U+0035"
The Digit Five key (5).
"U+0036"
The Digit Six key (6).
"U+0037"
The Digit Seven key (7).
"U+0038"
The Digit Eight key (8).
"U+0039"
The Digit Nine key (9).
"U+003A"
The Colon key (:).
"U+003B"
The Semicolon key (;).
"U+003C"
The Less-Than Sign key (<).
"U+003D"
The Equals Sign key (=).
"U+003E"
The Greater-Than Sign key (>).
"U+003F"
The Question Mark key (?).
"U+0040"
The Commercial At (@) key.
"U+0041"
The Latin Capital Letter A key (A).
"U+0042"
The Latin Capital Letter B key (B).
"U+0043"
The Latin Capital Letter C key (C).
"U+0044"
The Latin Capital Letter D key (D).
"U+0045"
The Latin Capital Letter E key (E).
"U+0046"
The Latin Capital Letter F key (F).
"U+0047"
The Latin Capital Letter G key (G).
"U+0048"
The Latin Capital Letter H key (H).
"U+0049"
The Latin Capital Letter I key (I).
"U+004A"
The Latin Capital Letter J key (J).
"U+004B"
The Latin Capital Letter K key (K).
"U+004C"
The Latin Capital Letter L key (L).
"U+004D"
The Latin Capital Letter M key (M).
"U+004E"
The Latin Capital Letter N key (N).
"U+004F"
The Latin Capital Letter O key (O).
"U+0050"
The Latin Capital Letter P key (P).
"U+0051"
The Latin Capital Letter Q key (Q).
"U+0052"
The Latin Capital Letter R key (R).
"U+0053"
The Latin Capital Letter S key (S).
"U+0054"
The Latin Capital Letter T key (T).
"U+0055"
The Latin Capital Letter U key (U).
"U+0056"
The Latin Capital Letter V key (V).

"U+0057"
: The Latin Capital Letter W key (W).

"U+0058"
: The Latin Capital Letter X key (X).

"U+0059"
: The Latin Capital Letter Y key (Y).

"U+005A"
: The Latin Capital Letter Z key (Z).

"U+005B"
: The Left Square Bracket (Opening Square Bracket) key ([).

"U+005C"
: The Reverse Solidus (Backslash) key (\).

"U+005D"
: The Right Square Bracket (Closing Square Bracket) key (]).

"U+005E"
: The Circumflex Accent key (^).

"U+005F"
: The Low Sign (Spacing Underscore, Underscore) key (_).

"U+0060"
: The Grave Accent (Back Quote) key (`).

"U+007B"
: The Left Curly Bracket (Opening Curly Bracket, Opening Brace, Brace Left) key ({).

"U+007C"
: The Vertical Line (Vertical Bar, Pipe) key (|).

"U+007D"
: The Right Curly Bracket (Closing Curly Bracket, Closing Brace, Brace Right) key (}).

"U+007F"
: The Delete (Del) Key.

"U+00A1"
: The Inverted Exclamation Mark key (¡).

"U+0300"
: The Combining Grave Accent (Greek Varia, Dead Grave) key.

"U+0301"
: The Combining Acute Accent (Stress Mark, Greek Oxia, Tonos, Dead Eacute) key.

"U+0302"
: The Combining Circumflex Accent (Hat, Dead Circumflex) key.

"U+0303"
: The Combining Tilde (Dead Tilde) key.

"U+0304"
: The Combining Macron (Long, Dead Macron) key.

"U+0306"
: The Combining Breve (Short, Dead Breve) key.

"U+0307"
: The Combining Dot Above (Derivative, Dead Above Dot) key.

"U+0308"
: The Combining Diaeresis (Double Dot Above, Umlaut, Greek Dialytika, Double Derivative, Dead Diaeresis) key.

"U+030A"
: The Combining Ring Above (Dead Above Ring) key.

"U+030B"
: The Combining Double Acute Accent (Dead Doubleacute) key.

"U+030C"
: The Combining Caron (Hacek, V Above, Dead Caron) key.

"U+0327"
: The Combining Cedilla (Dead Cedilla) key.

"U+0328"
: The Combining Ogonek (Nasal Hook, Dead Ogonek) key.

"U+0345"
: The Combining Greek Ypogegrammeni (Greek Non-Spacing Iota Below, Iota Subscript, Dead Iota) key.

"U+20AC"
: The Euro Currency Sign key (€).

"U+3099"
: The Combining Katakana-Hiragana Voiced Sound Mark (Dead Voiced Sound) key.

"U+309A"
: The Combining Katakana-Hiragana Semi-Voiced Sound Mark (Dead Semivoiced Sound) key.

A.6.8 UIEvent

The `UIEvent` interface provides specific contextual information associated with user interface events.

`Event` types that are `UIEvents`: `DOMFocusIn`, `DOMFocusOut`, `DOMActivate`, `MouseEvent`, `TextEvent`, `KeyboardEvent`,

This interface is a subset of the `UIEvent` interface defined in *DOM Level 2 Events* ([DOM2EVENTS, section 1.6.1).

IDL Definition

```
interface UIEvent : Event
{
        readonly attribute long detail;
};
```

No defined constants

Attributes

detail

See `detail`.

No defined methods

A.6.9 ProgressEvent

The progress events defined here are intended to be a subset of those defined in *Progress Events 1.0* [PROGRESSEVENTS].

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 (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 `loadstart`.

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 `loadend`.

The user agent may dispatch `ProgressEvent`s between the `loadstart` event and the `loadend` events. Such events are of type `progress`.

`Event` types that are `ProgressEvents`: `progress`, `loadstart`, `loadend`.

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 `loadstart` or `loadend` event.

total

Specifies the expected total number of bytes expected in a load operation. For a `progress` event, it should specify the total number of bytes expected.

No defined methods

Example: ProgressEvent

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny" width="300" height="430">

  <script><![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');
      var loadingAnimation = document.getElementById('loadingAnimation');
      progressBar.setFloatTrait("width", 0);
      loadingAnimation.beginElement();
    }

    function imageLoadProgress(evt) {
      if (evt.lengthComputable) {
        var progressBar = document.getElementById('progressBar');
        progressBar.setFloatTrait("width", 100 * (evt.loaded / evt.total));
        progressBar.setTrait("display", "inline");
      }
    }

    function imageLoadComplete(evt) {
      var progressBar = document.getElementById('progressBar');
      var loadingAnimation = document.getElementById('loadingAnimation');
      progressBar.setTrait("display", "none");
      loadingAnimation.endElement();
    }
  ]]></script>

  <image xml:id="myImage" xlink:href="imageA.png" width="300" height="400">
    <handler ev:event="loadstart">
      imageLoadStart(evt);
    </handler>

    <handler ev:event="progress">
      imageLoadProgress(evt);
    </handler>

    <handler ev:event="loadend">
      imageLoadComplete(evt);
    </handler>
  </image>

  <rect rx="4" x="50" y="400" width="200" height="30" cursor="pointer">
    <handler ev:event="click">
      showImage('imageB.png');
    </handler>
  </rect>
  <text x="150" y="420" font-size="15" fill="white" text-anchor="middle"
        text-decoration="underline" pointer-events="none">
    Load other image
  </text>

  <g display="none">
    <rect x="100" y="300" height="10" width="100" fill="black"/>
    <rect xml:id="progressBar" x="100" y="300" width="50" height="10" fill="lime"/>
  </g>
  <text x="150" y="330" font-size="15" text-anchor="middle" display="none">
    Loading...
    <animate xml:id="loadingAnimation" attributeName="display"
             begin="indefinite" dur="2s" repeatDur="indefinite"
             calcMode="discrete" values="inline; none"/>
  </text>
</svg>
```

A.7 Module: smil

Contains a single subsetted interface from the SMIL APIs.

A.7.1 ElementTimeControl

This interface defines common methods for elements which define animation behaviors compatible with SMIL (timed elements and the **'svg'** element).

This interface is a subset of the ElementTimeControl interface defined in *SMIL Animation* [SMILANIM].

Note: See the `SVGTimedElement` interface for pause functionality.

IDL Definition

```
interface ElementTimeControl
{
        void beginElementAt(in float offset);
        void beginElement();
        void endElementAt(in float offset);
        void endElement();
};
```

No defined constants
No defined attributes
Methods

beginElementAt

Creates a begin instance time for the current time plus or minus the specified 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 specified 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

A.7.2 TimeEvent

`TimeEvent` is an interface used to provide contextual information for events fired by animations in the document. It is a subset of the `TimeEvent` interface defined in *SMIL Animation* ([SMILANIM], section 6.2).

`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.8 Module: svg

A.8.1 SVGException

An exception thrown for SVG-specific errors.

This interface is identical to `SVGException` interface defined in SVG 1.1 ([SVG11], section B.3).

IDL Definition

```
exception SVGException
{
        unsigned short code;
};

// ExceptionCode
const unsigned short SVG_WRONG_TYPE_ERR         = 0;
const unsigned short SVG_INVALID_VALUE_ERR      = 1;
const unsigned short SVG_MATRIX_NOT_INVERTABLE  = 2;
```

Constants

SVG_WRONG_TYPE_ERR

See definition.

SVG_INVALID_VALUE_ERR

See definition.

SVG_MATRIX_NOT_INVERTABLE

See definition.

No defined attributes

No defined methods

A.8.2 SVGDocument

IDL Definition

```
interface SVGDocument : Document, EventTarget
{
};
```

No defined constants

No defined attributes

No defined methods

A.8.3 SVGUseElement

This interface represents the **'use'** element. In SVG 1.2 Tiny this interface has no additional methods to those it inherits; it is included for architectural consistency with other profiles of SVG.

IDL Definition

```
interface SVGUseElement : SVGLocatableElement
{
};
```

No defined constants
No defined attributes
No defined methods

A.8.4 SVGElementInstance

For each **'use'** element, the uDOM represents the referenced content with a shadow tree of `SVGElementInstance` objects.
This interface is a subset of the `SVGElementInstance` interface defined in SVG 1.1 ([SVG11], section 5.17).

IDL Definition

```
interface SVGElementInstance : EventTarget
{
        readonly attribute SVGElement correspondingElement;
        readonly attribute SVGUseElement correspondingUseElement;
};
```

No defined constants
Attributes

correspondingElement
See `correspondingElement`.

correspondingUseElement
See `correspondingUseElement`.

No defined methods

In the example below, 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 throw an exception when the original **'rect'** is clicked. This is because the `currentTarget` attribute would return an `SVGElement` that doesn't have the `correspondingUseElement` attribute.

Example: Usage of the SVGElementInstance interface (ECMAScript)

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:ev="http://www.w3.org/2001/xml-events"
     version="1.2" baseProfile="tiny" width="640" height="480" viewBox="0 0 640 480">

  <defs>
    <rect xml: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 xml:id="r2" x="250" y="50" width="90" height="65"/>

  <ev:listener observer="r1" event="ev:click" handler="#handler"/>

  <handler xml:id="handler" type="application/ecmascript">changeColor(evt);</handler>

  <script type="application/ecmascript">
    var target = document.getElementById("r2");
    function changeColor(evt) {
      var useElement = evt.currentTarget.correspondingUseElement;
      target.setRGBColorTrait("fill", useElement.getRGBColorTrait("fill"));
    }
  </script>
</svg>
```

A.8.5 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). Their values can potentially be modified through user agent specific UI, if "magnification" is enabled (i.e., **'zoomAndPan'** attribute is set to **magnify**). User agent transformation can be obtained by multiplying the matrix

```
[currentScale      0       currentTranslate.x]        [cos(currentRotate) -sin(currentRotate 0]
[     0      currentScale  currentTranslate.y]  by    [sin(currentRotate) cos(currentRotate) 0]
[     0            0               1         ]        [         0                  0         1]
```

That is, 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, SVGTimedElement
{
        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;
        float getCurrentTime();
        void setCurrentTime(in float seconds);
        SVGMatrix createSVGMatrixComponents(in float a, in float b, in float c, in float d, in float e,
                                            in float f);
        SVGRect createSVGRect();
        SVGPoint createSVGPoint();
        SVGPath createSVGPath();
        SVGRGBColor createSVGRGBColor(in float red, in float green, in float blue)
                raises(SVGException);
        void moveFocus(in unsigned short motionType)
                raises(DOMException);
        void setFocus(in EventTarget theObject)
                raises(DOMException);
        EventTarget 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
: The current user agent scale (zoom) coefficient. The initial value for `currentScale` is 1.

currentRotate
: The current user agent rotation angle in degrees. The initial value for `currentRotate` is 0.

currentTranslate
: The 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 for `currentTranslate` is an `SVGPoint` object with the value (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 CSS 2 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 CSS 2 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 exactly match SVG and CSS's "px" coordinates.

The object itself and its contents are both readonly. A `DOMException` with error code NO_MODIFICATION_ALLOWED_ERR is raised if an attempt is made to modify it. The returned `SVGRect` object is "live", i.e. its `x`, `y`, `width` and `height` attributes are automatically updated if the viewport size or position changes.

Methods

getCurrentTime
: Returns the document time in seconds.

If `getCurrentTime` is called before the document timeline has begun (for example, by script running in a **'script'** element before the rootmost 'svg' element's `load` event is dispatched, when **'playbackOrder'** is set to **'onLoad'**), then 0 is returned.

Return value

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

No parameters

No exceptions

setCurrentTime

Sets the document 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. If seconds is negative, then the document will seek to time 0s.

If `setCurrentTime` is called before the document timeline has begun (for example, by script running in a **'script'** element before the rootmost 'svg' element's `load` event is dispatched, when **'playbackOrder'** is set to **'onLoad'**), then the value of seconds in the most recent invocation of the method gives the time that the document time will be seeked to once the document timeline has begun.

Parameters

in float seconds	The document time to seek to, in seconds.

No return value

No exceptions

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` and `height` of this new `SVGRect` are zero.

No parameters

Return value

`SVGRect`	The created `SVGRect`.

No exceptions

createSVGPoint

Creates a new `SVGPoint` object. The initial values for `x` and `y` of this new `SVGPoint` are zero.

No parameters

Return value

`SVGPoint`	The created `SVGPoint`.

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. The parameters are floats, one per color component. 0.0 represents zero intensity and 255.0 represents full intensity of a given color component. Colors originally in the rgb(%,%,%) syntax may have fractional components. Out of gamut colors may have component values less than 0.0 or greater than 255.0.

Parameters

in float red	The red component of the `SVGRGBColor`.
in float green	The green component of the `SVGRGBColor`.
in float blue	The blue component of the `SVGRGBColor`.

Return value

`SVGRGBColor`	The created `SVGRGBColor`.

No exceptions

moveFocus

Moves the current focus to a different object based on the value of the parameter. 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 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 for a description of how navigation is managed. The behavior for this method must be the same as if an equivalent move was done by the end user (for example by using a joystick or pressing the Tab key) and not by scripting.

Whenever the method fails (that is, when 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_ERR: 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 navigation attribute 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 navigation attribute 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 newly focused object can be obtained using the `EventTarget` interface of the generated `DOMFocusIn` event.

Whenever the method fails (that is, when a `DOMException` is raised), focus must stay on the currently focused object and no `DOMFocusOut` or `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 `EventTarget` theObject	The object which should receive focus.

No return value

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised if the in parameter is not a `Node` or `SVGElementInstance`, or if the requested element is not focusable (i.e. its **'focusable'** attribute indicates that the element is not focusable).

getCurrentFocusedObject

Returns a reference to the object which has the focus. This returns an `EventTarget`.

No parameters

Return value

`EventTarget`	object	The currently focused object.

No exceptions

A.8.6 SVGRGBColor

This interface represents a color value made up of red, green, and blue components. It can be used to read and write traits that store color values (using `getRGBColorTrait`) such as **'fill'**, **'stroke'**, and **'color'**.

IDL Definition

```
interface SVGRGBColor
{
        attribute unsigned long red;
        attribute unsigned long green;
        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.8.7 SVGRect

This interface represents an `SVGRect` datatype, consisting of a minimum *x*, minimum *y*, width and height values.
This interface is identical to `SVGRect` interface defined in SVG 1.1 ([SVG11], section 4.3).

IDL Definition

```
interface SVGRect
{
        attribute float x;
        attribute float y;
        attribute float width;
        attribute float height;
};
```

No defined constants

Attributes

x
: See `x`.

y
: See `y`.

width
: See `width`.

height
: See `height`.

No defined methods

A.8.8 SVGPoint

Represents an `SVGPoint` datatype, identified by its *x* and *y* components.
This interface is identical to `SVGPoint` interface defined in SVG 1.1 ([SVG11], section 4.3).

IDL Definition

```
interface SVGPoint
{
        attribute float x;
        attribute float y;
        SVGPoint matrixTransform(in SVGMatrix matrix);
};
```

No defined constants

Attributes

x
: See `x`.

y
: See `y`.

Methods

matrixTransform
: See `matrixTransform`.

A.8.9 SVGPath

This interface represents an `SVGPath` datatype used to define path geometry.
Path data created or modified using this interface must be normalized as per the rules given in Path Normalization. However, path data that is just queried need not be normalized.

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
Represents a "move to" command. The numeric value is the Unicode codepoint of the letter "M".

LINE_TO
Represents a "line to" command. The numeric value is the Unicode codepoint of the letter "L".

CURVE_TO
Represents a "cubic Bézier curve to" command. The numeric value is the Unicode codepoint of the letter "C".

QUAD_TO
Represents a "quadrative Bézier curve to" command. The numeric value is the Unicode codepoint of the letter "Q".

CLOSE
Represents a "close" command. The numeric value is the Unicode codepoint 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 the segment index is out of bounds.

getSegmentParam
Returns segment parameter by zero-based command index and zero-based parameter index.

Parameters

in unsigned long cmdIndex	The command index for the segment command.

in unsigned long paramIndex	The parameter index to retrieve.

Return value

float	The segment parameter.

Exceptions

`DOMException`	INDEX_SIZE_ERR: Raised if the segment index is out of bounds, or the parameter index is out of bounds for the specified 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

Appends a 'z' (close path) segment to the path.

A.8.10 SVGMatrix

This interface is a matrix, as is used to represent 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:

```
[ x' ]    [  a  c  e  ]   [ x ]    [ a.x + c.y + e ]
[ y' ] =  [  b  d  f  ]   [ y ] =  [ b.x + d.y + f ]
[ 1  ]    [  0  0  1  ]   [ 1 ]    [       1       ]
```

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 (i.e., outside the range [0, 5]).

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 the 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 `mMultiply(T)`, where `T` is an `SVGMatrix` object represented by the following matrix:

```
[   1    0    x  ]
[   0    1    y  ]
[   0    0    1  ]
```

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 `mMultiply(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 `mMultiply(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.8.11 SVGLocatable

Interface for getting information about the location of elements.

IDL Definition

```
interface SVGLocatable
{
        SVGRect   getBBox();
        SVGMatrix getScreenCTM();
        SVGRect   getScreenBBox();
};
```

No defined constants
No defined attributes
Methods

getBBox

Returns the bounding box of the element.

No parameters

Return value

`SVGRect`	The bounding box. 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. The `clientX` and `clientY` coordinates of a `MouseEvent` 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. Also note that `getScreenCTM` reflects a snapshot of the current animated state, i.e. if one or several transforms that affect the element that `getScreenCTM` is called upon are animated then the returned transformation matrix reflects the current state of each such animated transform when calculating the returned matrix.

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 bounding box of the element in screen coordinate space. 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.

No parameters

Return value

`SVGRect`	The 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` objects 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 xml:id="group1" transform="translate(10, 20)" fill="red">
    <rect xml:id="rect1" transform="scale(2)" x="10" y="10" width="50" height="50"/>
    <rect xml:id="rect2" x="10" y="10" width="100" height="100"/>
    <g xml:id="group2" transform="translate(10, 20)">
        <rect xml:id="rect3" x="0" y="10" width="150" height="50"/>
        <circle xml: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 xml:id="group1" transform="translate(10, 20)" fill="red">
    <rect xml:id="rect2" x="10" y="10" width="400" height="0"/>
    <g xml:id="group2" transform="translate(10, 20)">
        <rect xml: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 xml:id="mySVG" version="1.2" baseProfile="tiny" width="10" height="20">
    <g xml:id="group1" transform="translate(10, 20)" fill="red">
        <rect xml:id="rect1" x="10" y="10" width="100" height="100"/>
        <ellipse xml: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 rootmost 'svg' element, the bounding boxes, based on the geometry, are still defined. Here, even though the rootmost 'svg' element has a zero width, the bounding boxes for the root itself and its children is precisely defined.

```
<svg xml:id="mySVG" version="1.2" baseProfile="tiny" width="0" height="50">
    <g xml:id="group1" transform="translate(10, 20)" fill="red" >
        <rect xml:id="rect1" x="10" y="10" width="50" height="50"/>
        <g xml:id="group2" transform="translate(10, 20)">
            <rect xml:id="rect2" x="0" y="10" width="150" height="0"/>
            <circle xml:id="circle1" cx="20" cy="20" r="500"/>
        </g>
    </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 xml:id="myRect" x="0" y="0" width="60" height="40"/>
    </defs>
    <use xml: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 xml: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 **'visibility'** set to **hidden** still contribute to their parent's bounding box computation.

```
<g xml:id="g1">
    <g xml:id="g1.1.display.none" display="none">
        <rect xml:id="rect1" x="10" y="10" width="40" height="40"/>
    </g>
    <rect xml: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 xml:id="g1">
    <line xml: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 xml: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 xml: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}
```

Example #11: Impact of elements which are not in the rendering tree
This example illustrates that elements which are not in the rendering tree have no impact on the computation of bounding boxes.

```
        <g xml:id="g1">
          <linearGradient xml:id="MyGradient"/>
            <stop offset="0.05" stop-color="#F60"/>
            <stop offset="0.95" stop-color="#FF6"/>
          </linearGradient>
        </g>
```

Result:

```
[g1] : {null}
```

A.8.12 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.8.13 TraitAccess

Trait manipulation interface. This interface is used to 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 unsupported trait must throw an exception. Also, unlike attributes, traits are typed and their values are normalized; for instance path data specified on a **'path'** element is parsed and all path commands are converted to their absolute variants, and it is not possible to determine from the value of the trait if a path command was absolute or relative. When getting and setting trait values, an accessor of the correct type must be used or an exception will be thrown.

For a trait corresponding to a property, the computed value is used: if the value of a given property is not specified on that element, then for inherited properties the value on the parent is used. For non-inherited values, or on the root element, the initial value of the property is used.

For a trait corresponding to a non-property attribute, if the attribute is inherited (such as **'xml:lang'**) the value of the parent is used. If the attribute is not inherited, or on the root element, the lacuna value for the attribute is used, if known. If not known (for example, for an unknown attribute, or a known attribute with no specified default), `null` is returned.

Note that when using the `TraitAccess` interface for getting traits on elements outside of the tree, for example on elements just created or removed, what values are returned is user agent dependent.

The trait getter methods (`getTrait`, `getTraitNS`, `getFloatTrait`, etc.) return base values (i.e., before animation is applied), and this is true for both static and animated content. Note however that if the attribute is inherited from an animated parent value, it will inherit the animated value. The trait presentation value getter methods (`getPresentationTrait`, `getPresentationTraitNS`, `getFloatPresentationTrait`, etc.) return either the current animated value if the given trait is currently being animated 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 for details.

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 corresponding to properties, 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 the `setTrait`, `setFloatTrait` and `setPathTrait` methods) consider a value invalid and throw a `DOMException` with the INVALID_ACCESS_ERR code. The first situation is when the trait value is invalid with regards to its definition. (For example, trying to set the **'stroke-linejoin'** trait to **'foo'** would result in this exception being thrown). The trait methods will consider the value to be invalid if it is an unsupported value. However, if the trait value being set is an IRI reference, such as when setting a <FuncIRI> value on the **'fill'** property or when setting the **'xlink:href'** attribute on an **'image'** element, an SVG user agent must not consider that trait value invalid if it is syntactically correct but is otherwise an invalid IRI reference. Thus, the `DOMException` with code INVALID_ACCESS_ERR must not be thrown in this case. This obviates the need for an SVG user agent to fetch the IRI upon setting the trait solely to determine whether it is an invalid IRI reference.

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.

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);
        sequence<float> getFloatListTrait(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);
        sequence<float> getFloatListPresentationTrait(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 setFloatListTrait(in DOMString name, in sequence<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 method 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`. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 is a non-numeric `float` (for example, when calling `getFloatTrait("width")` on the rootmost 'svg' element whose width attribute uses a percentage).

getFloatListTrait

Get the trait value as a `sequence<float>`. Parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters

in `DOMString` name	The name of the trait to retrieve.

Return value

`sequence<float>`	The trait value as a `sequence<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 `sequence<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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. If the actual trait value is not an `SVGRect`, e.g. the **'none'** value on the **'viewBox'** attribute, this method will return `null`. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 **'solidColor'**), this method must return `null`. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 method 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`. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 is a non-numeric `float` (for example, when calling `getFloatTrait("width")` on the rootmost 'svg' element whose width attribute uses a percentage).

getFloatListPresentationTrait

Get the trait presentation value as a `sequence<float>`. Parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters

in `DOMString` name	The name of the trait to retrieve.

Return value

`sequence<float>`	The trait presentation value as a `sequence<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 `sequence<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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. If the actual trait value is not an SVGRect, e.g. the **'none'** value on the **'viewBox'** attribute, this method will return `null`. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 value is not an `SVGRGBColor`, i.e. **'none'** or a link to a paint server (e.g. to a gradient or a **'solidColor'**), this method must return `null`. Parameter name must be a non-qualified trait name, i.e. without prefix.

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.
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.
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.

setFloatTrait

Set the trait value as a float. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 numeric `float` (e.g. `NaN`).
`DOMException`	INVALID_ACCESS_ERR: Raised if the input value is an invalid value for the given trait or `null` is specified.

setFloatListTrait

Set the trait value as a `sequence<float>`. Parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters

in `DOMString` name	The name of the trait to be set.
in `sequence<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 `sequence<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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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. Parameter name must be a non-qualified trait name, i.e. without prefix.

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 1.2 attributes and properties. The table below shows the attributes and properties that SVG Tiny 1.2 uDOM implementations must support trait access to. Each attribute row lists the allowed getters and setters. The "Lacuna Value" column specifies the lacuna value that must be used for each attribute or property. If a "Lacuna Value" column entry is empty, there is no lacuna value. Unless explicitly stated in the "Comments" column, a supported attribute is accessible on all elements it can belong to. See the attribute table 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.

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 user agent.

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 the `getTraitNS` method, it must also support the `getPresentationTraitNS` method.
- 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 trait presentation getter 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. They *cannot* be used for accessing font descriptors.

Trait Target	Trait Getter [Return Value]	Trait Setter [Argument Value]	Lacuna Value	Comments
Element text content	`getTrait("#text")` [The element text content]	`setTrait("#text", ...)` [New element text content]		See Text Conter Access.
accumulate, attribute	`getTrait("accumulate")` [none | sum]	`setTrait("accumulate", ...)` [none | sum]	"none"	
additive, attribute	`getTrait("additive")` [replace | sum]	`setTrait("additive", ...)` [replace | sum]	"replace"	
attributeName, attribute	`getTrait("attributeName")`	`setTrait("attributeName", ...)`		
audio-level, property	`getFloatTrait("audio-level")` [0 ≤ *value* ≤ 1]	`setFloatTrait("audio-level", ...)` [0 ≤ *value* ≤ 1] `setTrait("audio-level", ...)` [inherit]	1.0	
baseProfile, attribute	`getTrait("baseProfile")`	`setTrait("baseProfile", ...)`	"none"	
begin, attribute	N/A	`setTrait("begin", ...)`		
calcMode, attribute	`getTrait("calcMode")` [discrete | linear | paced | spline]	`setTrait("calcMode", ...)` [discrete | linear | paced | spline]	"paced" when accessed on an **'animateMotion'** element, "linear" otherwise	
color, property	`getRGBColorTrait("color")` [null, `SVGRGBColor`]	`setRGBColorTrait("color, ...")` [`SVGRGBColor`] `setTrait("color", ...)` [inherit | <color>]	rgb(0,0,0)	
cx, attribute	`getFloatTrait("cx")`	`setFloatTrait("cx", ...)`	0.5 when accessed on a **'radialGradient'** element, 0.0 otherwise	

cy, attribute	`getFloatTrait("cy")`	`setFloatTrait("cy", ...)`	0.5 when accessed on a **'radialGradient'** element, 0.0 otherwise	
d, attribute	`getPathTrait("d")` [`SVGPath`]	`setPathTrait("d", ...)` [`SVGPath`]	An `SVGPath` object with no path segments	See Accessing rules for path attributes.
display, property	`getTrait("display")` [inline | none]	`setTrait("display", ...)` [inline | none | inherit]	"inline"	See Accessing rules for **'display** property.
dur, attribute	N/A	`setTrait("dur", ...)`		
editable, attribute	`getTrait("editable")` [simple | none]	`setTrait("editable, ...)` [simple | none]	"none"	
end, attribute	N/A	`setTrait("end", ...)`		

fill, property	`getRGBColorTrait("fill")`
	[null, `SVGRGBColor`]
	
	`getTrait("fill")`
	[none | <FuncIRI> | <color> | <system paint>]
		`setRGBColorTrait("fill", ...)`
		[`SVGRGBColor`]
		
		`setTrait("fill", ...)`
		[none | currentColor | <FunCIRI> | <color> | <system paint> | inherit]
			rgb(0,0,0)	There are no tra accessors for th animation ele-ment **'fill'** attribute.

fill-opacity, property	`getFloatTrait("fill-opacity")`
	[0 ≤ *value* ≤ 1]
		`setFloatTrait("fill-opacity", ...)`
		[0 ≤ *value* ≤ 1]
		
		`setTrait("fill-opacity", ...)`
		[inherit]
			1.0	

fill-rule, property	`getTrait("fill-rule")` [nonzero | evenodd]	`setTrait("fill-rule", ...)` [nonzero | evenodd | inherit]	"nonzero"	
focusable, attribute	`getTrait("focusable")` [true | false]	`setTrait("focusable", ...)` [true | false | auto]	"auto"	
focusHighlight, attribute	`getTrait("focusHighlight")` [auto | none]	`setTrait("focusHighlight", ...)` [auto | none]	"auto"	
font-family, property	`getTrait("font-family")` [<font-family-value>]	`setTrait("font-family", ...)` [<font-family-value> | inherit]	User agent specific	See Accessing rules for font properties.

font-size, property	`getFloatTrait("font-size")`
	[*value* ≥ 0]
		`setFloatTrait("font-size", ...)`
		[*value* ≥ 0]
		
		`setTrait("font-size", ...)`
		[xx-small | x-small | small | medium | large | x-large | xx-large | larger | smaller | inherit]
			User agent specific	

font-style, property	`getTrait("font-style")` [normal | italic | oblique]	`setTrait("font-style", ...)` [normal | italic | oblique | inherit]	"normal"	See Accessing rules for font properties.

font-weight, property	`getTrait("font-weight")` [100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900]	`setTrait("font-weight", ...)` [normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit]	"normal"	See Accessing rules for font properties.
from, attribute	N/A	`setTrait("from", ...)`		
gradientUnits, attribute	`getTrait("gradientUnits")` [userSpaceOnUse | objectBoundingBox]	`setTrait("gradientUnits", ...)` [userSpaceOnUse | objectBoundingBox]	"objectBoundingBox"	
height, attribute	`getFloatTrait("height")` [*value* ≥ 0] `getTrait("height")` ["auto"]	`setFloatTrait("height", ...)` [*value* ≥ 0] `setTrait("height", ...)` ["auto"]	"auto" when accessed on a **'textArea'** element, 0.0 otherwise	See Accessing rules for **'width** and **'height'** attributes.
id, attribute	`getTrait("id")`	`setTrait("id", ...)`		
keyPoints, attribute	N/A	`setTrait("keyPoints", ...)`		
keySplines, attribute	N/A	`setTrait("keySplines", ...)`		
keyTimes, attribute	N/A	`setTrait("keyTimes", ...)`		
max, attribute	N/A	`setTrait("max", ...)`		
min, attribute	N/A	`setTrait("min", ...)`		
nav-right, attribute	`getTrait("nav-right")` [auto | self | <FuncIRI>]	`setTrait("nav-right", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-next, attribute	`getTrait("nav-next")` [auto | self | <FuncIRI>]	`setTrait("nav-next", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-up, attribute	`getTrait("nav-up")` [auto | self | <FuncIRI>]	`setTrait("nav-up", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-up-right, attribute	`getTrait("nav-up-right")` [auto | self | <FuncIRI>]	`setTrait("nav-up-right", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-up-left, attribute	`getTrait("nav-up-left")` [auto | self | <FuncIRI>]	`setTrait("nav-up-left", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-prev, attribute	`getTrait("nav-prev")` [auto | self | <FuncIRI>]	`setTrait("nav-prev", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-down, attribute	`getTrait("nav-down")` [auto | self | <FuncIRI>]	`setTrait("nav-down", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-down-right, attribute	`getTrait("nav-down-right")` [auto | self | <FuncIRI>]	`setTrait("nav-down-right", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-down-left, attribute	`getTrait("nav-down-left")` [auto | self | <FuncIRI>]	`setTrait("nav-down-left", ...)` [auto | self | <FuncIRI>]	"auto"	
nav-left, attribute	`getTrait("nav-left")` [auto | self | <FuncIRI>]	`setTrait("nav-left", ...)` [auto | self | <FuncIRI>]	"auto"	
offset, attribute	`getFloatTrait("offset")` [0 ≤ *value* ≤ 1]	`setFloatTrait("offset", ...)` [0 ≤ *value* ≤ 1]	0.0	
opacity, property	`getFloatTrait("opacity")` [0 ≤ *value* ≤ 1]	`setFloatTrait("opacity", ...)` [0 ≤ *value* ≤ 1]	1.0	

		`setTrait("opacity", ...)` [inherit]		
path, attribute	`getPathTrait("path")` [`SVGPath`]	`setPathTrait("path", ...)` [`SVGPath`]	An `SVGPath` object with no path segments	See Accessing rules for path attributes.
points, attribute	`getFloatListTrait("points")`	`setFloatListTrait("points", ...)`		
r, attribute	`getFloatTrait("r")` [*value* ≥ 0]	`setFloatTrait("r", ...)` [*value* ≥ 0]	0.5 when accessed on a **'radialGradient'** element, 0.0 otherwise	
repeatCount, attribute	N/A	`setTrait("repeatCount", ...)`		
repeatDur, attribute	N/A	`setTrait("repeatDur", ...)`		
restart, attribute	`getTrait("restart")` [always | whenNotActive | never]	`setTrait("restart", ...)` [always | whenNotActive | never]	"always"	
rx, attribute	`getFloatTrait("rx")` [*value* ≥ 0]	`setFloatTrait("rx", ...)` [*value* ≥ 0]	0.0	
ry, attribute	`getFloatTrait("ry")` [*value* ≥ 0]	`setFloatTrait("ry", ...)` [*value* ≥ 0]	0.0	
snapshotTime, attribute	`getFloatTrait("snapshotTime")` [*value* ≥ 0]	`setFloatTrait("snapshotTime", ...)` [*value* ≥ 0]	0.0	

solid-color, property	`getRGBColorTrait("solid-color")`
	[null, `SVGRGBColor`]
		`setRGBColorTrait("solid-color", ...)`
		[`SVGRGBColor`]
		
		`setTrait("solid-color", ...)`
		[<color> | inherit]
			rgb(0,0,0)	

solid-opacity, property	`getFloatTrait("solid-opacity")`
	[0 ≤ *value* ≤ 1]
		`setFloatTrait("solid-opacity", ...)`
		[0 ≤ *value* ≤ 1]
		
		`setTrait("solid-opacity")`
		[inherit]
			1.0	

stop-color, property	`getRGBColorTrait("stop-color")`
	[null, `SVGRGBColor`]
	
	`getTrait("stop-color")`
	[none | <color>]
		`setRGBColorTrait("stop-color", ...)`
		[`SVGRGBColor`]
		
		`setTrait("stop-color")`
		[none | currentColor | <color> | inherit]
			rgb(0,0,0)	

stop-opacity, property	`getFloatTrait("stop-opacity")`
	[0 ≤ *value* ≤ 1]
		`setFloatTrait("stop-opacity", ...)`
		[0 ≤ *value* ≤ 1]
		
		`setTrait("stop-opacity", ...)`
		[inherit]
			1.0	

stroke, property	`getRGBColorTrait("stroke")` [null, `SVGRGBColor`] `getTrait("stroke")` [none | <FuncIRI> | <color> | <system paint>]	`setRGBColorTrait("stroke", ...)` [`SVGRGBColor`] `setTrait("stroke", ...)` [none | currentColor | <FuncIRI> | <color> | <system paint> | inherit]	"none"	
stroke-dasharray, property	`getFloatListTrait("stroke-dasharray")` `getTrait("stroke-dasharray")` [none, inherit]	`setFloatListTrait("stroke-dasharray", ...)` `setTrait("stroke-dasharray", ...)` [none, inherit]	"none"	
stroke-dashoffset, property	`getFloatTrait("stroke-dashoffset")`	`setFloatTrait("stroke-dashoffset", ...)` `setTrait("stroke-dashoffset", ...)` [inherit]	0.0	
stroke-linecap, property	`getTrait("stroke-linecap")` [butt | round | square]	`setTrait("stroke-linecap", ...)` [butt | round | square | inherit]	"butt"	
stroke-linejoin, property	`getTrait("stroke-linejoin")` [miter | round | bevel]	`setTrait("stroke-linejoin", ...)` [miter | round | bevel | inherit]	"miter"	
stroke-miterlimit, property	`getFloatTrait("stroke-miterlimit")` [$value \geq 1$]	`setFloatTrait("stroke-miterlimit")` [$value \geq 1$] `setTrait("stroke-miterlimit", ...)` [inherit]	4.0	
stroke-opacity, property	`getFloatTrait("stroke-opacity")` [$0 \leq value \leq 1$]	`setFloatTrait("stroke-opacity", ...)` [$0 \leq value \leq 1$] `setTrait("stroke-opacity", ...)` [inherit]	1.0	
stroke-width, property	`getFloatTrait("stroke-width")` [$value \geq 0$]	`setFloatTrait("stroke-width", ...)` [$value \geq 0$] `setTrait("stroke-width", ...)` [inherit]	1.0	
target, attribute	`getTrait("target")`	`setTrait("target", ...)`	"_self"	
text-anchor, property	`getTrait("text-anchor")` [start | middle | end]	`setTrait("text-anchor", ...)` [start | middle | end | inherit]	"start"	
to, attribute	N/A	`setTrait("to", ...)`		

transform, attribute	`getMatrixTrait("transform")` [`SVGMatrix`] `getTrait("transform")` [<transform-list> | <transform-ref>]	`setMatrixTrait("transform", ...)` [`SVGMatrix`] `setTrait("transform", ...)` [<transform-list> | <transform-ref> | none]	Identity matrix (1,0,0,1,0,0)	See Accessing rules for **'transform'** attribute.
type, attribute	`getTrait("type")` [translate | scale | rotate | skewX | skewY]	`setTrait("type", ...)` [translate | scale | rotate | skewX | skewY]		These are the tra accessors for th **'type'** attribute c the **'animateTransfor** element. There a no trait accesso for the **'type'** at tribute on the **'audio'**, **'handler** **'image'**, **'script'** ar **'video'** element
values, attribute	N/A	`setTrait("values", ...)`		
vector-effect, property	`getTrait("vector-effect")` [none | non-scaling-stroke]	`setTrait("vector-effect", ...)` [none | non-scaling-stroke | inherit]	"none"	
version, attribute	`getTrait("version")`	`setTrait("version", ...)`	User agent specific	
viewBox, attribute	`getRectTrait("viewBox")` [`null` | `SVGRect`]	`setRectTrait("viewBox", ...)` [`SVGRect`] `setTrait("viewBox", ...)` [none]	`null`	If the **'viewBox'** a tribute has the value **'none'**, th `getRectTrait` met od will return `nul`
viewport-fill, property	`getRGBColorTrait("viewport-fill")` [null, `SVGRGBColor`] `getTrait("viewport-fill")` [none | <color>]	`setRGBColorTrait("viewport-fill", ...)` [`SVGRGBColor`] `setTrait("viewport-fill", ...)` [none | currentColor | <color> | inherit]	"none"	
viewport-fill-opacity, property	`getFloatTrait("viewport-fill-opacity")` [0 ≤ *value* ≤ 1]	`setFloatTrait("viewport-fill-opacity", ...)` [0 ≤ *value* ≤ 1] `setTrait("viewport-fill-opacity", ...)` [inherit]	1.0	
visibility, property	`getTrait("visibility")` [visible | hidden]	`setTrait("visibility", ...)` [visible | hidden | inherit]	"visible"	
width, attribute	`getFloatTrait("width")` [*value* ≥ 0] `getTrait("width")` ["auto"]	`setFloatTrait("width", ...)` [*value* ≥ 0] `setTrait("width", ...)` ["auto"]	"auto" when accessed on a **'textArea'** element, 0.0 otherwise	See Accessing rules for **'width** and **'height'** attributes.

x, attribute	`getFloatTrait("x")` `getFloatListTrait("x")`	`setFloatTrait("x", ...)` `setFloatListTrait("x", ...)`	0.0	See Accessing rules for 'x' and ' attributes.
x1, attribute	`getFloatTrait("x1")`	`setFloatTrait("x1", ...)`	0.0	
x2, attribute	`getFloatTrait("x2")`	`setFloatTrait("x2", ...)`	1.0 when accessed on a **'linearGradient'** element, 0.0 otherwise	
xlink:href, attribute	`getTraitNS("http://www.w3.org/1999/xlink", "href")` [absolute IRI]	`setTraitNS("http://www.w3.org/1999/xlink", "href", ...)`	""	
y, attribute	`getFloatTrait("y")` `getFloatListTrait("y")`	`setFloatTrait("y", ...)` `setFloatListTrait("y", ...)`	0.0	See Accessing rules for 'x' and ' attributes.
y1, attribute	`getFloatTrait("y1")`	`setFloatTrait("y1", ...)`	0.0	
y2, attribute	`getFloatTrait("y2")`	`setFloatTrait("y2", ...)`	0.0	
zoomAndPan, attribute	`getTrait("zoomAndPan")` [disable | magnify]	`setTrait("zoomAndPan", ...)` [disable | magnify]	"magnify"	

A.8.14 Additional accessing rules

Accessing rules for 'transform' attribute

The **'transform'** attribute in SVG Tiny 1.2 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'**, `getTrait` must be used. By using `setTrait` the user 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 SVG Tiny 1.2 user agent is allowed to normalize its attribute data as described in Display normalization.

Accessing rules for animation related elements

The following rule applies to SMIL animation elements (**'animate'**, **'animateTransform'**, **'animateColor'**, **'animateMotion'**, **'set'** , **'discard'**).

These elements can be inserted and removed from the tree but they cannot be modified using the `TraitAccess` methods once inserted into the tree. If an attempt is made to do so, the `TraitAccess` method will throw a `DOMException` with code NOT_SUPPORTED_ERR. Modifying the element using the `setAttribute` and `setAttributeNS` methods of the `Element` interface will change the document, but will have no effect on the animation.

This restriction means that if the author wishes to add animations via script, the attributes of the animation elements must be modified before being inserted into the tree. The following is an example of adding an animation to the document, setting the relevant attributes prior to insertion.

Example: Animating via the uDOM

```
<svg version="1.2" baseProfile="tiny" xmlns="http://www.w3.org/2000/svg" xml:id="svg-root"
     width="100%" height="100%" viewBox="0 0 480 360">
   <rect xml:id="myRect" fill="green" x="10" y="10" width="200" height="100" stroke="black" stroke-width="2"/>
</svg>
```

A script such as the following Java code 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);
```

Accessing rules for 'x' and 'y' attributes

If `getFloatTrait` is used to retrieve the value of the **'x'** or **'y'** attribute on a **'text'** element, where the attribute value has more than one <coordinate>, a `DOMException` with error code TYPE_MISMATCH_ERR must be raised. When the attribute value has only one coordinate, this is the value returned by `getFloatTrait`.

If `getFloatListTrait` or `setFloatListTrait` are used on elements other than a **'text'** element, a `DOMException` with error code TYPE_MISMATCH_ERR must be raised.

Accessing rules for 'width' and 'height' attributes

If `getFloatTrait` is used to retrieve the value of the **'width'** or **'height'** attribute on an **'svg'** element, where the value is specified as a percentage or unit value, a `DOMException` with error code TYPE_MISMATCH_ERR must be raised.

Accessing rules for font properties

If trait accessors are used to get or set the **'font-family'**, **'font-style'** or **'font-weight'** properties on a **'font-face'** element, a `DOMException` with error code NOT_SUPPORTED_ERR must be raised.

Accessing rules for path attributes

If the getPathTrait method is used to get an `SVGPath` and the path attribute was syntactically invalid at some point, the return value must be an SVGPath containing all valid path segments until the point of the error.

Accessing rules for multimedia elements

The following rule applies to multimedia elements (**'audio'**, **'video'**, **'animation'**).

SVG timing attributes cannot be modified using the `TraitAccess` methods once inserted into the tree. If an attempt is made to do so, the `TraitAccess` method will throw a `DOMException` with code NOT_SUPPORTED_ERR. Modifying these attributes using the `setAttribute` and `setAttributeNS` methods of the `Element` interface will change the document, but will have no effect on the element.

A.8.15 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
{
        attribute DOMString id;
};
```

No defined constants

Attributes

id

On read, returns the element's **'xml:id'** or **'id'** attribute according to the rules defined in the Structure chapter, or `null` if no ID specified.

On write, sets the element's **'xml:id'** or **'id'** attributes according to the rules defined in the Structure chapter.

No defined methods

A.8.16 SVGTimedElement

This interface represents an `SVGTimedElement` which is implemented by timed elements and the **'svg'** element.

IDL Definition

```
interface SVGTimedElement : SVGElement, smil::ElementTimeControl
{
        void pauseElement();
        void resumeElement();
        readonly attribute boolean isPaused;
};
```

No defined constants

Attributes

isPaused

`true` if the timed element is paused. `false` otherwise. See Paused element and the active duration ([SMIL21], section 10.4.3). Note that an element that is stopped (has reached the end of its active duration) is not paused.

Methods

pauseElement

Pauses the timed element. See Paused element and the active duration ([SMIL21], section 10.4.3).

No parameters

No return value

No exceptions

resumeElement

Resumes the timed element. See Paused element and the active duration ([SMIL21], section 10.4.3).

No parameters

No return value

No exceptions

A.8.17 SVGAnimationElement

This interface is implemented by the following SMIL animation elements: **'animate'**, **'animateTransform'**, **'animateColor'**, **'animateMotion'** and **'set'**. It is included for historical reasons and has been deprecated. Note that this interface is unrelated to the new **'animation'** element.

IDL Definition

```
interface SVGAnimationElement : SVGTimedElement
{
};
```

No defined constants

No defined attributes

No defined methods

A.8.18 SVGVisualMediaElement

This interface represents a media element that is visual, i.e. has a physical location on the screen. It is implemented by: **'animation'** and **'video'**.

IDL Definition

```
interface SVGVisualMediaElement : SVGLocatableElement, SVGTimedElement
{
};
```

No defined constants
No defined attributes
No defined methods

A.8.19 SVGTimer

The `SVGTimer` interface provides an API for scheduling a one time or repetitive event. A `SVGTimer` object is always either in the running (attribute `running` is `true`) or waiting (attribute `running` is `false`) state. After each interval of the timer, an `Event` of type `SVGTimer` is triggered.

`SVGTimer` events are triggered only when the `timer` is in the `running` state. The `SVGTimer` event is limited to the target phase. Since `SVGTimer` is an `EventTarget`, `EventListener`s can be registered on it using `addEventListener` with `SVGTimer` as the event type. Event listeners can access their corresponding SVGTimer object through the event object's target property.

`SVGTimer` instances are created using the `createTimer` method of the `SVGGlobal` interface.

IDL Definition

```
interface SVGTimer : events::EventTarget
{
        attribute long delay;
        attribute long repeatInterval;
        readonly attribute boolean running;
        void start();
        void stop();
};
```

No defined constants
Attributes

delay

This attribute specifies the time remaining in milliseconds until the next event is fired. When the `SVGTimer` is in the `running` state this attribute is dynamically updated to reflect the remaining time in the current interval. When the `SVGTimer` is `waiting` the delay reflects the time that remained when stopped. Getting the delay attribute returns the current value, i.e. a snapshot value of the remaining delay. After delay period has passed while the object is in the `running` state, the `SVGTimer` object will trigger an `Event` of type `SVGTimer`. The delay will then be updated with the `repeatInterval` value and a new count down will start. Setting the delay resets the current interval to the new value. If this attribute is `0`, it means that the event will be triggered as soon as possible. Assigning a negative value is equivalent to calling the `stop()` method. The initial value is set through the initialInterval parameter in the `createTimer` method on the `SVGGlobal` interface, and defines the first interval of the `SVGTimer`.

repeatInterval

This attribute specifies in milliseconds the interval for each repeat of the `SVGTimer`, i.e. each timer interval subsequent to the initial interval. The initial value of this attribute is set through the `repeatInterval` parameter in the `createTimer` method on the `SVGGlobal` interface. Assigning a negative value disables the repetitive triggering of the event making it a one time timer which triggers an event after the `delay`.

running

`SVGTimer` state. Value is `true` if the timer is running, `false` if the timer is waiting. Note that the `repeatInterval` and `delay` properties can be non-negative if the timer is stopped (but if `delay` is negative, the timer is stopped).

Methods

start

Changes the `SVGTimer` state into running. If the timer is already in the running state, it has no effect. Initially the timer is waiting, and must be started with this method. If the timer `delay` had a negative value when started, for example if the time had been stopped by setting the delay to a negative value, the delay value is reset to `repeatInterval` when the this method is called.

No parameters

No return value

No exceptions

stop

Changes the `SVGTimer` state into waiting. If the timer is already in the waiting state, calling this method has no effect.

No parameters

No return value

No exceptions

A.8.20 SVGGlobal

Many scripted SVG documents in existence make use of functions on a browser specific Window interface. SVG Tiny 1.2 specifies an `SVGGlobal` interface, on which some of these de facto standard functions are defined, as well as some functions for new features defines in this specification. The `SVGGlobal` interface must be implemented by the object that represents the default view of the document ([DOM2VIEWS], section 1.1). This object also implements `AbstractView`. Thus, in the ECMAScript language binding, the global script object implements `SVGGlobal`. The `SVGGlobal` object for a document can also be obtained through `DocumentView::defaultView`.

IDL Definition

```
interface SVGGlobal
{
        SVGTimer createTimer(in long initialInterval, in long repeatInterval);
        void getURL(in DOMString iri, in AsyncStatusCallback callback);
        void 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
No defined attributes
Methods

createTimer

Creates a `SVGTimer` with the provided initial and repeat Intervals. The `SVGTimer` will initially be in the waiting state.

Parameters

in long initialInterval	Specifies the first interval in milliseconds for a repetitive `SVGTimer`, i.e. sets the initial value of the `delay` attribute on the timer. In the case the `SVGTimer` is not repetitive, it specifies the interval for the one time timer. Setting this parameter with a negative value will create an `SVGTimer` which is in the waiting state.
in long repeatInterval	Specifies the time interval on which the `SVGTimer` repeats subsequent to the initial interval. A negative value will make the `SVGTimer` a one time timer.

Return value

`SVGTimer`	The created `SVGTimer`.

No exceptions

getURL

Given an IRI and an `AsyncStatusCallback` object on which to call a callback function, this method will attempt to fetch the resource at that IRI. If the IRI uses the HTTP or HTTPS scheme, the HTTP GET method will be used. Implementations may support other schemes, but are not required to.

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 for HTTP requests and must decode such content before passing it on to the callback. User agents are not required to support gzip 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 are not required to.

It is important to note that for security reasons, user agents are strongly encouraged to restrict these requests by origin. 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 status 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 call a callback function, a media type, and a content coding, this method will post the data to the specified IRI using the requested media type and content coding. User agents must support `postURL` being invoked with an HTTP or HTTPS IRI, but may support other IRI schemes if they indicate protocols that are functionally compatible with HTTP. Once the request has been completed the callback is invoked as described in the `AsyncStatusCallback` interface. If `postURL` is invoked with an IRI that does not support posting content, or which does not post content in a manner compatible with HTTP, a `DOMException` with code NOT_SUPPORTED_ERR must be thrown.

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` iri	The IRI of the resource that is being requested.
in `DOMString` data	The data that will be the body of the POST request.
in `AsyncStatusCallback` callback	The object on which the callback will be called upon completion of the request.
in `DOMString` type	The content type of the POST request.
in `DOMString` encoding	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 1.0 or Namespaces in XML 1.1 respectively, this method must return a `null` value.

When parsing the input string, the `contextDoc` parameter is used only for setting the `ownerDocument` field in the parsed nodes.

If during parsing a **'script'** element is encountered, it must not be executed at that time. It will only be executed if it inserted into the current SVG document.

There is no requirement to load any external resources, e.g. external entities, stylesheets, scripts, raster images, video, audio, etc., for the parsing to complete. XSL stylesheets must not be applied.

If the `contextDoc` parameter 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 `contextDoc` 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

Parameters

in `DOMString` data	The data that is to be parsed as XML.
in `Document` contextDoc	The `Document` object in the context of which to perform the parsing.

Return value

`Node`	A `Node` (either a `Document` or an `Element`) representing the content that was parsed.

Exceptions

`DOMException`	INVALID_CHARACTER_ERR: Raised if one of the parsed XML names is not an XML name according to the XML version of the contextDoc document.

A.8.21 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. The `operationComplete` method of the object implementing this interface is called 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 fetching a resource using `getURL` or `postURL`. Upon completion of the request, the method is called with an `AsyncURLStatus` object that holds the resource contents and information about the request.

Parameters

in `AsyncURLStatus`	status	An object representing the HTTP response.

No return value
No exceptions

A.8.22 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 HTTP requests with response status codes in the 200 range, this attribute must be set to true, and for status codes in the 400 and 500 ranges it must be set to false. Status codes in the 100 range must be ignored and those in the 300 range must be processed as indicated in `getURL`'s processing requirements.

When fetching non-HTTP resources, this attribute must be set to true if the resource was successfully retrieved in full, and false otherwise.

contentType

A string containing the media type of the response.

For HTTP requests, this attribute must be set to the value of the Content-Type HTTP header. If there was no Content-Type header, the attribute must be set to `null`.

When fetching non-HTTP resources, this attribute must be set to `null`.

content

A string containing the contents of the fetched resource.

If the resource is not a valid sequence of characters (as interpreted according to the media type and other headers for an HTTP request, or as appropriate for non-HTTP resources), then this attribute must be set to null.

For HTTP requests, if the media type of the response 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 response body had one or more content codings applied to it then it must be fully decoded before setting this field. If the HTTP response status code was an error code but carried a body, the content of that body must still be exposed.

No defined methods

A.8.23 EventListenerInitializer2

The `EventListenerInitializer2` interface is used to provide a way for scripts written in languages that do not have a concept of a "global object" to initialize their event listeners. Specifically, it is used for Java event listeners, but this general approach is suggested for other such scripting languages. See the description of the **'script'** element for details on how the object implementing `EventListenerInitializer2` is discovered and used.

IDL Definition

```
interface EventListenerInitializer2
{
        void initializeEventListeners(in Element scriptElement);
        EventListener createEventListener(in Element handlerElement);
};
```

No defined constants

No defined attributes

Methods

initializeEventListeners

Invoked to indicate that the script given by the specified **'script'** element should be executed.

Parameters

in Element scriptElement	The **'script'** element that identifies the script to execute.

No return value

No exceptions

createEventListener

Invoked to obtain an `EventListener` that corresponds to the specified **'handler'** element.

Parameters

in Element handlerElement	The **'handler'** element for which a corresponding `EventListener` is to be returned.

Return value

EventListener	The `EventListener`.

No exceptions

B IDL Definitions

This appendix is normative.

This appendix contains the complete 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.

```
module dom
{
    valuetype DOMString sequence<unsigned short>;
    typedef Object DOMObject;
    interface Node;
    interface Element;
    interface Document;

    exception DOMException
    {
        unsigned short code;
    };

    const unsigned short      INDEX_SIZE_ERR                 = 1;
    const unsigned short      DOMSTRING_SIZE_ERR             = 2;
    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_DATA_ALLOWED_ERR            = 6;
    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      INUSE_ATTRIBUTE_ERR            = 10;
    const unsigned short      INVALID_STATE_ERR              = 11;
    const unsigned short      SYNTAX_ERR                     = 12;
    const unsigned short      INVALID_MODIFICATION_ERR       = 13;
    const unsigned short      NAMESPACE_ERR                  = 14;
    const unsigned short      INVALID_ACCESS_ERR             = 15;
    const unsigned short      VALIDATION_ERR                 = 16;
    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, ElementTraversal
    {
        DOMString getAttributeNS(in DOMString namespaceURI, in DOMString localName)
                raises(DOMException);
        void setAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName,
                            in DOMString value)
                                            raises(DOMException);
        DOMString getAttribute(in DOMString name);
        void setAttribute(in DOMString name, in DOMString value)
                raises(DOMException);
    };

    interface Document : Node
    {
        Element createElementNS(in DOMString namespaceURI, in DOMString qualifiedName)
                raises(DOMException);
        readonly attribute Element documentElement;
        Element getElementById(in DOMString elementId);
    };

    interface ElementTraversal
    {
        readonly attribute Element firstElementChild;
        readonly attribute Element lastElementChild;
        readonly attribute Element nextElementSibling;
        readonly attribute Element previousElementSibling;
        readonly attribute unsigned long childElementCount;
    };

    interface Location
    {
        void assign(in DOMString iri);
        void reload();
    };

    interface Window
    {
        readonly attribute Window parent;
        readonly attribute Location location;
    };

};

module views
{
    interface DocumentView;

    interface AbstractView
    {
        readonly attribute DocumentView document;
    };

    interface DocumentView
    {
        readonly attribute AbstractView defaultView;
    };

};

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);
        void removeEventListener(in DOMString type, in EventListener listener,
                                 in boolean useCapture);
    };

    interface EventListener
    {
        void handleEvent(in Event evt);
    };

    interface Event
    {
        readonly attribute EventTarget target;
        readonly attribute EventTarget currentTarget;
        readonly attribute DOMString type;
        readonly attribute boolean cancelable;
        readonly attribute boolean defaultPrevented;
        void stopPropagation();
        void preventDefault();
    };

    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 MouseWheelEvent : MouseEvent
    {
        readonly attribute long wheelDelta;
    };

    interface TextEvent : UIEvent
    {
        readonly attribute DOMString data;
    };

    interface KeyboardEvent : UIEvent
    {
        readonly attribute DOMString keyIdentifier;
    };

    interface UIEvent : Event
    {
        readonly attribute long detail;
    };

    interface ProgressEvent : Event
    {
        readonly attribute boolean lengthComputable;
        readonly attribute unsigned long loaded;
        readonly attribute unsigned long total;
    };

};

module smil
{
    typedef events::Event Event;

    interface ElementTimeControl
    {
        void beginElementAt(in float offset);
        void beginElement();
        void endElementAt(in float offset);
        void endElement();
    };

    interface TimeEvent : Event
    {
        readonly attribute long detail;
    };

};

module svg
{
    typedef dom::DOMString DOMString;
    typedef dom::DOMException DOMException;
    typedef dom::DOMObject DOMObject;
    typedef dom::Document Document;
    typedef dom::Element Element;
    typedef dom::Node Node;
    typedef events::Event Event;
    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 SVGTimedElement;
interface SVGAnimationElement;
interface SVGDocument;
interface SVGGlobal;

exception SVGException
{
    unsigned short code;
};

const unsigned short SVG_WRONG_TYPE_ERR         = 0;
const unsigned short SVG_INVALID_VALUE_ERR      = 1;
const unsigned short SVG_MATRIX_NOT_INVERTABLE  = 2;

interface SVGDocument : Document, EventTarget
{
};

interface SVGUseElement : SVGLocatableElement
{
};

interface SVGElementInstance : EventTarget
{
    readonly attribute SVGElement correspondingElement;
    readonly attribute SVGUseElement correspondingUseElement;
};

interface SVGSVGElement : SVGLocatableElement, SVGTimedElement
{
    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;
    float getCurrentTime();
    void setCurrentTime(in float seconds);
    SVGMatrix createSVGMatrixComponents(in float a, in float b, in float c, in float d,
                                        in float e, in float f);
    SVGRect createSVGRect();
    SVGPoint createSVGPoint();
    SVGPath createSVGPath();
    SVGRGBColor createSVGRGBColor(in float red, in float green, in float blue)
            raises(SVGException);
    void moveFocus(in unsigned short motionType)
            raises(DOMException);
    void setFocus(in EventTarget theObject)
            raises(DOMException);
    EventTarget getCurrentFocusedObject();
};

interface SVGRGBColor
{
    attribute unsigned long red;
    attribute unsigned long green;
```

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

    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);
        sequence<float> getFloatListTrait(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);
    sequence<float> getFloatListPresentationTrait(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 setFloatListTrait(in DOMString name, in sequence<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 SVGElement : Element, EventTarget, TraitAccess
{
    attribute DOMString id;
};

interface SVGTimedElement : SVGElement, smil::ElementTimeControl
{
    void pauseElement();
    void resumeElement();
    readonly attribute boolean isPaused;
};

interface SVGAnimationElement : SVGTimedElement
{
};

interface SVGVisualMediaElement : SVGLocatableElement, SVGTimedElement
{
};

interface SVGTimer : events::EventTarget
{
   attribute long delay;
   attribute long repeatInterval;
   readonly attribute boolean running;
   void start();
   void stop();
};

interface SVGGlobal
{
   SVGTimer createTimer(in long initialInterval, in long repeatInterval);
   void getURL(in DOMString iri, in AsyncStatusCallback callback);
   void 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;
    };

    interface EventListenerInitializer2
    {
        void initializeEventListeners(in Element scriptElement);
        EventListener createEventListener(in Element handlerElement);
    };

};
```

C Implementation Requirements

Contents

C.1 Introduction 338
C.2 Unsupported elements, attributes, properties, attribute values and property values 338
C.3 Error processing 338
C.4 Namespace, version, baseProfile, requiredFeatures and requiredExtensions 339
C.5 Clamping of color and opacity Values 339
C.6 'path' element implementation notes 340
C.7 Text selection implementation notes 340
C.8 Printing implementation notes 341

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 treat unknown attributes, attribute values, styling properties, styling property values, and descendant elements as follows:

- Within an SVG document fragment, any subtree that is rooted by an unknown element (including those in the SVG or XML Events namespaces) or a known element that occurs in unexpected location, is not rendered. The nodes in the subtree are not processed beyond including the relevant DOM objects in the document tree. Those DOM objects will still implement the DOM interfaces appropriate for the element type, however.
- 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.
- 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.
- The user agent must not stop processing or rendering under any of the above situations, but at user option it may alert, log, or otherwise provide a notification that unsupported content was encountered.

In Example ignored-fill below, the **'fill'** property on the **'circle'** element is ignored due to the above rules. Since **'fill'** is an inherited property, 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>
```

When unsupported content is encountered, an SVG user agent which provides access to an error console should report the incidence of unsupported content, the location where it was encountered, and other appropriate messages that may help the author, to the error console.

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 version of XML used (either the XML 1.0 [XML10] or XML 1.1 specifications [XML11])

- When the content is not namespace-well-formed according to the Namespaces in XML 1.0 specification [XML-NS10] or the Namespaces in XML 1.1 specification [XML-NS] (depending on the version of XML used as per the previous bullet)
- 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 uDOM 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 SVG user agent must provide a highly perceivable indication of error. Additionally, an SVG user agent which provides access to an error console should report the error, the location where it was encountered, and other appropriate messages that may help the author, to the error console.

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

SVG user agents must only consider elements explicitly placed in the SVG namespace as being SVG elements. For example, when parsing a document, SVG user agents must only consider elements explicitly placed in the SVG namespace by XML namespace declarations in the document to be SVG elements. Similarly, any element created with the `Document::createElementNS` uDOM method must have had the SVG namespace passed as the namespaceURI parameter for it to be considered considered to be an SVG element.

SVG content can use attributes **'requiredFeatures'** and **'requiredExtensions'** to provide explicit indication of the minimal features that must be supported by the SVG user agent in order to render the SVG content correctly. SVG content can also use the **'version'** attribute 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 SVG user agent 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 SVG user agent should a newer version be available which does support these features.

Note that the intent of the version number is to indicate the minimum implementation version for the content; an SVG user agent which encounters content with a version number unknown to the user agent must still render those elements and attributes that it does support, and must not render elements or document fragments with root elements which it does not support.

C.5 Clamping of color 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 SVG 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:

```
(newx1, newy1) = (curx - (oldx2 - curx), cury - (oldy2 - cury))
               = (2 * curx - oldx2, 2 * cury - oldy2)
```

C.7 Text selection implementation notes

This specification does not mandate any particular scheme for visual feedback for text selection, but a conforming SVG user agent should provide such feedback as is consistent with system norms.

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 behavior 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 content 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 selection, the user agent must provide a mechanism for selecting text with a full range of available UI devices (e.g. keyboard, mouse, pen, or any other pointer device) as a default, even when the given text is part of a link. One implementation option for platforms which support selection via a pointer device such as a mouse is for the user agent to provide for a small additional region around character cells which initiates text selection operations but does not initiate event handlers or links. Additionally, the user agent must allow scripted or programmatic DOM access to all text content, and specifically any selected or focused content, at all times.

C.8 Printing implementation notes

In 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 342
D.2 Terminology 342
D.3 SVG content conformance........ 342
D.3.1 Conforming SVG Document Fragments........ 342
D.3.2 Conforming SVG Stand-Alone Documents 343
D.3.3 Conforming SVG Included Document Fragments 343
D.3.4 Conditionally Conforming SVG Tiny 1.2 Document Fragments........ 344
D.4 SVG writer conformance........ 344
D.4.1 Conforming SVG Generators 344
D.4.2 Conforming SVG Authoring Tools 344
D.4.3 Conforming SVG Servers 344
D.5 SVG reader conformance 345
D.5.1 Conforming SVG Interpreters........ 345
D.5.2 Conforming SVG Viewers........ 345
D.6 Extension conformance requirements 347
D.7 Non-XML encoding conformance requirements........ 347

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 runtime 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 [RFC2119]. However, for readability, these words do not necessarily 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) Tiny 1.2 Specification) including SVG's schema (see RelaxNG schema) and also:

- Is well-formed according to the version of XML used (either the XML 1.0 [XML10] or XML 1.1 [XML11]) and conforms to the corresponding Namespaces in XML specification (Namespaces in XML 1.0 [XML-NS10] or Namespaces in XML 1.1 [XML-NS]).
- Conforms to all applicable 'C' conformance criteria in Character Model for the World Wide Web 1.0: Fundamentals [CHARMOD].
- 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/nvdl/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 a 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 the Namespaces in XML 1.0 specification [XML-NS10] or the Namespaces in XML 1.1 specification [XML-NS] (depending on the version of XML used).

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 conditional processing 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 using a **'requiredFeatures'** attribute to require a feature not defined in this specification are marked as false
2. All elements using a **'requiredExtensions'** attribute to require an extension 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 documents or document fragments that satisfy the criteria of at least one of the following classes: Conforming SVG Document Fragments, Conforming SVG Stand-Alone Documents and 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, an SVG Generator should prefer the **'id'** attribute over the **'xml:id'** attribute [XMLID] for content that is known to target SVG viewers, and **'xml:id'** over **'id'** for content that is known to target generic XML processors. An SVG Generator must not include both attributes for the same document on elements in the SVG namespace.

SVG generators are encouraged to follow W3C developments in the area of internationalization such as Character Model for the World Wide Web 1.0: Normalization [CHARMOD-NORM]. Future versions of the SVG specification may require support of that specification 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 file is compressed with gzip or deflate, Conforming SVG Servers must indicate this with the appropriate header, according to what the protocol supports. Specifically, for content compressed by the server immediately prior to transfer, the server must use the "Transfer-Encoding: gzip" or "Transfer-Encoding: deflate" headers as appropriate, and for content stored in a compressed format on the server (e.g. with the file extension "svgz"), the server must use the "Content-Encoding: gzip" or "Content-Encoding: deflate" headers as appropriate.

Note: Compression of stored *content* (the "entity," in HTTP terms) is distinct from automatic compression of the *message body*, as defined in HTTP/1.1 TE/ Transfer Encoding ([RFC2616], sections 14.39 and 14.41).

D.5 SVG reader conformance

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 a search engine (e.g. a service which indexes text, metadata, or other SVG content), 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 [SVGM11].

A *Conforming SVG Interpreter* must conform to all applicable 'I' conformance criteria in Character Model for the World Wide Web 1.0: Fundamentals [CHARMOD].

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.)

A Conforming SVG Interpreter which indexes SVG content (e.g. a search engine) must, at a minimum, extract and process all textual data, including the content of the text content elements and descriptive elements, with attention paid to author-supplied alternate languages for purpose of presentation and translation. Additionally, it should process element types, document structure, metadata, and link data to inform the indexing. A Conforming SVG Interpreter which indexes images should categorize and represent SVG content as an image. Such an SVG Interpreter may apply heuristics to the geometric semantics of the SVG document or to the rendered image (such as performing shape-recognition) to improve indexing.

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. To ensure ease of navigation and maintaining context, SVG user agents are recommended to allow the user to specify a center point for zoom operations, and allow for multiple levels of zooming.
- 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 ([RFC2616], section 14.3) those encodings that it accepts, including at minimum gzip and deflate, and then decompress any gzip-encoded and deflate-encoded data streams that are downloaded from the server. When an SVG viewer retrieves compressed content (e.g., an .svgz file) over HTTP, if the "Content-Encoding" and "Transfer-Encoding" response headers are missing or specify a value that does not match the compression method that has been applied to the content, then the SVG viewer must not render the content and must treat the document as being in error. 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 color [SRGB], all sRGB color computations and all resulting color values must be accurate to within one sRGB color component value, where sRGB color component values are integers that 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.

D.6 Extension conformance requirements

Specifications and implementations are allowed to extend the SVG specification but in order to claim conformance the following criteria need to be met:

- An extension must support the normative chapter of the SVG specification that defines conformance to SVG.
- An extension must support the normative chapter of the SVG specification that details extensibility.
- An extension must support the normative chapter of the SVG specification that defines conditional processing.
- If using features defined in SVG Full, an extension must not redefine the syntax of the syntax of those features.
- An extension must not redefine the semantics of any existing SVG element or attribute.
- SVG attribute and element names must not be reused in extensions, even in a separate namespace, in order to avoid author confusion.

D.7 Non-XML encoding conformance requirements

An SVG Document Fragment, SVG Stand-Alone Document or SVG Included Document encoded using a non-XML encoding (e.g. efficient XML compression) conforms to this specification if and only if the non-XML encoding in question guarantees roundtripping from XML to the encoding and back in such a way that the resulting document when processed by an SVG user agent must always render, animate, and interact in the exact same way as the original. Note that this requires a high-level of fidelity from the encoding including, but not limited to, the ability to encode non-conforming content and content from foreign namespaces without loss, maintaining ID typing supplied by the internal subset of the formal part the Document Type Definition, and not removing non-rendered elements such as **'desc'**, **'title'**, or elements removed from the rendering tree through use of conditional processing attributes.

E Conformance to QA Framework Specification Guidelines

Contents

E.1 Introduction. 348
E.2 Checklist table. 348
E.3 List of deprecated features. 350

E.1 Introduction

This appendix is informative.

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			
Good Practice 09: Define unfamiliar terms in-line and consolidate the definitions in a glossary section.	YES			
Good Practice 10: Use terms already defined without changing their definition.	YES			
Requirement 07: Use a consistent style for conformance requirements and explain how to distinguish them.	YES			
Requirement 08: Indicate which conformance requirements are mandatory, which are recommended, and which are optional.	YES			
Good Practice 11: Use formal languages when possible.	yes			
Good Practice 12: Write Test Assertions.		no		
2.4 Managing Variability				
Good Practice 13: Create subdivisions of the technology when warranted.	YES			
Requirement 09: If the technology is subdivided, then indicate which subdivisions are mandatory for conformance.	YES			
Requirement 10: If the technology is subdivided, then address subdivision constraints.	YES			
Good Practice 14: If the technology is profiled, define rules for creating new profiles.	YES			
Good Practice 15:Use optional features as warranted.	YES			
Good Practice 16: Clearly identify optional features.	YES			
Good Practice 17: Indicate any limitations or constraints on optional features.	YES			
Requirement 11: Address Extensibility.	YES			
Good Practice 18: If extensibility is allowed, define an extension mechanism.	YES			
Good Practice 19: Warn extension creators to create extensions that do not interfere with conformance.	YES			
Good Practice 20: Define error handling for unknown extensions.	YES			
Requirement 12: Identify deprecated features.	YES			Deprecated features are clearly marked as such.
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			

E.3 List of deprecated features

- The following events have been deprecated:
  - `SVGLoad`
  - `SVGResize`
  - `SVGScroll`
  - `SVGZoom`
- The value **'text/ecmascript'** has been deprecated for the **'type'** attribute on the **'script'** element.

F Accessibility Support

Contents

F.1 WAI accessibility guidelines 351
F.2 SVG content accessibility guidelines. 351
F.3 SVG user agent accessibility guidelines 352

This appendix is informative.

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] and Web Content Accessibility Guidelines 2.0 [WCAG2] explain 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 to the Priority 1 accessibility guidelines defined in UAAG, and should conform also to Priorities 2 and 3. 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 content element) explains its function, no text equivalent is required. Use the **'title'** child element to explain the function of text content 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'** conditional processing 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 localizing 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.

Provide semantic metadata.

- Use the **'role'** attribute with appropriate values to indicate the functionality of elements.
- Indicate relationships between elements with extensible metadata attributes.
- Use structured metadata with well-known semantics that are understood by accessibility tools, rather than arbitrary values (e.g., use `"button"` instead of `"btn"`).

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 353
G.2 Internationalization and SVG 353
G.3 SVG internationalization guidelines 353
G.4 Markup for the internationalization and localization of SVG 353

This appendix is informative.

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].

G.4 Markup for the internationalization and localization of SVG

To help the international adoption and easy localization of SVG content, the SVG Tiny 1.2 RelaxNG schema contains declarations from the Internationalization Tag Set (ITS) 1.0 [ITS]. The usage of this markup vocabulary is explained in detail in the Best Practices for XML Internationalization document [XI18N-BP]. The markup from ITS 1.0 does not influence SVG Tiny 1.2 processing, following the definitions in the section 19.1 Foreign namespaces and private data.

ITS 1.0 local markup declarations have been added to the content model of the text content elements, the descriptive elements, and the **'metadata'** element. ITS enables a content author to express, for example, that some parts of an SVG document should not be translated during the localization process. This is demonstrated by the following example which contains ITS 1.0 markup. In this example, the **'its:translate'** attribute expresses that the content of the **'desc'** element should not be translated.

Example: its.svg

```
<svg version="1.2" baseProfile="tiny"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:its="http://www.w3.org/2005/11/its">

     <desc its:translate="no">An explanation which should not be translated.</desc>
```

```
    <!-- other content -->
</svg>
```

The following ITS rules file, `its-rules.xml`, specifies some defaults about the translatability of SVG Tiny 1.2 markup. Implementors are encouraged to provide theses rules to users with an international audience, for example as the content of the **'metadata'** element.

Example: its-rules.xml

```
<its:rules xmlns:its="http://www.w3.org/2005/11/its" version="1.0"
           xmlns:svg="http://www.w3.org/2000/svg">
  <its:translateRule selector="//svg:*" translate="no"/>
  <its:translateRule selector="//svg:text | //svg:tspan | //svg:textArea |
                               //svg:title | //svg:desc | //svg:metadata"
                               translate="yes"/>
</its:rules>
```

The **'its:translateRule'** elements above mean that the default for all elements from the SVG namespace is that they are not translatable (first **'its:translateRule'**), with the exception of the specific SVG elements listed (second **'its:translateRule'**).

H JPEG Support

Contents

H.1 Introduction 355
H.2 Required support 355

H.1 Introduction

This appendix is normative.

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.2 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 ([JPEG], section 4.7).
- Support for the DNL Marker ([JPEG], section 3.2) is not required.
- Support for non-integer sampling ratios is not required ([JPEG], section A.1.1).

The following encoding processes are not required, but may be supported:

- Complete Extended DCT-based processes
- Lossless Processes
- Hierarchical Processes

SVG Tiny 1.2 user agents should convert Y,Cb,Cr values compressed in the JPEG image to RGB as defined in the JFIF specification [JFIF] and may assume, in the absence of a color profile, 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 Béziers, quadratic Béziers) to prevent the need to tessellate into polylines; and shortcut formulations exist for common forms of cubic Bézier segments, quadratic Bézier segments, and horizontal and vertical straight line segments so that the minimum number of coordinates need to be specified.
- Text can be specified using XML content, thus there is 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 [RFC2616]. Here are some sample compression results using gzip compression [RFC1952] 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 lacuna values whenever possible rather than defining all attributes and properties explicitly.
- Take advantage of the path data compaction facilities: use relative coordinates; use *h* and *v* for horizontal and vertical lines; use *s* or *t* for cubic and quadratic Bézier segments whenever possible; eliminate extraneous white space and separators.
- Utilize symbols if the same graphic appears multiple times in the document.
- For user agents that support styling with CSS, utilize CSS property inheritance and selectors to consolidate commonly used properties into named styles or to assign the properties to a parent **'g'** element.

J Feature Strings

Contents

J.1 General feature strings 357
J.2 Specific feature strings 358

This appendix is normative.

The following are the feature strings for the **'requiredFeatures'** attribute. In some cases the feature strings map directly to SVG elements and attributes, in others they represent some functionality of the user agent (that it is a static viewer for example).

J.1 General feature strings

Support for a feature string indicates that the SVG Viewer can process and render successfully all of the language features listed in the feature column.

Feature String	Feature
http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static	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/#TextFlow, 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/#ExternalResourcesRequired, 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
http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static-DOM	Conformance to the uDOM according to Conforming to the SVG uDOM as well as support for: http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static, 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
http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-animated	http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static, http://www.w3.org/Graphics/SVG/feature/1.2/#TimedAnimation
http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-interactive	http://www.w3.org/Graphics/SVG/feature/1.2/#EditableTextAttribute, http://www.w3.org/Graphics/SVG/feature/1.2/#NavigationAttribute
http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all	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-interactive, http://www.w3.org/Graphics/SVG/feature/1.2/#Audio, http://www.w3.org/Graphics/SVG/feature/1.2/#Video,

	http://www.w3.org/Graphics/SVG/feature/1.2/#Animation, http://www.w3.org/Graphics/SVG/feature/1.2/#Discard, http://www.w3.org/Graphics/SVG/feature/1.2/#MediaAttribute

J.2 Specific feature strings

Support for a feature string indicates that the SVG Viewer can process and render successfully all of the elements, attributes and language features listed in the feature column.

Feature String	Feature
http://www.w3.org/Graphics/SVG/feature/1.2/#CoreAttribute	**'about', 'class', 'content', 'datatype', 'id', 'property', 'rel', 'resource', 'rev', 'role', 'typeof', 'xml:base', 'xml:id'**
http://www.w3.org/Graphics/SVG/feature/1.2/#NavigationAttribute	**'focusHighlight', 'focusable', 'nav-down', 'nav-down-left', 'nav-down-right', 'nav-left', 'nav-next', 'nav-prev', 'nav-right', 'nav-up', 'nav-up-left', 'nav-up-right', 'pointer-events'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Structure	**'defs', 'desc', 'g', 'metadata', 'svg', 'title', 'use'**
http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessing	**'switch'**
http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessingAttribute	**'requiredExtensions', 'requiredFeatures', 'requiredFonts', 'requiredFormats', 'systemLanguage'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Image	**'image'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Prefetch	**'prefetch'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Discard	**'discard'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Shape	**'circle', 'ellipse', 'line', 'path', 'polygon', 'polyline', 'rect'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Text	**'text', 'tspan', 'font-family', 'font-size', 'font-style', 'font-variant', 'font-weight', 'text-anchor'**
http://www.w3.org/Graphics/SVG/feature/1.2/#PaintAttribute	**'color', 'color-rendering', 'fill', 'fill-rule', 'solid-color', 'stroke', 'stroke-dasharray', 'stroke-dashoffset', 'stroke-linecap', 'stroke-linejoin', 'stroke-miterlimit', 'stroke-width', 'viewport-fill'**
http://www.w3.org/Graphics/SVG/feature/1.2/#OpacityAttribute	**'fill-opacity', 'opacity', 'solid-opacity', 'stop-opacity', 'stroke-opacity', 'viewport-fill-opacity'**
http://www.w3.org/Graphics/SVG/feature/1.2/#GraphicsAttribute	**'buffered-rendering', 'display', 'image-rendering', 'pointer-events', 'shape-rendering', 'text-rendering', 'vector-effect', 'visibility'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Gradient	**'linearGradient', 'radialGradient', 'stop', 'stop-color'**
http://www.w3.org/Graphics/SVG/feature/1.2/#SolidColor	**'solidColor'**

http://www.w3.org/Graphics/SVG/feature/1.2/#Hyperlinking	**'a'**
http://www.w3.org/Graphics/SVG/feature/1.2/#XlinkAttribute	**'xlink:actuate'**, **'xlink:arcrole'**, **'xlink:href'**, **'xlink:role'**, **'xlink:show'**, **'xlink:title'**, **'xlink:type'**
http://www.w3.org/Graphics/SVG/feature/1.2/#ExternalResourcesRequired	**'externalResourcesRequired'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Scripting	**'script'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Handler	**'handler'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Listener	**'listener'**
http://www.w3.org/Graphics/SVG/feature/1.2/#TimedAnimation	**'animate'**, **'animateColor'**, **'animateMotion'**, **'animateTransform'**, **'mpath'**, **'set'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Animation	**'animation'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Audio	**'audio'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Video	**'video'**
http://www.w3.org/Graphics/SVG/feature/1.2/#Font	**'font'**, **'font-face'**, **'font-face-src'**, **'font-face-uri'**, **'glyph'**, **'hkern'**, **'missing-glyph'**,
http://www.w3.org/Graphics/SVG/feature/1.2/#Extensibility	**'foreignObject'**
http://www.w3.org/Graphics/SVG/feature/1.2/#MediaAttribute	**'audio-level'**
http://www.w3.org/Graphics/SVG/feature/1.2/#TextFlow	**'tbreak'**, **'textArea'**, **'display-align'**, **'line-increment'**
http://www.w3.org/Graphics/SVG/feature/1.2/#TransformedVideo	The ability to perform any trans-formation (including scaling) on video content.
http://www.w3.org/Graphics/SVG/feature/1.2/#ComposedVideo	The ability to compose video con-tent with other content.
http://www.w3.org/Graphics/SVG/feature/1.2/#EditableTextAttribute	**'editable'**

K Element Table

This appendix is informative.

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.

M
: 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'**, **'buffered-rendering'**, **'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	Attributes	prop.	Possible Children
'a'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, target, transform, typeof, xlink:actuate, xlink:arcrole, xlink:href, xlink:role, xlink:show, xlink:title, xlink:type, xml:base, xml:id, xml:lang, xml:space	✓	The **'a'** element may contain any element that its parent may contain, except itself.
'animate'	about, accumulate, additive, attributeName, attributeType, begin, by, calcMode, class, content, datatype, dur, end, fill, from, id, keySplines, keyTimes, max, min, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, systemLanguage, to, typeof, values, 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
'animateColor'	about, accumulate, additive, attributeName, attributeType, begin, by, calcMode, class, content, datatype, dur, end, fill, from, id, keySplines, keyTimes, max, min, property, rel, repeatCount, repeatDur,	✗	desc, handler, metadata, switch, title

Elements	Attributes	prop.	Possible Children
	requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, systemLanguage, to, typeof, values, xlink:actuate, xlink:arcrole, xlink:href, xlink:role, xlink:show, xlink:title, xlink:type, xml:base, xml:id, xml:lang, xml:space		
'animateMotion'	about, accumulate, additive, begin, by, calcMode, class, content, datatype, dur, end, fill, from, id, keyPoints, keySplines, keyTimes, max, min, origin, path, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, rotate, systemLanguage, to, typeof, values, 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, mpath, switch, title
'animateTransform'	about, accumulate, additive, attributeName, attributeType, begin, by, calcMode, class, content, datatype, dur, end, fill, from, id, keySplines, keyTimes, max, min, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, systemLanguage, to, type, typeof, values, 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
'animation'	about, begin, class, content, datatype, 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, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, syncBehavior, syncMaster, syncTolerance, systemLanguage, transform, typeof, 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
'audio'	about, begin, class, content, datatype, dur, end, externalResourcesRequired, fill, id, max, min, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, syncBehavior, syncMaster, syncTolerance, systemLanguage, type, typeof, 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
'circle'	about, class, content, cx, cy, datatype, 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, property, r, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title

Elements	Attributes	prop.	Possible Children
'defs'	about, class, content, datatype, id, property, rel, resource, rev, role, typeof, 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
'desc'	about, class, content, datatype, id, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, typeof, xml:base, xml:id, xml:lang, xml:space	M	<text>
'discard'	about, begin, class, content, datatype, id, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, typeof, 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
'ellipse'	about, class, content, cx, cy, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, rx, ry, systemLanguage, transform, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title
'font'	about, class, content, datatype, externalResourcesRequired, horiz-adv-x, horiz-origin-x, id, property, rel, resource, rev, role, typeof, xml:base, xml:id, xml:lang, xml:space	✗	desc, font-face, glyph, hkern, metadata, missing-glyph, switch, title
'font-face'	about, accent-height, alphabetic, ascent, bbox, cap-height, class, content, datatype, descent, externalResourcesRequired, font-family, font-stretch, font-style, font-variant, font-weight, hanging, id, ideographic, mathematical, overline-position, overline-thickness, panose-1, property, rel, resource, rev, role, slope, stemh, stemv, strikethrough-position, strikethrough-thickness, typeof, underline-position, underline-thickness, unicode-range, units-per-em, widths, x-height, xml:base, xml:id, xml:lang, xml:space	✗	desc, font-face-src, metadata, switch, title
'font-face-src'	about, class, content, datatype, id, property, rel, resource, rev, role, typeof, xml:base, xml:id, xml:lang, xml:space	✗	desc, font-face-uri, metadata, switch, title
'font-face-uri'	about, class, content, datatype, externalResourcesRequired, id, property, rel, resource, rev, role, typeof, xlink:actuate, xlink:arcrole, xlink:href, xlink:role,	✗	desc, metadata, switch, title

Elements	Attributes	prop.	Possible Children
	xlink:show, xlink:title, xlink:type, xml:base, xml:id, xml:lang, xml:space		
'foreignObject'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, 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, svg, switch, title
'g'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, 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'	about, arabic-form, class, content, d, datatype, glyph-name, horiz-adv-x, id, lang, property, rel, resource, rev, role, typeof, unicode, xml:base, xml:id, xml:lang, xml:space	✗	desc, metadata, switch, title
'handler'	about, class, content, datatype, ev:event, externalResourcesRequired, id, property, rel, resource, rev, role, type, typeof, xlink:actuate, xlink:arcrole, xlink:href, xlink:role, xlink:show, xlink:title, xlink:type, xml:base, xml:id, xml:lang, xml:space	✗	<text>, desc, metadata, switch, title
'hkern'	about, class, content, datatype, g1, g2, id, k, property, rel, resource, rev, role, typeof, u1, u2, xml:base, xml:id, xml:lang, xml:space	✗	desc, metadata, switch, title
'image'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, type, typeof, 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
'line'	about, class, content, datatype, 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, property, rel, requiredExtensions,	✓	animate, animateColor, animateMotion, animateTransform, desc,

Elements	Attributes	prop.	Possible Children
	requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, x1, x2, xml:base, xml:id, xml:lang, xml:space, y1, y2		discard, handler, metadata, set, switch, title
'linearGradient'	about, class, content, datatype, gradientUnits, id, property, rel, resource, rev, role, typeof, x1, x2, xml:base, xml:id, xml:lang, xml:space, y1, y2	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, metadata, set, stop, switch, title
'listener'	about, class, content, datatype, defaultAction, event, handler, id, observer, phase, propagate, property, rel, resource, rev, role, target, typeof, xml:base, xml:id, xml:lang, xml:space	✗	
'metadata'	about, class, content, datatype, id, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, typeof, xml:base, xml:id, xml:lang, xml:space	M	<text>
'missing-glyph'	about, class, content, d, datatype, horiz-adv-x, id, property, rel, resource, rev, role, typeof, xml:base, xml:id, xml:lang, xml:space	✗	desc, metadata, switch, title
'mpath'	about, class, content, datatype, id, property, rel, resource, rev, role, typeof, 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
'path'	about, class, content, d, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title
'polygon'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title
'polyline'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title
'prefetch'	about, bandwidth, class, content, datatype, id, mediaCharacterEncoding, mediaContentEncodings, mediaSize, mediaTime, property, rel, resource, rev, role, typeof, xlink:actuate, xlink:arcrole, xlink:href, xlink:role,	✗	desc, metadata, switch, title

Elements	Attributes	prop.	Possible Children
	xlink:show, xlink:title, xlink:type, xml:base, xml:id, xml:lang, xml:space		
'radialGradient'	about, class, content, cx, cy, datatype, gradientUnits, id, property, r, rel, resource, rev, role, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, metadata, set, stop, switch, title
'rect'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, rx, ry, systemLanguage, transform, typeof, width, x, xml:base, xml:id, xml:lang, xml:space, y	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title
'script'	about, class, content, datatype, externalResourcesRequired, id, property, rel, resource, rev, role, type, typeof, xlink:actuate, xlink:arcrole, xlink:href, xlink:role, xlink:show, xlink:title, xlink:type, xml:base, xml:id, xml:lang, xml:space	✗	<text>, desc, metadata, switch, title
'set'	about, attributeName, attributeType, begin, class, content, datatype, dur, end, fill, id, max, min, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, systemLanguage, to, typeof, 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
'solidColor'	about, class, content, datatype, id, property, rel, resource, rev, role, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title
'stop'	about, class, content, datatype, id, offset, property, rel, resource, rev, role, typeof, xml:base, xml:id, xml:lang, xml:space	✓	animate, animateColor, animateMotion, animateTransform, desc, discard, metadata, set, switch, title
'svg'	about, baseProfile, class, content, contentScriptType, datatype, 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, property, rel, resource, rev, role, snapshotTime, syncBehaviorDefault, syncToleranceDefault, timelineBegin, typeof, 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

Elements	Attributes	prop.	Possible Children
'switch'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, xml:base, xml:id, xml:lang, xml:space	✓	The **'switch'** element may contain any element that its parent may contain.
'tbreak'	about, class, content, datatype, id, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, typeof, xml:base, xml:id, xml:lang, xml:space	✗	
'text'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, rotate, systemLanguage, transform, typeof, x, xml:base, xml:id, xml:lang, xml:space, y	✓	<text>, a, animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, title, tspan
'textArea'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, width, x, xml:base, xml:id, xml:lang, xml:space, y	✓	<text>, a, animate, animateColor, animateMotion, animateTransform, desc, discard, handler, metadata, set, switch, tbreak, title, tspan
'title'	about, class, content, datatype, id, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, typeof, xml:base, xml:id, xml:lang, xml:space	M	<text>
'tspan'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, typeof, xml:base, xml:id, xml:lang, xml:space	✓	The **'tspan'** element may contain any element that its parent may contain.
'use'	about, class, content, datatype, 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, property, rel, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, rev, role, systemLanguage, transform, typeof, 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'	about, begin, class, content, datatype, 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,	M	animate, animateColor, animateMotion, animateTransform, desc,

Elements	Attributes	prop.	Possible Children
	nav-right, nav-up, nav-up-left, nav-up-right, overlay, preserveAspectRatio, property, rel, repeatCount, repeatDur, requiredExtensions, requiredFeatures, requiredFonts, requiredFormats, resource, restart, rev, role, syncBehavior, syncMaster, syncTolerance, systemLanguage, transform, transformBehavior, type, typeof, 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		discard, handler, metadata, set, switch, title

L Attribute and Property Tables

Contents

L.1 Property table 368
L.2 Attribute table. 370

This appendix is informative.

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 if 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 is inherited by default.

✗
This attribute or property is not be inherited by default.

<text>
Indicates that character data can occur as the value of a given attribute.

L.1 Property table

Property	ani.	inh.	Value
'audio-level'	✓	✗	<number> | 'inherit'
'buffered-rendering'	✓	✗	'auto' | 'dynamic' | 'static' | 'inherit'
'color'	✓	✓	<color> | 'inherit'
'color-rendering'	✓	✓	'auto' | 'optimizeSpeed' | 'optimizeQuality' | 'inherit'
'direction'	✗	✓	'ltr' | 'rtl' | 'inherit'
'display'	✓	✗	'inline' | 'block' | 'list-item' | 'run-in' | 'compact' | 'marker' | 'table' | 'inline-table' | 'table-row-group' | 'table-header-group' | 'table-footer-group' | 'table-row' |

Property	ani.	inh.	Value
			'table-column-group' | 'table-column' | 'table-cell' | 'table-caption' | 'none' | 'inherit'
'display-align'	✓	✓	'auto' | 'before' | 'center' | 'after' | 'inherit'
'fill'	✓	✓	<paint> | 'inherit'
'fill-opacity'	✓	✓	<number> | 'inherit'
'fill-rule'	✓	✓	'nonzero' | 'evenodd' | 'inherit'
'font-family'	✓	✓	<font-family-value> | 'inherit'
'font-size'	✓	✓	<font-size-value> | 'inherit'
'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' | <number> | 'inherit'
'opacity'	✓	✗	<number> | 'inherit'
'pointer-events'	✓	✓	'visiblePainted' | 'visibleFill' | 'visibleStroke' | 'visible' | 'painted' | 'fill' | 'stroke' | 'all' | 'none' | 'inherit'
'shape-rendering'	✓	✓	'auto' | 'optimizeSpeed' | 'crispEdges' | 'geometricPrecision' | 'inherit'
'solid-color'	✓	✗	<color> | 'inherit'
'solid-opacity'	✓	✗	<number> | 'inherit'
'stop-color'	✓	✗	<color> | 'inherit'
'stop-opacity'	✓	✗	<number> | 'inherit'
'stroke'	✓	✓	<paint> | 'inherit'
'stroke-dasharray'	✓	✓	'none' | <list-of-lengths> | 'inherit'
'stroke-dashoffset'	✓	✓	<length> | 'inherit'
'stroke-linecap'	✓	✓	'butt' | 'round' | 'square' | 'inherit'
'stroke-linejoin'	✓	✓	'miter' | 'round' | 'bevel' | 'inherit'
'stroke-miterlimit'	✓	✓	<number> | 'inherit'
'stroke-opacity'	✓	✓	<number> | 'inherit'
'stroke-width'	✓	✓	<length> | 'inherit'

Property	ani.	inh.	Value
'text-align'	✓	✓	'start' | 'center' | 'end' | 'inherit'
'text-anchor'	✓	✓	'start' | 'middle' | 'end' | 'inherit'
'text-rendering'	✓	✓	'auto' | 'optimizeSpeed' | 'optimizeLegibility' | 'geometricPrecision' | 'inherit'
'unicode-bidi'	✗	✗	'normal' | 'embed' | 'bidi-override' | 'inherit'
'vector-effect'	✓	✗	'none' | 'non-scaling-stroke' | 'inherit'
'viewport-fill'	✓	✗	'none' | <color> | 'inherit'
'viewport-fill-opacity'	✓	✗	<number> | 'inherit'
'visibility'	✓	✓	'visible' | 'hidden' | 'inherit'

L.2 Attribute table

Attribute	ani.	inh.	Value	Elements
about	✓	✗	<list-of-strings>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
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	✗	✗	<QName>	animate, animateColor, animateTransform, set
attributeType	✗	✗	'XML' | 'CSS' | 'auto'	animate, animateColor, animateTransform, set
bandwidth	✗	✗	<number> | 'auto'	prefetch
baseProfile	✗	✗	'none' | 'tiny' | 'basic' | 'full'	svg
bbox	✗	✗	<text>	font-face

Attribute	ani.	inh.	Value	Elements
begin	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform, animation, audio, discard, set, video
by	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform
calcMode	✗	✗	'discrete' | 'linear' | 'paced' | 'spline'	animate, animateColor, animateMotion, animateTransform
cap-height	✗	✗	<number>	font-face
class	✓	✗	<XML-NMTOKENS>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
content	✓	✗	<string>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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	✗	✗	<content-type>	svg
cx	✓	✗	<coordinate>	circle, ellipse, radialGradient
cy	✓	✗	<coordinate>	circle, ellipse, radialGradient
d	✓	✗	<path-data>	glyph, missing-glyph, path
datatype	✓	✗	<string>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
defaultAction	✗	✗	'perform' | 'cancel'	listener
descent	✗	✗	<number>	font-face
dur	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform, animation, audio, set, video

Attribute	ani.	inh.	Value	Elements
editable	✓	✗	'none' | 'simple'	text, textArea
end	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform, animation, audio, set, video
ev:event	✗	✗	<XML-NMTOKEN>	handler
event	✗	✗	<XML-NMTOKEN>	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

Attribute	ani.	inh.	Value	Elements
id	✗	✗	<ID> | <NCName>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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	✗	✗	<list-of-language-ids>	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
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

Attribute	ani.	inh.	Value	Elements
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	✗	✗	<IDREF>	listener
offset	✓	✗	<number>	stop
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-data>	polygon, polyline
preserveAspectRatio	✓	✗	<text>	animation, image, svg, video
propagate	✗	✗	'continue' | 'stop'	listener
property	✓	✗	<list-of-strings>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, font-face-src, font-face-uri, foreignObject, g, glyph, handler, hkern, image, line, linearGradient, listener, metadata, missing-glyph, mpath, path, polygon, polyline,

Attribute	ani.	inh.	Value	Elements
				prefetch, radialGradient, rect, script, set, solidColor, stop, svg, switch, tbreak, text, textArea, title, tspan, use, video
r	✓	✗	<length>	circle, radialGradient
rel	✓	✗	<list-of-strings>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
repeatCount	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform, animation, audio, set, video
repeatDur	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform, animation, audio, set, video
requiredExtensions	✗	✗	<list-of-IRIs>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, desc, discard, ellipse, foreignObject, g, image, line, metadata, path, polygon, polyline, rect, set, switch, tbreak, text, textArea, title, tspan, use, video
requiredFeatures	✗	✗	<list-of-IRIs>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, desc, discard, ellipse, foreignObject, g, image, line, metadata, path, polygon, polyline, rect, set, switch, tbreak, text, textArea, title, tspan, use, video
requiredFonts	✗	✗	<list-of-family-names>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, desc, discard, ellipse, foreignObject, g, image, line, metadata, path, polygon, polyline, rect, set, switch, tbreak, text, textArea, title, tspan, use, video
requiredFormats	✗	✗	<list-of-content-types>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, desc, discard, ellipse, foreignObject, g, image, line, metadata, path, polygon, polyline, rect, set, switch, tbreak, text, textArea, title, tspan, use, video
resource	✓	✗	<string>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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,

Attribute	ani.	inh.	Value	Elements
				solidColor, stop, svg, switch, tbreak, text, textArea, title, tspan, use, video
restart	✗	✗	'always' | 'never' | 'whenNotActive'	animate, animateColor, animateMotion, animateTransform, animation, audio, set, video
rev	✓	✗	<list-of-strings>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
role	✓	✗	<list-of-strings>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
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

Attribute	ani.	inh.	Value	Elements
syncTolerance	✗	✗	<Clock-value> | 'default'	animation, audio, video
syncToleranceDefault	✗	✗	<Clock-value> | 'inherit'	svg
systemLanguage	✗	✗	<list-of-language-ids>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, desc, discard, ellipse, foreignObject, g, image, line, metadata, path, polygon, polyline, rect, set, switch, tbreak, text, textArea, title, tspan, use, video
target	✓	✗	'_replace' | '_self' | '_parent' | '_top' | '_blank' | <XML-Name>	a
target	✗	✗	<IDREF>	listener
timelineBegin	✗	✗	'onLoad' | 'onStart'	svg
to	✗	✗	<text>	animate, animateColor, animateMotion, animateTransform, set
transform	✓	✗	<transform> | 'none'	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	✗	✗	<content-type>	handler, script
type	✓	✗	<content-type>	audio, image, video
type	✗	✗	'translate' | 'scale' | 'rotate' | 'skewX' | 'skewY'	animateTransform
typeof	✓	✗	<list-of-strings>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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
u1	✗	✗	<text>	hkern
u2	✗	✗	<text>	hkern
underline-position	✗	✗	<number>	font-face

Attribute	ani.	inh.	Value	Elements
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	✗	✗	<IRI>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, discard, font-face-uri, foreignObject, handler, image, mpath, prefetch, script, set, use, video
xlink:show	✗	✗	'other'	animate, animateColor, animateMotion, animateTransform, discard, font-face-uri, handler, mpath, prefetch, script, set

Attribute	ani.	inh.	Value	Elements
xlink:show	✗	✗	'embed'	animation, audio, foreignObject, image, use, video
xlink:show	✗	✗	'new' | 'replace'	a
xlink:title	✗	✗	<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-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	✗	✗	<NCName>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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	✗	✗	<language-id>	a, animate, animateColor, animateMotion, animateTransform, animation, audio, circle, defs, desc, discard, ellipse, font, font-face, 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-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

Attribute	ani.	inh.	Value	Elements
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 381
M.2 Registration of media type image/svg+xml 381

This appendix is normative.

M.1 Introduction

This appendix registers a new MIME media type, "image/svg+xml" in conformance with BCP 13 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:

Chris Lilley, Doug Schepers (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

This appendix is normative.

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 driver schema is available, as is a complete schema directory.

O ECMAScript Language Binding

Contents

O.1 Module dom 384
O.2 Module views 386
O.3 Module events 386
O.4 Module smil 388
O.5 Module svg 388
O.6 Sequences 393

This appendix is normative.

O.1 Module dom

Prototype Object DOMException

The DOMException class has the following constants:

DOMException.INDEX_SIZE_ERR
This constant is of type **Number** and its value is **1.**

DOMException.DOMSTRING_SIZE_ERR
This constant is of type **Number** and its value is **2.**

DOMException.HIERARCHY_REQUEST_ERR
This constant is of type **Number** and its value is **3.**

DOMException.WRONG_DOCUMENT_ERR
This constant is of type **Number** and its value is **4.**

DOMException.INVALID_CHARACTER_ERR
This constant is of type **Number** and its value is **5.**

DOMException.NO_DATA_ALLOWED_ERR
This constant is of type **Number** and its value is **6.**

DOMException.NO_MODIFICATION_ALLOWED_ERR
This constant is of type **Number** and its value is **7.**

DOMException.NOT_FOUND_ERR
This constant is of type **Number** and its value is **8.**

DOMException.NOT_SUPPORTED_ERR
This constant is of type **Number** and its value is **9.**

DOMException.INUSE_ATTRIBUTE_ERR
This constant is of type **Number** and its value is **10.**

DOMException.INVALID_STATE_ERR
This constant is of type **Number** and its value is **11.**

DOMException.SYNTAX_ERR
This constant is of type **Number** and its value is **12.**

DOMException.INVALID_MODIFICATION_ERR
This constant is of type **Number** and its value is **13.**

DOMException.NAMESPACE_ERR
This constant is of type **Number** and its value is **14.**

DOMException.INVALID_ACCESS_ERR
This constant is of type **Number** and its value is **15.**

DOMException.VALIDATION_ERR
This constant is of type **Number** and its value is **16.**

DOMException.TYPE_MISMATCH_ERR
This constant is of type **Number** and its value is **17.**

Object DOMException

The DOMException object has the following properties:

code
This property is of type **Number**.

Object Node

The Node object has the following properties:

namespaceURI
This property is of type **String**.

localName
This property is of type **String**.

parentNode
This property is of type Node.

ownerDocument
This property is of type Document.

textContent
This property is of type **String**.

The Node object has the following methods:

appendChild(newChild)
This method returns a(n) Node. The newChild parameter is of type Node.

insertBefore(newChild, refChild)
This method returns a(n) Node. The newChild parameter is of type Node. The refChild parameter is of type Node.

removeChild(oldChild)
This method returns a(n) Node. The oldChild parameter is of type Node.

cloneNode(deep)
This method returns a(n) Node. The deep parameter is of type **Boolean**.

Object Element

Element has all the properties and methods of Node, ElementTraversal as well as the properties and methods defined below.

The Element object has the following methods:

getAttributeNS(namespaceURI, localName)
This method returns a(n) **String**. The namespaceURI parameter is of type **String**. The localName parameter is of type **String**.

setAttributeNS(namespaceURI, qualifiedName, value)
This method has no return value. The namespaceURI parameter is of type **String**. The qualifiedName parameter is of type **String**. The value parameter is of type **String**.

getAttribute(name)
This method returns a(n) **String**. The name parameter is of type **String**.

setAttribute(name, value)
This method has no return value. The name parameter is of type **String**. The value parameter is of type **String**.

Object Document

Document has all the properties and methods of Node as well as the properties and methods defined below.

The Document object has the following properties:

documentElement
This property is of type Element.

The Document object has the following methods:

createElementNS(namespaceURI, qualifiedName)
This method returns a(n) Element. The namespaceURI parameter is of type **String**. The qualifiedName parameter is of type **String**.

getElementById(elementId)
This method returns a(n) Element. The elementId parameter is of type **String**.

Object ElementTraversal

The ElementTraversal object has the following properties:

firstElementChild
This property is of type Element.

lastElementChild
This property is of type Element.

nextElementSibling
This property is of type Element.

previousElementSibling
This property is of type Element.
childElementCount
This property is of type **Number**.

Object Location
The Location object has the following methods:
assign(iri)
This method has no return value. The iri parameter is of type **String**.
reload()
This method has no return value.

Object Window
The Window object has the following properties:
parent
This property is of type Window.
location
This property is of type Location.

O.2 Module `views`

Object AbstractView
The AbstractView object has the following properties:
document
This property is of type DocumentView.

Object DocumentView
The DocumentView object has the following properties:
defaultView
This property is of type AbstractView.

O.3 Module `events`

Object EventTarget
The EventTarget object has the following methods:
addEventListener(type, listener, useCapture)
This method has no return value. The type parameter is of type **String**. The listener parameter is of type EventListener. The useCapture parameter is of type **Boolean**.
removeEventListener(type, listener, useCapture)
This method has no return value. The type parameter is of type **String**. The listener parameter is of type EventListener. The useCapture parameter is of type **Boolean**.

Function EventListener
In ECMAScript, an EventListener can be specified in one of two ways. The first is an ECMAScript Function object, where the function itself is considered to be the implementation of the handleEvent operation. The second is an ECMAScript object with a property handleEvent, which provides the implementation of the handleEvent operation. If an ECMAScript Function object also has a handleEvent property, then that handleEvent property and not the Function itself is considered to be the implementation of the handleEvent operation.
The EventListener object has the following methods:
handleEvent(evt)
This method has no return value. The evt parameter is of type Event.

Object Event
The Event object has the following properties:
target
This property is of type EventTarget.
currentTarget
This property is of type EventTarget.
type
This property is of type **String**.

cancelable
This property is of type **Boolean**.
defaultPrevented
This property is of type **Boolean**.

The Event object has the following methods:
stopPropagation()
This method has no return value.
preventDefault()
This method has no return value.

Object MouseEvent
MouseEvent has all the properties and methods of UIEvent as well as the properties and methods defined below.
The MouseEvent object has the following properties:
screenX
This property is of type **Number**.
screenY
This property is of type **Number**.
clientX
This property is of type **Number**.
clientY
This property is of type **Number**.
button
This property is of type **Number**.

Object MouseWheelEvent
MouseWheelEvent has all the properties and methods of MouseEvent as well as the properties and methods defined below.
The MouseWheelEvent object has the following properties:
wheelDelta
This property is of type **Number**.

Object TextEvent
TextEvent has all the properties and methods of UIEvent as well as the properties and methods defined below.
The TextEvent object has the following properties:
data
This property is of type **String**.

Object KeyboardEvent
KeyboardEvent has all the properties and methods of UIEvent as well as the properties and methods defined below.
The KeyboardEvent object has the following properties:
keyIdentifier
This property is of type **String**.

Object UIEvent
UIEvent has all the properties and methods of Event as well as the properties and methods defined below.
The UIEvent object has the following properties:
detail
This property is of type **Number**.

Object ProgressEvent
ProgressEvent has all the properties and methods of Event as well as the properties and methods defined below.
The ProgressEvent object has the following properties:
lengthComputable
This property is of type **Boolean**.
loaded
This property is of type **Number**.
total
This property is of type **Number**.

O.4 Module `smil`

Object ElementTimeControl

The ElementTimeControl object has the following methods:

beginElementAt(offset)

This method has no return value. The offset parameter is of type **Number**.

beginElement()

This method has no return value.

endElementAt(offset)

This method has no return value. The offset parameter is of type **Number**.

endElement()

This method has no return value.

Object TimeEvent

TimeEvent has all the properties and methods of Event as well as the properties and methods defined below.

The TimeEvent object has the following properties:

detail

This property is of type **Number**.

O.5 Module `svg`

Prototype Object SVGException

The SVGException class has the following constants:

SVGException.SVG_WRONG_TYPE_ERR

This constant is of type **Number** and its value is **0.**

SVGException.SVG_INVALID_VALUE_ERR

This constant is of type **Number** and its value is **1.**

SVGException.SVG_MATRIX_NOT_INVERTABLE

This constant is of type **Number** and its value is **2.**

Object SVGException

The SVGException object has the following properties:

code

This property is of type **Number**.

Object SVGDocument

SVGDocument has all the properties and methods of Document, EventTarget as well as the properties and methods defined below.

Object SVGUseElement

SVGUseElement has all the properties and methods of SVGLocatableElement as well as the properties and methods defined below.

Object SVGElementInstance

SVGElementInstance has all the properties and methods of EventTarget as well as the properties and methods defined below.

The SVGElementInstance object has the following properties:

correspondingElement

This property is of type SVGElement.

correspondingUseElement

This property is of type SVGUseElement.

Prototype Object SVGSVGElement

The SVGSVGElement class has the following constants:

SVGSVGElement.NAV_AUTO

This constant is of type **Number** and its value is **1.**

SVGSVGElement.NAV_NEXT

This constant is of type **Number** and its value is **2.**

SVGSVGElement.NAV_PREV

This constant is of type **Number** and its value is **3.**

SVGSVGElement.NAV_UP

This constant is of type **Number** and its value is **4.**

SVGSVGElement.NAV_UP_RIGHT
This constant is of type **Number** and its value is **5.**
SVGSVGElement.NAV_RIGHT
This constant is of type **Number** and its value is **6.**
SVGSVGElement.NAV_DOWN_RIGHT
This constant is of type **Number** and its value is **7.**
SVGSVGElement.NAV_DOWN
This constant is of type **Number** and its value is **8.**
SVGSVGElement.NAV_DOWN_LEFT
This constant is of type **Number** and its value is **9.**
SVGSVGElement.NAV_LEFT
This constant is of type **Number** and its value is **10.**
SVGSVGElement.NAV_UP_LEFT
This constant is of type **Number** and its value is **11.**

Object SVGSVGElement

SVGSVGElement has all the properties and methods of SVGLocatableElement, SVGTimedElement as well as the properties and methods defined below.

The SVGSVGElement object has the following properties:

currentScale
This property is of type **Number**.
currentRotate
This property is of type **Number**.
currentTranslate
This property is of type SVGPoint.
viewport
This property is of type SVGRect.

The SVGSVGElement object has the following methods:

getCurrentTime()
This method returns a(n) **Number**.
setCurrentTime(seconds)
This method has no return value. The seconds parameter is of type **Number**.
createSVGMatrixComponents(a, b, c, d, e, f)
This method returns a(n) SVGMatrix. The a parameter is of type **Number**. The b parameter is of type **Number**. The c parameter is of type **Number**. The d parameter is of type **Number**. The e parameter is of type **Number**. The f parameter is of type **Number**.
createSVGRect()
This method returns a(n) SVGRect.
createSVGPoint()
This method returns a(n) SVGPoint.
createSVGPath()
This method returns a(n) SVGPath.
createSVGRGBColor(red, green, blue)
This method returns a(n) SVGRGBColor. The red parameter is of type **Number**. The green parameter is of type **Number**. The blue parameter is of type **Number**.
moveFocus(motionType)
This method has no return value. The motionType parameter is of type **Number**.
setFocus(theObject)
This method has no return value. The theObject parameter is of type EventTarget.
getCurrentFocusedObject()
This method returns a(n) EventTarget.

Object SVGRGBColor

The SVGRGBColor object has the following properties:

red
This property is of type **Number**.
green
This property is of type **Number**.

blue
This property is of type **Number**.

Object SVGRect
The SVGRect object has the following properties:
x
This property is of type **Number**.
y
This property is of type **Number**.
width
This property is of type **Number**.
height
This property is of type **Number**.

Object SVGPoint
The SVGPoint object has the following properties:
x
This property is of type **Number**.
y
This property is of type **Number**.
The SVGPoint object has the following methods:
matrixTransform(matrix)
This method returns a(n) SVGPoint. The matrix parameter is of type SVGMatrix.

Prototype Object SVGPath
The SVGPath class has the following constants:
SVGPath.MOVE_TO
This constant is of type **Number** and its value is **77.**
SVGPath.LINE_TO
This constant is of type **Number** and its value is **76.**
SVGPath.CURVE_TO
This constant is of type **Number** and its value is **67.**
SVGPath.QUAD_TO
This constant is of type **Number** and its value is **81.**
SVGPath.CLOSE
This constant is of type **Number** and its value is **90.**

Object SVGPath
The SVGPath object has the following properties:
numberOfSegments
This property is of type **Number**.
The SVGPath object has the following methods:
getSegment(cmdIndex)
This method returns a(n) **Number**. The cmdIndex parameter is of type **Number**.
getSegmentParam(cmdIndex, paramIndex)
This method returns a(n) **Number**. The cmdIndex parameter is of type **Number**. The paramIndex parameter is of type **Number**.
moveTo(x, y)
This method has no return value. The x parameter is of type **Number**. The y parameter is of type **Number**.
lineTo(x, y)
This method has no return value. The x parameter is of type **Number**. The y parameter is of type **Number**.
quadTo(x1, y1, x2, y2)
This method has no return value. The x1 parameter is of type **Number**. The y1 parameter is of type **Number**. The x2 parameter is of type **Number**. The y2 parameter is of type **Number**.
curveTo(x1, y1, x2, y2, x3, y3)
This method has no return value. The x1 parameter is of type **Number**. The y1 parameter is of type **Number**. The x2 parameter is of type **Number**. The y2 parameter is of type **Number**. The x3 parameter is of type **Number**. The y3 parameter is of type **Number**.

close()
This method has no return value.

Object SVGMatrix

The SVGMatrix object has the following methods:

getComponent(index)
This method returns a(n) **Number**. The index parameter is of type **Number**.

mMultiply(secondMatrix)
This method returns a(n) SVGMatrix. The secondMatrix parameter is of type SVGMatrix.

inverse()
This method returns a(n) SVGMatrix.

mTranslate(x, y)
This method returns a(n) SVGMatrix. The x parameter is of type **Number**. The y parameter is of type **Number**.

mScale(scaleFactor)
This method returns a(n) SVGMatrix. The scaleFactor parameter is of type **Number**.

mRotate(angle)
This method returns a(n) SVGMatrix. The angle parameter is of type **Number**.

Object SVGLocatable

The SVGLocatable object has the following methods:

getBBox()
This method returns a(n) SVGRect.

getScreenCTM()
This method returns a(n) SVGMatrix.

getScreenBBox()
This method returns a(n) SVGRect.

Object SVGLocatableElement

SVGLocatableElement has all the properties and methods of SVGElement, SVGLocatable as well as the properties and methods defined below.

Object TraitAccess

The TraitAccess object has the following methods:

getTrait(name)
This method returns a(n) **String**. The name parameter is of type **String**.

getTraitNS(namespaceURI, name)
This method returns a(n) **String**. The namespaceURI parameter is of type **String**. The name parameter is of type **String**.

getFloatTrait(name)
This method returns a(n) **Number**. The name parameter is of type **String**.

getFloatListTrait(name)
This method returns a(n) **Object** (but see Sequences below for details). The name parameter is of type **String**.

getMatrixTrait(name)
This method returns a(n) SVGMatrix. The name parameter is of type **String**.

getRectTrait(name)
This method returns a(n) SVGRect. The name parameter is of type **String**.

getPathTrait(name)
This method returns a(n) SVGPath. The name parameter is of type **String**.

getRGBColorTrait(name)
This method returns a(n) SVGRGBColor. The name parameter is of type **String**.

getPresentationTrait(name)
This method returns a(n) **String**. The name parameter is of type **String**.

getPresentationTraitNS(namespaceURI, name)
This method returns a(n) **String**. The namespaceURI parameter is of type **String**. The name parameter is of type **String**.

getFloatPresentationTrait(name)
This method returns a(n) **Number**. The name parameter is of type **String**.

getFloatListPresentationTrait(name)
: This method returns a(n) **Object** (but see Sequences below for details). The name parameter is of type **String**.

getMatrixPresentationTrait(name)
: This method returns a(n) SVGMatrix. The name parameter is of type **String**.

getRectPresentationTrait(name)
: This method returns a(n) SVGRect. The name parameter is of type **String**.

getPathPresentationTrait(name)
: This method returns a(n) SVGPath. The name parameter is of type **String**.

getRGBColorPresentationTrait(name)
: This method returns a(n) SVGRGBColor. The name parameter is of type **String**.

setTrait(name, value)
: This method has no return value. The name parameter is of type **String**. The value parameter is of type **String**.

setTraitNS(namespaceURI, name, value)
: This method has no return value. The namespaceURI parameter is of type **String**. The name parameter is of type **String**. The value parameter is of type **String**.

setFloatTrait(name, value)
: This method has no return value. The name parameter is of type **String**. The value parameter is of type **Number**.

setFloatListTrait(name, value)
: This method has no return value. The name parameter is of type **String**. The value parameter is of type **Object** (but see Sequences below for details).

setMatrixTrait(name, matrix)
: This method has no return value. The name parameter is of type **String**. The matrix parameter is of type SVGMatrix.

setRectTrait(name, rect)
: This method has no return value. The name parameter is of type **String**. The rect parameter is of type SVGRect.

setPathTrait(name, path)
: This method has no return value. The name parameter is of type **String**. The path parameter is of type SVGPath.

setRGBColorTrait(name, color)
: This method has no return value. The name parameter is of type **String**. The color parameter is of type SVGRGBColor.

Object SVGElement

SVGElement has all the properties and methods of Element, EventTarget, TraitAccess as well as the properties and methods defined below.

The SVGElement object has the following properties:

id
: This property is of type **String**.

Object SVGTimedElement

SVGTimedElement has all the properties and methods of SVGElement, smil::ElementTimeControl as well as the properties and methods defined below.

The SVGTimedElement object has the following properties:

isPaused
: This property is of type **Boolean**.

The SVGTimedElement object has the following methods:

pauseElement()
: This method has no return value.

resumeElement()
: This method has no return value.

Object SVGAnimationElement

SVGAnimationElement has all the properties and methods of SVGTimedElement as well as the properties and methods defined below.

Object SVGVisualMediaElement

SVGVisualMediaElement has all the properties and methods of SVGLocatableElement, SVGTimedElement as well as the properties and methods defined below.

Object SVGTimer

SVGTimer has all the properties and methods of events::EventTarget as well as the properties and methods defined below.

The SVGTimer object has the following properties:

delay

This property is of type **Number**.

repeatInterval

This property is of type **Number**.

running

This property is of type **Boolean**.

The SVGTimer object has the following methods:

start()

This method has no return value.

stop()

This method has no return value.

Object SVGGlobal

The SVGGlobal object has the following methods:

createTimer(initialInterval, repeatInterval)

This method returns a(n) SVGTimer. The initialInterval parameter is of type **Number**. The repeatInterval parameter is of type **Number**.

getURL(iri, callback)

This method has no return value. The iri parameter is of type **String**. The callback parameter is of type AsyncStatusCallback.

postURL(iri, data, callback, type, encoding)

This method has no return value. The iri parameter is of type **String**. The data parameter is of type **String**. The callback parameter is of type AsyncStatusCallback. The type parameter is of type **String**. The encoding parameter is of type **String**.

parseXML(data, contextDoc)

This method returns a(n) Node. The data parameter is of type **String**. The contextDoc parameter is of type Document.

Object AsyncStatusCallback

The AsyncStatusCallback object has the following methods:

operationComplete(status)

This method has no return value. The status parameter is of type AsyncURLStatus.

Object AsyncURLStatus

The AsyncURLStatus object has the following properties:

success

This property is of type **Boolean**.

contentType

This property is of type **String**.

content

This property is of type **String**.

O.6 Sequences

In the ECMAScript language binding, a value of type **sequence<float>** is represented by an ECMAScript object, hereafter called the *sequence object*.

The sequence object must have a length property whose value is a non-negative integer **Number**, which gives the length of the sequence.

For each index in the range [0, length), the sequence object must have a property (an *index property*) whose name is the index and whose value gives the element of the sequence at that index. When returned from a host object (that is, when an operation whose return type is **sequence<float>** is called, or when an attribute whose type is

sequence<float> is retrieved), then the element values returned from these properties must be an ECMAScript **Number**.

When passed to a host object (that is, when passed as an operation argument whose type is **sequence<float>**, or when assigned to an attribute whose type is **sequence<float>** is assigned to), then the value of each index property gives the element value, which is the result of passing the value of the corresponding index property to the ToNumber operator (defined in section 9.3 of the *ECMAScript Language Specification* [ECMA-262]).

P Java Language Binding

Contents

P.1 Package org.w3c.dom 395
P.2 Package org.w3c.dom.views 397
P.3 Package org.w3c.dom.events 397
P.4 Package org.w3c.dom.smil. 398
P.5 Package org.w3c.dom.svg 399

This appendix is normative.

Constants in this binding are always `public static final`. All fields and methods are `public`.

P.1 Package `org.w3c.dom`

Class constants for org.w3c.dom.DOMException

The org.w3c.dom.DOMException class holds the following constants:

INDEX_SIZE_ERR
This constant is a `short` and its value is **1.**

DOMSTRING_SIZE_ERR
This constant is a `short` and its value is **2.**

HIERARCHY_REQUEST_ERR
This constant is a `short` and its value is **3.**

WRONG_DOCUMENT_ERR
This constant is a `short` and its value is **4.**

INVALID_CHARACTER_ERR
This constant is a `short` and its value is **5.**

NO_DATA_ALLOWED_ERR
This constant is a `short` and its value is **6.**

NO_MODIFICATION_ALLOWED_ERR
This constant is a `short` and its value is **7.**

NOT_FOUND_ERR
This constant is a `short` and its value is **8.**

NOT_SUPPORTED_ERR
This constant is a `short` and its value is **9.**

INUSE_ATTRIBUTE_ERR
This constant is a `short` and its value is **10.**

INVALID_STATE_ERR
This constant is a `short` and its value is **11.**

SYNTAX_ERR
This constant is a `short` and its value is **12.**

INVALID_MODIFICATION_ERR
This constant is a `short` and its value is **13.**

NAMESPACE_ERR
This constant is a `short` and its value is **14.**

INVALID_ACCESS_ERR
This constant is a `short` and its value is **15.**

VALIDATION_ERR
This constant is a `short` and its value is **16.**

TYPE_MISMATCH_ERR
This constant is a `short` and its value is **17.**

Class org.w3c.dom.DOMException

org.w3c.dom.DOMException extends java.lang.RuntimeException.

The org.w3c.dom.DOMException class has the following methods:

getCode()
This method returns a **short.**

setCode(short code)
This method has no return value.

Interface org.w3c.dom.Node

The org.w3c.dom.Node interface has the following methods:

getNamespaceURI()
This method returns a **String.**

getLocalName()
This method returns a **String.**

getParentNode()
This method returns a **Node.**

getOwnerDocument()
This method returns a **Document.**

getTextContent()
This method returns a **String.**

setTextContent(String textContent)
This method has no return value.

appendChild(Node newChild)
This method returns a **Node.**
This method throws DOMException exceptions.

insertBefore(Node newChild, Node refChild)
This method returns a **Node.**
This method throws DOMException exceptions.

removeChild(Node oldChild)
This method returns a **Node.**
This method throws DOMException exceptions.

cloneNode(boolean deep)
This method returns a **Node.**

Interface org.w3c.dom.Element

org.w3c.dom.Element extends Node, ElementTraversal.

The org.w3c.dom.Element interface has the following methods:

getAttributeNS(String namespaceURI, String localName)
This method returns a **String.**
This method throws DOMException exceptions.

setAttributeNS(String namespaceURI, String qualifiedName, String value)
This method has no return value.
This method throws DOMException exceptions.

getAttribute(String name)
This method returns a **String.**

setAttribute(String name, String value)
This method has no return value.
This method throws DOMException exceptions.

Interface org.w3c.dom.Document

org.w3c.dom.Document extends Node.

The org.w3c.dom.Document interface has the following methods:

createElementNS(String namespaceURI, String qualifiedName)
This method returns a **Element.**
This method throws DOMException exceptions.

getDocumentElement()
This method returns a **Element.**

getElementById(String elementId)
This method returns a **Element.**

Interface org.w3c.dom.ElementTraversal

The org.w3c.dom.ElementTraversal interface has the following methods:

getFirstElementChild()
This method returns a **Element.**

getLastElementChild()
This method returns a **Element.**
getNextElementSibling()
This method returns a **Element.**
getPreviousElementSibling()
This method returns a **Element.**
getChildElementCount()
This method returns a **long.**

Interface org.w3c.dom.Location
The org.w3c.dom.Location interface has the following methods:
assign(String iri)
This method has no return value.
reload()
This method has no return value.

Interface org.w3c.dom.Window
The org.w3c.dom.Window interface has the following methods:
getParent()
This method returns a **Window.**
getLocation()
This method returns a **Location.**

P.2 Package `org.w3c.dom.views`

Interface org.w3c.dom.views.AbstractView
The org.w3c.dom.views.AbstractView interface has the following methods:
getDocument()
This method returns a **DocumentView.**

Interface org.w3c.dom.views.DocumentView
The org.w3c.dom.views.DocumentView interface has the following methods:
getDefaultView()
This method returns a **AbstractView.**

P.3 Package `org.w3c.dom.events`

Interface org.w3c.dom.events.EventTarget
The org.w3c.dom.events.EventTarget interface has the following methods:
addEventListener(String type, EventListener listener, boolean useCapture)
This method has no return value.
removeEventListener(String type, EventListener listener, boolean useCapture)
This method has no return value.

Interface org.w3c.dom.events.EventListener
The org.w3c.dom.events.EventListener interface has the following methods:
handleEvent(Event evt)
This method has no return value.

Interface org.w3c.dom.events.Event
The org.w3c.dom.events.Event interface has the following methods:
getTarget()
This method returns a **EventTarget.**
getCurrentTarget()
This method returns a **EventTarget.**
getType()
This method returns a **String.**
getCancelable()
This method returns a **boolean.**
getDefaultPrevented()
This method returns a **boolean.**

stopPropagation()
This method has no return value.
preventDefault()
This method has no return value.

Interface org.w3c.dom.events.MouseEvent
org.w3c.dom.events.MouseEvent extends UIEvent.
The org.w3c.dom.events.MouseEvent interface has the following methods:
getScreenX()
This method returns a **long.**
getScreenY()
This method returns a **long.**
getClientX()
This method returns a **long.**
getClientY()
This method returns a **long.**
getButton()
This method returns a **short.**

Interface org.w3c.dom.events.MouseWheelEvent
org.w3c.dom.events.MouseWheelEvent extends MouseEvent.
The org.w3c.dom.events.MouseWheelEvent interface has the following methods:
getWheelDelta()
This method returns a **long.**

Interface org.w3c.dom.events.TextEvent
org.w3c.dom.events.TextEvent extends UIEvent.
The org.w3c.dom.events.TextEvent interface has the following methods:
getData()
This method returns a **String.**

Interface org.w3c.dom.events.KeyboardEvent
org.w3c.dom.events.KeyboardEvent extends UIEvent.
The org.w3c.dom.events.KeyboardEvent interface has the following methods:
getKeyIdentifier()
This method returns a **String.**

Interface org.w3c.dom.events.UIEvent
org.w3c.dom.events.UIEvent extends Event.
The org.w3c.dom.events.UIEvent interface has the following methods:
getDetail()
This method returns a **long.**

Interface org.w3c.dom.events.ProgressEvent
org.w3c.dom.events.ProgressEvent extends Event.
The org.w3c.dom.events.ProgressEvent interface has the following methods:
getLengthComputable()
This method returns a **boolean.**
getLoaded()
This method returns a **long.**
getTotal()
This method returns a **long.**

P.4 Package `org.w3c.dom.smil`

Interface org.w3c.dom.smil.ElementTimeControl
The org.w3c.dom.smil.ElementTimeControl interface has the following methods:
beginElementAt(float offset)
This method has no return value.
beginElement()
This method has no return value.

endElementAt(float offset)

This method has no return value.

endElement()

This method has no return value.

Interface org.w3c.dom.smil.TimeEvent

org.w3c.dom.smil.TimeEvent extends Event.

The org.w3c.dom.smil.TimeEvent interface has the following methods:

getDetail()

This method returns a **long.**

P.5 Package `org.w3c.dom.svg`

Class constants for org.w3c.dom.svg.SVGException

The org.w3c.dom.svg.SVGException class holds the following constants:

SVG_WRONG_TYPE_ERR

This constant is a `short` and its value is **0.**

SVG_INVALID_VALUE_ERR

This constant is a `short` and its value is **1.**

SVG_MATRIX_NOT_INVERTABLE

This constant is a `short` and its value is **2.**

Class org.w3c.dom.svg.SVGException

org.w3c.dom.svg.SVGException extends java.lang.RuntimeException.

The org.w3c.dom.svg.SVGException class has the following methods:

getCode()

This method returns a **short.**

setCode(short code)

This method has no return value.

Interface org.w3c.dom.svg.SVGDocument

org.w3c.dom.svg.SVGDocument extends Document, EventTarget.

Interface org.w3c.dom.svg.SVGUseElement

org.w3c.dom.svg.SVGUseElement extends SVGLocatableElement.

Interface org.w3c.dom.svg.SVGElementInstance

org.w3c.dom.svg.SVGElementInstance extends EventTarget.

The org.w3c.dom.svg.SVGElementInstance interface has the following methods:

getCorrespondingElement()

This method returns a **SVGElement.**

getCorrespondingUseElement()

This method returns a **SVGUseElement.**

Interface constants org.w3c.dom.svg.SVGSVGElement

The org.w3c.dom.svg.SVGSVGElement interface has the following constants:

NAV_AUTO

This constant is a `short` and its value is **1.**

NAV_NEXT

This constant is a `short` and its value is **2.**

NAV_PREV

This constant is a `short` and its value is **3.**

NAV_UP

This constant is a `short` and its value is **4.**

NAV_UP_RIGHT

This constant is a `short` and its value is **5.**

NAV_RIGHT

This constant is a `short` and its value is **6.**

NAV_DOWN_RIGHT

This constant is a `short` and its value is **7.**

NAV_DOWN

This constant is a `short` and its value is **8.**

NAV_DOWN_LEFT
This constant is a `short` and its value is **9.**
NAV_LEFT
This constant is a `short` and its value is **10.**
NAV_UP_LEFT
This constant is a `short` and its value is **11.**

Interface org.w3c.dom.svg.SVGSVGElement
org.w3c.dom.svg.SVGSVGElement extends SVGLocatableElement, SVGTimedElement.
The org.w3c.dom.svg.SVGSVGElement interface has the following methods:

getCurrentScale()
This method returns a **float.**
setCurrentScale(float currentScale)
This method has no return value.
getCurrentRotate()
This method returns a **float.**
setCurrentRotate(float currentRotate)
This method has no return value.
getCurrentTranslate()
This method returns a **SVGPoint.**
getViewport()
This method returns a **SVGRect.**
getCurrentTime()
This method returns a **float.**
setCurrentTime(float seconds)
This method has no return value.
createSVGMatrixComponents(float a, float b, float c, float d, float e, float f)
This method returns a **SVGMatrix.**
createSVGRect()
This method returns a **SVGRect.**
createSVGPoint()
This method returns a **SVGPoint.**
createSVGPath()
This method returns a **SVGPath.**
createSVGRGBColor(float red, float green, float blue)
This method returns a **SVGRGBColor.**
This method throws SVGException exceptions.
moveFocus(short motionType)
This method has no return value.
This method throws DOMException exceptions.
setFocus(EventTarget theObject)
This method has no return value.
This method throws DOMException exceptions.
getCurrentFocusedObject()
This method returns a **EventTarget.**

Interface org.w3c.dom.svg.SVGRGBColor
The org.w3c.dom.svg.SVGRGBColor interface has the following methods:

getRed()
This method returns a **long.**
setRed(long red)
This method has no return value.
getGreen()
This method returns a **long.**
setGreen(long green)
This method has no return value.
getBlue()
This method returns a **long.**

setBlue(long blue)
This method has no return value.

Interface org.w3c.dom.svg.SVGRect

The org.w3c.dom.svg.SVGRect interface has the following methods:

getX()
This method returns a **float.**

setX(float x)
This method has no return value.

getY()
This method returns a **float.**

setY(float y)
This method has no return value.

getWidth()
This method returns a **float.**

setWidth(float width)
This method has no return value.

getHeight()
This method returns a **float.**

setHeight(float height)
This method has no return value.

Interface org.w3c.dom.svg.SVGPoint

The org.w3c.dom.svg.SVGPoint interface has the following methods:

getX()
This method returns a **float.**

setX(float x)
This method has no return value.

getY()
This method returns a **float.**

setY(float y)
This method has no return value.

matrixTransform(SVGMatrix matrix)
This method returns a **SVGPoint.**

Interface constants org.w3c.dom.svg.SVGPath

The org.w3c.dom.svg.SVGPath interface has the following constants:

MOVE_TO
This constant is a `short` and its value is **77.**

LINE_TO
This constant is a `short` and its value is **76.**

CURVE_TO
This constant is a `short` and its value is **67.**

QUAD_TO
This constant is a `short` and its value is **81.**

CLOSE
This constant is a `short` and its value is **90.**

Interface org.w3c.dom.svg.SVGPath

The org.w3c.dom.svg.SVGPath interface has the following methods:

getNumberOfSegments()
This method returns a **long.**

getSegment(long cmdIndex)
This method returns a **short.**
This method throws DOMException exceptions.

getSegmentParam(long cmdIndex, long paramIndex)
This method returns a **float.**
This method throws DOMException exceptions.

moveTo(float x, float y)
This method has no return value.

lineTo(float x, float y)
This method has no return value.
quadTo(float x1, float y1, float x2, float y2)
This method has no return value.
curveTo(float x1, float y1, float x2, float y2, float x3, float y3)
This method has no return value.
close()
This method has no return value.

Interface org.w3c.dom.svg.SVGMatrix
The org.w3c.dom.svg.SVGMatrix interface has the following methods:
getComponent(long index)
This method returns a **float.**
This method throws DOMException exceptions.
mMultiply(SVGMatrix secondMatrix)
This method returns a **SVGMatrix.**
inverse()
This method returns a **SVGMatrix.**
This method throws SVGException exceptions.
mTranslate(float x, float y)
This method returns a **SVGMatrix.**
mScale(float scaleFactor)
This method returns a **SVGMatrix.**
mRotate(float angle)
This method returns a **SVGMatrix.**

Interface org.w3c.dom.svg.SVGLocatable
The org.w3c.dom.svg.SVGLocatable interface has the following methods:
getBBox()
This method returns a **SVGRect.**
getScreenCTM()
This method returns a **SVGMatrix.**
getScreenBBox()
This method returns a **SVGRect.**

Interface org.w3c.dom.svg.SVGLocatableElement
org.w3c.dom.svg.SVGLocatableElement extends SVGElement, SVGLocatable.

Interface org.w3c.dom.svg.TraitAccess
The org.w3c.dom.svg.TraitAccess interface has the following methods:
getTrait(String name)
This method returns a **String.**
This method throws DOMException exceptions.
getTraitNS(String namespaceURI, String name)
This method returns a **String.**
This method throws DOMException exceptions.
getFloatTrait(String name)
This method returns a **float.**
This method throws DOMException exceptions.
getFloatListTrait(String name)
This method returns a **float[].**
This method throws DOMException exceptions.
getMatrixTrait(String name)
This method returns a **SVGMatrix.**
This method throws DOMException exceptions.
getRectTrait(String name)
This method returns a **SVGRect.**
This method throws DOMException exceptions.
getPathTrait(String name)
This method returns a **SVGPath.**

This method throws DOMException exceptions.

getRGBColorTrait(String name)

This method returns a **SVGRGBColor.**

This method throws DOMException exceptions.

getPresentationTrait(String name)

This method returns a **String.**

This method throws DOMException exceptions.

getPresentationTraitNS(String namespaceURI, String name)

This method returns a **String.**

This method throws DOMException exceptions.

getFloatPresentationTrait(String name)

This method returns a **float.**

This method throws DOMException exceptions.

getFloatListPresentationTrait(String name)

This method returns a **float[].**

This method throws DOMException exceptions.

getMatrixPresentationTrait(String name)

This method returns a **SVGMatrix.**

This method throws DOMException exceptions.

getRectPresentationTrait(String name)

This method returns a **SVGRect.**

This method throws DOMException exceptions.

getPathPresentationTrait(String name)

This method returns a **SVGPath.**

This method throws DOMException exceptions.

getRGBColorPresentationTrait(String name)

This method returns a **SVGRGBColor.**

This method throws DOMException exceptions.

setTrait(String name, String value)

This method has no return value.

This method throws DOMException exceptions.

setTraitNS(String namespaceURI, String name, String value)

This method has no return value.

This method throws DOMException exceptions.

setFloatTrait(String name, float value)

This method has no return value.

This method throws DOMException exceptions.

setFloatListTrait(String name, float[] value)

This method has no return value.

This method throws DOMException exceptions.

setMatrixTrait(String name, SVGMatrix matrix)

This method has no return value.

This method throws DOMException exceptions.

setRectTrait(String name, SVGRect rect)

This method has no return value.

This method throws DOMException exceptions.

setPathTrait(String name, SVGPath path)

This method has no return value.

This method throws DOMException exceptions.

setRGBColorTrait(String name, SVGRGBColor color)

This method has no return value.

This method throws DOMException exceptions.

Interface org.w3c.dom.svg.SVGElement

org.w3c.dom.svg.SVGElement extends Element, EventTarget, TraitAccess.

The org.w3c.dom.svg.SVGElement interface has the following methods:

getId()

This method returns a **String.**

setId(String id)

This method has no return value.

Interface org.w3c.dom.svg.SVGTimedElement

org.w3c.dom.svg.SVGTimedElement extends SVGElement, smil::ElementTimeControl.

The org.w3c.dom.svg.SVGTimedElement interface has the following methods:

pauseElement()

This method has no return value.

resumeElement()

This method has no return value.

getIsPaused()

This method returns a **boolean.**

Interface org.w3c.dom.svg.SVGAnimationElement

org.w3c.dom.svg.SVGAnimationElement extends SVGTimedElement.

Interface org.w3c.dom.svg.SVGVisualMediaElement

org.w3c.dom.svg.SVGVisualMediaElement extends SVGLocatableElement, SVGTimedElement.

Interface org.w3c.dom.svg.SVGTimer

org.w3c.dom.svg.SVGTimer extends events::EventTarget.

The org.w3c.dom.svg.SVGTimer interface has the following methods:

getDelay()

This method returns a **long.**

setDelay(long delay)

This method has no return value.

getRepeatInterval()

This method returns a **long.**

setRepeatInterval(long repeatInterval)

This method has no return value.

getRunning()

This method returns a **boolean.**

start()

This method has no return value.

stop()

This method has no return value.

Interface org.w3c.dom.svg.SVGGlobal

The org.w3c.dom.svg.SVGGlobal interface has the following methods:

createTimer(long initialInterval, long repeatInterval)

This method returns a **SVGTimer.**

getURL(String iri, AsyncStatusCallback callback)

This method has no return value.

postURL(String iri, String data, AsyncStatusCallback callback, String type, String encoding)

This method has no return value.

parseXML(String data, Document contextDoc)

This method returns a **Node.**

Interface org.w3c.dom.svg.AsyncStatusCallback

The org.w3c.dom.svg.AsyncStatusCallback interface has the following methods:

operationComplete(AsyncURLStatus status)

This method has no return value.

Interface org.w3c.dom.svg.AsyncURLStatus

The org.w3c.dom.svg.AsyncURLStatus interface has the following methods:

getSuccess()

This method returns a **boolean.**

getContentType()

This method returns a **String.**

getContent()

This method returns a **String.**

Interface org.w3c.dom.svg.EventListenerInitializer2

The org.w3c.dom.svg.EventListenerInitializer2 interface has the following methods:

initializeEventListeners(Element scriptElement)

This method has no return value.

createEventListener(Element handlerElement)

This method returns a **EventListener.**

Q Perl Language Binding

Contents

Q.1 Module dom 406
Q.2 Module views 408
Q.3 Module events 408
Q.4 Module smil 409
Q.5 Module svg 410

This appendix is normative.

Fields in this binding are implemented as methods which get the value when not parameter is passed, and set it when it is given. Upon setting the return value is that of the field after it has been set. Note that it may differ from the value that was passed to the setter as normalization may have taken place.

Q.1 Module dom

Package W3C::DOM::DOMException

The W3C::DOM::DOMException package holds the following constants:

INDEX_SIZE_ERR
This constant is a number and its value is **1**.

DOMSTRING_SIZE_ERR
This constant is a number and its value is **2**.

HIERARCHY_REQUEST_ERR
This constant is a number and its value is **3**.

WRONG_DOCUMENT_ERR
This constant is a number and its value is **4**.

INVALID_CHARACTER_ERR
This constant is a number and its value is **5**.

NO_DATA_ALLOWED_ERR
This constant is a number and its value is **6**.

NO_MODIFICATION_ALLOWED_ERR
This constant is a number and its value is **7**.

NOT_FOUND_ERR
This constant is a number and its value is **8**.

NOT_SUPPORTED_ERR
This constant is a number and its value is **9**.

INUSE_ATTRIBUTE_ERR
This constant is a number and its value is **10**.

INVALID_STATE_ERR
This constant is a number and its value is **11**.

SYNTAX_ERR
This constant is a number and its value is **12**.

INVALID_MODIFICATION_ERR
This constant is a number and its value is **13**.

NAMESPACE_ERR
This constant is a number and its value is **14**.

INVALID_ACCESS_ERR
This constant is a number and its value is **15**.

VALIDATION_ERR
This constant is a number and its value is **16**.

TYPE_MISMATCH_ERR
This constant is a number and its value is **17**.

Class W3C::DOM::DOMException

The W3C::DOM::DOMException class has the following fields:

code

This field holds a **number**.

Class W3C::DOM::Node

The W3C::DOM::Node class has the following fields:

namespaceURI

This field holds a **string**.

localName

This field holds a **string**.

parentNode

This field holds a **W3C::DOM::Node**.

ownerDocument

This field holds a **W3C::DOM::Document**.

textContent

This field holds a **string**.

The W3C::DOM::Node class has the following methods:

appendChild($newChild)

This method returns a **W3C::DOM::Node**. The $newChild parameter is a **W3C::DOM::Node**.

insertBefore($newChild, $refChild)

This method returns a **W3C::DOM::Node**. The $newChild parameter is a **W3C::DOM::Node**. The $refChild parameter is a **W3C::DOM::Node**.

removeChild($oldChild)

This method returns a **W3C::DOM::Node**. The $oldChild parameter is a **W3C::DOM::Node**.

cloneNode($deep)

This method returns a **W3C::DOM::Node**. The $deep parameter is a **boolean**.

Class W3C::DOM::Element

W3C::DOM::Element inherits from W3C::DOM::Node, W3C::DOM::ElementTraversal.

The W3C::DOM::Element class has the following methods:

getAttributeNS($namespaceURI, $localName)

This method returns a **string**. The $namespaceURI parameter is a **string**. The $localName parameter is a **string**.

setAttributeNS($namespaceURI, $qualifiedName, $value)

This method has no return value. The $namespaceURI parameter is a **string**. The $qualifiedName parameter is a **string**. The $value parameter is a **string**.

getAttribute($name)

This method returns a **string**. The $name parameter is a **string**.

setAttribute($name, $value)

This method has no return value. The $name parameter is a **string**. The $value parameter is a **string**.

Class W3C::DOM::Document

W3C::DOM::Document inherits from W3C::DOM::Node.

The W3C::DOM::Document class has the following fields:

documentElement

This field holds a **W3C::DOM::Element**.

The W3C::DOM::Document class has the following methods:

createElementNS($namespaceURI, $qualifiedName)

This method returns a **W3C::DOM::Element**. The $namespaceURI parameter is a **string**. The $qualifiedName parameter is a **string**.

getElementById($elementId)

This method returns a **W3C::DOM::Element**. The $elementId parameter is a **string**.

Class W3C::DOM::ElementTraversal

The W3C::DOM::ElementTraversal class has the following fields:

firstElementChild

This field holds a **W3C::DOM::Element**.

lastElementChild
This field holds a **W3C::DOM::Element**.
nextElementSibling
This field holds a **W3C::DOM::Element**.
previousElementSibling
This field holds a **W3C::DOM::Element**.
childElementCount
This field holds a **number**.

Class W3C::DOM::Location
The W3C::DOM::Location class has the following methods:
assign($iri)
This method has no return value. The $iri parameter is a **string**.
reload()
This method has no return value.

Class W3C::DOM::Window
The W3C::DOM::Window class has the following fields:
parent
This field holds a **W3C::DOM::Window**.
location
This field holds a **W3C::DOM::Location**.

Q.2 Module `views`

Class W3C::DOM::Views::AbstractView
The W3C::DOM::Views::AbstractView class has the following fields:
document
This field holds a **W3C::DOM::Views::DocumentView**.

Class W3C::DOM::Views::DocumentView
The W3C::DOM::Views::DocumentView class has the following fields:
defaultView
This field holds a **W3C::DOM::Views::AbstractView**.

Q.3 Module `events`

Class W3C::DOM::Events::EventTarget
The W3C::DOM::Events::EventTarget class has the following methods:
addEventListener($type, $listener, $useCapture)
This method has no return value. The $type parameter is a **string**. The $listener parameter is a **W3C::DOM::Events::EventListener**. The $useCapture parameter is a **boolean**.
removeEventListener($type, $listener, $useCapture)
This method has no return value. The $type parameter is a **string**. The $listener parameter is a **W3C::DOM::Events::EventListener**. The $useCapture parameter is a **boolean**.

Class W3C::DOM::Events::EventListener
The W3C::DOM::Events::EventListener class has the following methods:
handleEvent($evt)
This method has no return value. The $evt parameter is a **W3C::DOM::Events::Event**.

Class W3C::DOM::Events::Event
The W3C::DOM::Events::Event class has the following fields:
target
This field holds a **W3C::DOM::Events::EventTarget**.
currentTarget
This field holds a **W3C::DOM::Events::EventTarget**.
type
This field holds a **string**.
cancelable
This field holds a **boolean**.

defaultPrevented
This field holds a **boolean**.

The W3C::DOM::Events::Event class has the following methods:

stopPropagation()
This method has no return value.

preventDefault()
This method has no return value.

Class W3C::DOM::Events::MouseEvent

W3C::DOM::Events::MouseEvent inherits from W3C::DOM::Events::UIEvent.

The W3C::DOM::Events::MouseEvent class has the following fields:

screenX
This field holds a **number**.

screenY
This field holds a **number**.

clientX
This field holds a **number**.

clientY
This field holds a **number**.

button
This field holds a **number**.

Class W3C::DOM::Events::MouseWheelEvent

W3C::DOM::Events::MouseWheelEvent inherits from W3C::DOM::Events::MouseEvent.

The W3C::DOM::Events::MouseWheelEvent class has the following fields:

wheelDelta
This field holds a **number**.

Class W3C::DOM::Events::TextEvent

W3C::DOM::Events::TextEvent inherits from W3C::DOM::Events::UIEvent.

The W3C::DOM::Events::TextEvent class has the following fields:

data
This field holds a **string**.

Class W3C::DOM::Events::KeyboardEvent

W3C::DOM::Events::KeyboardEvent inherits from W3C::DOM::Events::UIEvent.

The W3C::DOM::Events::KeyboardEvent class has the following fields:

keyIdentifier
This field holds a **string**.

Class W3C::DOM::Events::UIEvent

W3C::DOM::Events::UIEvent inherits from W3C::DOM::Events::Event.

The W3C::DOM::Events::UIEvent class has the following fields:

detail
This field holds a **number**.

Class W3C::DOM::Events::ProgressEvent

W3C::DOM::Events::ProgressEvent inherits from W3C::DOM::Events::Event.

The W3C::DOM::Events::ProgressEvent class has the following fields:

lengthComputable
This field holds a **boolean**.

loaded
This field holds a **number**.

total
This field holds a **number**.

Q.4 Module `smil`

Class W3C::DOM::SMIL::ElementTimeControl

The W3C::DOM::SMIL::ElementTimeControl class has the following methods:

beginElementAt($offset)
This method has no return value. The $offset parameter is a **number**.

beginElement()

This method has no return value.

endElementAt($offset)

This method has no return value. The $offset parameter is a **number**.

endElement()

This method has no return value.

Class W3C::DOM::SMIL::TimeEvent

W3C::DOM::SMIL::TimeEvent inherits from W3C::DOM::Events::Event.

The W3C::DOM::SMIL::TimeEvent class has the following fields:

detail

This field holds a **number**.

Q.5 Module svg

Package W3C::DOM::SVG::SVGException

The W3C::DOM::SVG::SVGException package holds the following constants:

SVG_WRONG_TYPE_ERR

This constant is a number and its value is **0**.

SVG_INVALID_VALUE_ERR

This constant is a number and its value is **1**.

SVG_MATRIX_NOT_INVERTABLE

This constant is a number and its value is **2**.

Class W3C::DOM::SVG::SVGException

The W3C::DOM::SVG::SVGException class has the following fields:

code

This field holds a **number**.

Class W3C::DOM::SVG::SVGDocument

W3C::DOM::SVG::SVGDocument inherits from W3C::DOM::Document, W3C::DOM::Events::EventTarget.

Class W3C::DOM::SVG::SVGUseElement

W3C::DOM::SVG::SVGUseElement inherits from W3C::DOM::SVG::SVGLocatableElement.

Class W3C::DOM::SVG::SVGElementInstance

W3C::DOM::SVG::SVGElementInstance inherits from W3C::DOM::Events::EventTarget.

The W3C::DOM::SVG::SVGElementInstance class has the following fields:

correspondingElement

This field holds a **W3C::DOM::SVG::SVGElement**.

correspondingUseElement

This field holds a **W3C::DOM::SVG::SVGUseElement**.

Package W3C::DOM::SVG::SVGSVGElement

The W3C::DOM::SVG::SVGSVGElement package has the following constants:

NAV_AUTO

This constant is a number and its value is **1**.

NAV_NEXT

This constant is a number and its value is **2**.

NAV_PREV

This constant is a number and its value is **3**.

NAV_UP

This constant is a number and its value is **4**.

NAV_UP_RIGHT

This constant is a number and its value is **5**.

NAV_RIGHT

This constant is a number and its value is **6**.

NAV_DOWN_RIGHT

This constant is a number and its value is **7**.

NAV_DOWN

This constant is a number and its value is **8**.

NAV_DOWN_LEFT

This constant is a number and its value is **9**.

NAV_LEFT

This constant is a number and its value is **10**.

NAV_UP_LEFT

This constant is a number and its value is **11**.

Class W3C::DOM::SVG::SVGSVGElement

W3C::DOM::SVG::SVGSVGElement inherits from W3C::DOM::SVG::SVGLocatableElement, W3C::DOM::SVG::SVGTimedElement.

The W3C::DOM::SVG::SVGSVGElement class has the following fields:

currentScale

This field holds a **number**.

currentRotate

This field holds a **number**.

currentTranslate

This field holds a **W3C::DOM::SVG::SVGPoint**.

viewport

This field holds a **W3C::DOM::SVG::SVGRect**.

The W3C::DOM::SVG::SVGSVGElement class has the following methods:

getCurrentTime()

This method returns a **number**.

setCurrentTime($seconds)

This method has no return value. The $seconds parameter is a **number**.

createSVGMatrixComponents($a, $b, $c, $d, $e, $f)

This method returns a **W3C::DOM::SVG::SVGMatrix**. The $a parameter is a **number**. The $b parameter is a **number**. The $c parameter is a **number**. The $d parameter is a **number**. The $e parameter is a **number**. The $f parameter is a **number**.

createSVGRect()

This method returns a **W3C::DOM::SVG::SVGRect**.

createSVGPoint()

This method returns a **W3C::DOM::SVG::SVGPoint**.

createSVGPath()

This method returns a **W3C::DOM::SVG::SVGPath**.

createSVGRGBColor($red, $green, $blue)

This method returns a **W3C::DOM::SVG::SVGRGBColor**. The $red parameter is a **number**. The $green parameter is a **number**. The $blue parameter is a **number**.

moveFocus($motionType)

This method has no return value. The $motionType parameter is a **number**.

setFocus($theObject)

This method has no return value. The $theObject parameter is a **W3C::DOM::Events::EventTarget**.

getCurrentFocusedObject()

This method returns a **W3C::DOM::Events::EventTarget**.

Class W3C::DOM::SVG::SVGRGBColor

The W3C::DOM::SVG::SVGRGBColor class has the following fields:

red

This field holds a **number**.

green

This field holds a **number**.

blue

This field holds a **number**.

Class W3C::DOM::SVG::SVGRect

The W3C::DOM::SVG::SVGRect class has the following fields:

x

This field holds a **number**.

y

This field holds a **number**.

width
This field holds a **number**.

height
This field holds a **number**.

Class W3C::DOM::SVG::SVGPoint

The W3C::DOM::SVG::SVGPoint class has the following fields:

x
This field holds a **number**.

y
This field holds a **number**.

The W3C::DOM::SVG::SVGPoint class has the following methods:

matrixTransform($matrix)
This method returns a **W3C::DOM::SVG::SVGPoint**. The $matrix parameter is a **W3C::DOM::SVG::SVGMatrix**.

Package W3C::DOM::SVG::SVGPath

The W3C::DOM::SVG::SVGPath package has the following constants:

MOVE_TO
This constant is a number and its value is **77**.

LINE_TO
This constant is a number and its value is **76**.

CURVE_TO
This constant is a number and its value is **67**.

QUAD_TO
This constant is a number and its value is **81**.

CLOSE
This constant is a number and its value is **90**.

Class W3C::DOM::SVG::SVGPath

The W3C::DOM::SVG::SVGPath class has the following fields:

numberOfSegments
This field holds a **number**.

The W3C::DOM::SVG::SVGPath class has the following methods:

getSegment($cmdIndex)
This method returns a **number**. The $cmdIndex parameter is a **number**.

getSegmentParam($cmdIndex, $paramIndex)
This method returns a **number**. The $cmdIndex parameter is a **number**. The $paramIndex parameter is a **number**.

moveTo($x, $y)
This method has no return value. The $x parameter is a **number**. The $y parameter is a **number**.

lineTo($x, $y)
This method has no return value. The $x parameter is a **number**. The $y parameter is a **number**.

quadTo($x1, $y1, $x2, $y2)
This method has no return value. The $x1 parameter is a **number**. The $y1 parameter is a **number**. The $x2 parameter is a **number**. The $y2 parameter is a **number**.

curveTo($x1, $y1, $x2, $y2, $x3, $y3)
This method has no return value. The $x1 parameter is a **number**. The $y1 parameter is a **number**. The $x2 parameter is a **number**. The $y2 parameter is a **number**. The $x3 parameter is a **number**. The $y3 parameter is a **number**.

close()
This method has no return value.

Class W3C::DOM::SVG::SVGMatrix

The W3C::DOM::SVG::SVGMatrix class has the following methods:

getComponent($index)
This method returns a **number**. The $index parameter is a **number**.

mMultiply($secondMatrix)
This method returns a **W3C::DOM::SVG::SVGMatrix**. The $secondMatrix parameter is a **W3C::DOM::SVG::SVGMatrix**.

inverse()
This method returns a **W3C::DOM::SVG::SVGMatrix**.

mTranslate($x, $y)
This method returns a **W3C::DOM::SVG::SVGMatrix**. The $x parameter is a **number**. The $y parameter is a **number**.

mScale($scaleFactor)
This method returns a **W3C::DOM::SVG::SVGMatrix**. The $scaleFactor parameter is a **number**.

mRotate($angle)
This method returns a **W3C::DOM::SVG::SVGMatrix**. The $angle parameter is a **number**.

Class W3C::DOM::SVG::SVGLocatable

The W3C::DOM::SVG::SVGLocatable class has the following methods:

getBBox()
This method returns a **W3C::DOM::SVG::SVGRect**.

getScreenCTM()
This method returns a **W3C::DOM::SVG::SVGMatrix**.

getScreenBBox()
This method returns a **W3C::DOM::SVG::SVGRect**.

Class W3C::DOM::SVG::SVGLocatableElement

W3C::DOM::SVG::SVGLocatableElement inherits from W3C::DOM::SVG::SVGElement, W3C::DOM::SVG::SVGLocatable.

Class W3C::DOM::SVG::TraitAccess

The W3C::DOM::SVG::TraitAccess class has the following methods:

getTrait($name)
This method returns a **string**. The $name parameter is a **string**.

getTraitNS($namespaceURI, $name)
This method returns a **string**. The $namespaceURI parameter is a **string**. The $name parameter is a **string**.

getFloatTrait($name)
This method returns a **number**. The $name parameter is a **string**.

getFloatListTrait($name)
This method returns a **Perl array**. The $name parameter is a **string**.

getMatrixTrait($name)
This method returns a **W3C::DOM::SVG::SVGMatrix**. The $name parameter is a **string**.

getRectTrait($name)
This method returns a **W3C::DOM::SVG::SVGRect**. The $name parameter is a **string**.

getPathTrait($name)
This method returns a **W3C::DOM::SVG::SVGPath**. The $name parameter is a **string**.

getRGBColorTrait($name)
This method returns a **W3C::DOM::SVG::SVGRGBColor**. The $name parameter is a **string**.

getPresentationTrait($name)
This method returns a **string**. The $name parameter is a **string**.

getPresentationTraitNS($namespaceURI, $name)
This method returns a **string**. The $namespaceURI parameter is a **string**. The $name parameter is a **string**.

getFloatPresentationTrait($name)
This method returns a **number**. The $name parameter is a **string**.

getFloatListPresentationTrait($name)
This method returns a **Perl array**. The $name parameter is a **string**.

getMatrixPresentationTrait($name)
This method returns a **W3C::DOM::SVG::SVGMatrix**. The $name parameter is a **string**.

getRectPresentationTrait($name)
This method returns a **W3C::DOM::SVG::SVGRect**. The $name parameter is a **string**.

getPathPresentationTrait($name)
This method returns a **W3C::DOM::SVG::SVGPath**. The $name parameter is a **string**.

getRGBColorPresentationTrait($name)
This method returns a **W3C::DOM::SVG::SVGRGBColor**. The $name parameter is a **string**.

setTrait($name, $value)
This method has no return value. The $name parameter is a **string**. The $value parameter is a **string**.

setTraitNS($namespaceURI, $name, $value)
This method has no return value. The $namespaceURI parameter is a **string**. The $name parameter is a **string**. The $value parameter is a **string**.

setFloatTrait($name, $value)
This method has no return value. The $name parameter is a **string**. The $value parameter is a **number**.

setFloatListTrait($name, $value)
This method has no return value. The $name parameter is a **string**. The $value parameter is a **Perl array**.

setMatrixTrait($name, $matrix)
This method has no return value. The $name parameter is a **string**. The $matrix parameter is a **W3C::DOM::SVG::SVGMatrix**.

setRectTrait($name, $rect)
This method has no return value. The $name parameter is a **string**. The $rect parameter is a **W3C::DOM::SVG::SVGRect**.

setPathTrait($name, $path)
This method has no return value. The $name parameter is a **string**. The $path parameter is a **W3C::DOM::SVG::SVGPath**.

setRGBColorTrait($name, $color)
This method has no return value. The $name parameter is a **string**. The $color parameter is a **W3C::DOM::SVG::SVGRGBColor**.

Class W3C::DOM::SVG::SVGElement
W3C::DOM::SVG::SVGElement inherits from W3C::DOM::Element, W3C::DOM::Events::EventTarget, W3C::DOM::SVG::TraitAccess.

The W3C::DOM::SVG::SVGElement class has the following fields:

id
This field holds a **string**.

Class W3C::DOM::SVG::SVGTimedElement
W3C::DOM::SVG::SVGTimedElement inherits from W3C::DOM::SVG::SVGElement, ::smil::ElementTimeControl.

The W3C::DOM::SVG::SVGTimedElement class has the following fields:

isPaused
This field holds a **boolean**.

The W3C::DOM::SVG::SVGTimedElement class has the following methods:

pauseElement()
This method has no return value.

resumeElement()
This method has no return value.

Class W3C::DOM::SVG::SVGAnimationElement
W3C::DOM::SVG::SVGAnimationElement inherits from W3C::DOM::SVG::SVGTimedElement.

Class W3C::DOM::SVG::SVGVisualMediaElement
W3C::DOM::SVG::SVGVisualMediaElement inherits from W3C::DOM::SVG::SVGLocatableElement, W3C::DOM::SVG::SVGTimedElement.

Class W3C::DOM::SVG::SVGTimer
W3C::DOM::SVG::SVGTimer inherits from ::events::EventTarget.

The W3C::DOM::SVG::SVGTimer class has the following fields:

delay
This field holds a **number**.

repeatInterval
This field holds a **number**.

running
This field holds a **boolean**.

The W3C::DOM::SVG::SVGTimer class has the following methods:

start()
This method has no return value.

stop()
This method has no return value.

Class W3C::DOM::SVG::SVGGlobal

The W3C::DOM::SVG::SVGGlobal class has the following methods:

createTimer($initialInterval, $repeatInterval)
This method returns a **W3C::DOM::SVG::SVGTimer**. The $initialInterval parameter is a **number**. The $repeatInterval parameter is a **number**.

getURL($iri, $callback)
This method has no return value. The $iri parameter is a **string**. The $callback parameter is a **W3C::DOM::SVG::AsyncStatusCallback**.

postURL($iri, $data, $callback, $type, $encoding)
This method has no return value. The $iri parameter is a **string**. The $data parameter is a **string**. The $callback parameter is a **W3C::DOM::SVG::AsyncStatusCallback**. The $type parameter is a **string**. The $encoding parameter is a **string**.

parseXML($data, $contextDoc)
This method returns a **W3C::DOM::Node**. The $data parameter is a **string**. The $contextDoc parameter is a **W3C::DOM::Document**.

Class W3C::DOM::SVG::AsyncStatusCallback

The W3C::DOM::SVG::AsyncStatusCallback class has the following methods:

operationComplete($status)
This method has no return value. The $status parameter is a **W3C::DOM::SVG::AsyncURLStatus**.

Class W3C::DOM::SVG::AsyncURLStatus

The W3C::DOM::SVG::AsyncURLStatus class has the following fields:

success
This field holds a **boolean**.

contentType
This field holds a **string**.

content
This field holds a **string**.

Class W3C::DOM::SVG::EventListenerInitializer2

The W3C::DOM::SVG::EventListenerInitializer2 class has the following methods:

initializeEventListeners($scriptElement)
This method has no return value. The $scriptElement parameter is a **W3C::DOM::Element**.

createEventListener($handlerElement)
This method returns a **W3C::DOM::Events::EventListener**. The $handlerElement parameter is a **W3C::DOM::Element**.

R Python Language Binding

Contents

R.1 Module dom 416
R.2 Module views 418
R.3 Module events 418
R.4 Module smil 419
R.5 Module svg 420

This appendix is normative.

R.1 Module `dom`

Error constants for DOMException

The DOMException class holds the following constants:

INDEX_SIZE_ERR
This constant is an `int` and its value is **1**.

DOMSTRING_SIZE_ERR
This constant is an `int` and its value is **2**.

HIERARCHY_REQUEST_ERR
This constant is an `int` and its value is **3**.

WRONG_DOCUMENT_ERR
This constant is an `int` and its value is **4**.

INVALID_CHARACTER_ERR
This constant is an `int` and its value is **5**.

NO_DATA_ALLOWED_ERR
This constant is an `int` and its value is **6**.

NO_MODIFICATION_ALLOWED_ERR
This constant is an `int` and its value is **7**.

NOT_FOUND_ERR
This constant is an `int` and its value is **8**.

NOT_SUPPORTED_ERR
This constant is an `int` and its value is **9**.

INUSE_ATTRIBUTE_ERR
This constant is an `int` and its value is **10**.

INVALID_STATE_ERR
This constant is an `int` and its value is **11**.

SYNTAX_ERR
This constant is an `int` and its value is **12**.

INVALID_MODIFICATION_ERR
This constant is an `int` and its value is **13**.

NAMESPACE_ERR
This constant is an `int` and its value is **14**.

INVALID_ACCESS_ERR
This constant is an `int` and its value is **15**.

VALIDATION_ERR
This constant is an `int` and its value is **16**.

TYPE_MISMATCH_ERR
This constant is an `int` and its value is **17**.

Exception Class DOMException

The DOMException class has the following attributes:

code
This attribute holds a **int**.

Class Node

The Node class has the following attributes:

namespaceURI
: This attribute holds a **unicode**.

localName
: This attribute holds a **unicode**.

parentNode
: This attribute holds a **Node**.

ownerDocument
: This attribute holds a **Document**.

textContent
: This attribute holds a **unicode**.

The Node class has the following methods:

appendChild(newChild)
: This method returns a **Node**. The newChild argument is a **Node**.

insertBefore(newChild, refChild)
: This method returns a **Node**. The newChild argument is a **Node**. The refChild argument is a **Node**.

removeChild(oldChild)
: This method returns a **Node**. The oldChild argument is a **Node**.

cloneNode(deep)
: This method returns a **Node**. The deep argument is a **boolean**.

Class Element

Element inherits from Node, ElementTraversal.

The Element class has the following methods:

getAttributeNS(namespaceURI, localName)
: This method returns a **unicode**. The namespaceURI argument is a **unicode**. The localName argument is a **unicode**.

setAttributeNS(namespaceURI, qualifiedName, value)
: This method has no return value. The namespaceURI argument is a **unicode**. The qualifiedName argument is a **unicode**. The value argument is a **unicode**.

getAttribute(name)
: This method returns a **unicode**. The name argument is a **unicode**.

setAttribute(name, value)
: This method has no return value. The name argument is a **unicode**. The value argument is a **unicode**.

Class Document

Document inherits from Node.

The Document class has the following attributes:

documentElement
: This attribute holds a **Element**.

The Document class has the following methods:

createElementNS(namespaceURI, qualifiedName)
: This method returns a **Element**. The namespaceURI argument is a **unicode**. The qualifiedName argument is a **unicode**.

getElementById(elementId)
: This method returns a **Element**. The elementId argument is a **unicode**.

Class ElementTraversal

The ElementTraversal class has the following attributes:

firstElementChild
: This attribute holds a **Element**.

lastElementChild
: This attribute holds a **Element**.

nextElementSibling
: This attribute holds a **Element**.

previousElementSibling
: This attribute holds a **Element**.

childElementCount
This attribute holds a **long**.

Class Location
The Location class has the following methods:

assign(iri)
This method has no return value. The iri argument is a **unicode**.

reload()
This method has no return value.

Class Window
The Window class has the following attributes:

parent
This attribute holds a **Window**.

location
This attribute holds a **Location**.

R.2 Module `views`

Class AbstractView
The AbstractView class has the following attributes:

document
This attribute holds a **DocumentView**.

Class DocumentView
The DocumentView class has the following attributes:

defaultView
This attribute holds a **AbstractView**.

R.3 Module `events`

Class EventTarget
The EventTarget class has the following methods:

addEventListener(type, listener, useCapture)
This method has no return value. The type argument is a **unicode**. The listener argument is a **EventListener**. The useCapture argument is a **boolean**.

removeEventListener(type, listener, useCapture)
This method has no return value. The type argument is a **unicode**. The listener argument is a **EventListener**. The useCapture argument is a **boolean**.

Class EventListener
The EventListener class has the following methods:

handleEvent(evt)
This method has no return value. The evt argument is a **Event**.

Class Event
The Event class has the following attributes:

target
This attribute holds a **EventTarget**.

currentTarget
This attribute holds a **EventTarget**.

type
This attribute holds a **unicode**.

cancelable
This attribute holds a **boolean**.

defaultPrevented
This attribute holds a **boolean**.

The Event class has the following methods:

stopPropagation()
This method has no return value.

preventDefault()
This method has no return value.

Class MouseEvent

MouseEvent inherits from UIEvent.

The MouseEvent class has the following attributes:

screenX

This attribute holds a **long**.

screenY

This attribute holds a **long**.

clientX

This attribute holds a **long**.

clientY

This attribute holds a **long**.

button

This attribute holds a **int**.

Class MouseWheelEvent

MouseWheelEvent inherits from MouseEvent.

The MouseWheelEvent class has the following attributes:

wheelDelta

This attribute holds a **long**.

Class TextEvent

TextEvent inherits from UIEvent.

The TextEvent class has the following attributes:

data

This attribute holds a **unicode**.

Class KeyboardEvent

KeyboardEvent inherits from UIEvent.

The KeyboardEvent class has the following attributes:

keyIdentifier

This attribute holds a **unicode**.

Class UIEvent

UIEvent inherits from Event.

The UIEvent class has the following attributes:

detail

This attribute holds a **long**.

Class ProgressEvent

ProgressEvent inherits from Event.

The ProgressEvent class has the following attributes:

lengthComputable

This attribute holds a **boolean**.

loaded

This attribute holds a **long**.

total

This attribute holds a **long**.

R.4 Module `smil`

Class ElementTimeControl

The ElementTimeControl class has the following methods:

beginElementAt(offset)

This method has no return value. The offset argument is a **float**.

beginElement()

This method has no return value.

endElementAt(offset)

This method has no return value. The offset argument is a **float**.

endElement()

This method has no return value.

Class TimeEvent

TimeEvent inherits from Event.

The TimeEvent class has the following attributes:

detail

This attribute holds a **long**.

R.5 Module `svg`

Error constants for SVGException

The SVGException class holds the following constants:

SVG_WRONG_TYPE_ERR

This constant is an `int` and its value is **0**.

SVG_INVALID_VALUE_ERR

This constant is an `int` and its value is **1**.

SVG_MATRIX_NOT_INVERTABLE

This constant is an `int` and its value is **2**.

Exception Class SVGException

The SVGException class has the following attributes:

code

This attribute holds a **int**.

Class SVGDocument

SVGDocument inherits from Document, EventTarget.

Class SVGUseElement

SVGUseElement inherits from SVGLocatableElement.

Class SVGElementInstance

SVGElementInstance inherits from EventTarget.

The SVGElementInstance class has the following attributes:

correspondingElement

This attribute holds a **SVGElement**.

correspondingUseElement

This attribute holds a **SVGUseElement**.

Class constants SVGSVGElement

The SVGSVGElement class has the following constants:

NAV_AUTO

This constant is an `int` and its value is **1**.

NAV_NEXT

This constant is an `int` and its value is **2**.

NAV_PREV

This constant is an `int` and its value is **3**.

NAV_UP

This constant is an `int` and its value is **4**.

NAV_UP_RIGHT

This constant is an `int` and its value is **5**.

NAV_RIGHT

This constant is an `int` and its value is **6**.

NAV_DOWN_RIGHT

This constant is an `int` and its value is **7**.

NAV_DOWN

This constant is an `int` and its value is **8**.

NAV_DOWN_LEFT

This constant is an `int` and its value is **9**.

NAV_LEFT

This constant is an `int` and its value is **10**.

NAV_UP_LEFT

This constant is an `int` and its value is **11**.

Class SVGSVGElement

SVGSVGElement inherits from SVGLocatableElement, SVGTimedElement.

The SVGSVGElement class has the following attributes:

currentScale

This attribute holds a **float**.

currentRotate

This attribute holds a **float**.

currentTranslate

This attribute holds a **SVGPoint**.

viewport

This attribute holds a **SVGRect**.

The SVGSVGElement class has the following methods:

getCurrentTime()

This method returns a **float**.

setCurrentTime(seconds)

This method has no return value. The seconds argument is a **float**.

createSVGMatrixComponents(a, b, c, d, e, f)

This method returns a **SVGMatrix**. The a argument is a **float**. The b argument is a **float**. The c argument is a **float**. The d argument is a **float**. The e argument is a **float**. The f argument is a **float**.

createSVGRect()

This method returns a **SVGRect**.

createSVGPoint()

This method returns a **SVGPoint**.

createSVGPath()

This method returns a **SVGPath**.

createSVGRGBColor(red, green, blue)

This method returns a **SVGRGBColor**. The red argument is a **float**. The green argument is a **float**. The blue argument is a **float**.

moveFocus(motionType)

This method has no return value. The motionType argument is a **int**.

setFocus(theObject)

This method has no return value. The theObject argument is a **EventTarget**.

getCurrentFocusedObject()

This method returns a **EventTarget**.

Class SVGRGBColor

The SVGRGBColor class has the following attributes:

red

This attribute holds a **long**.

green

This attribute holds a **long**.

blue

This attribute holds a **long**.

Class SVGRect

The SVGRect class has the following attributes:

x

This attribute holds a **float**.

y

This attribute holds a **float**.

width

This attribute holds a **float**.

height

This attribute holds a **float**.

Class SVGPoint

The SVGPoint class has the following attributes:

x

This attribute holds a **float**.

y
This attribute holds a **float**.

The SVGPoint class has the following methods:

matrixTransform(matrix)
This method returns a **SVGPoint**. The matrix argument is a **SVGMatrix**.

Class constants SVGPath

The SVGPath class has the following constants:

MOVE_TO
This constant is an `int` and its value is **77**.

LINE_TO
This constant is an `int` and its value is **76**.

CURVE_TO
This constant is an `int` and its value is **67**.

QUAD_TO
This constant is an `int` and its value is **81**.

CLOSE
This constant is an `int` and its value is **90**.

Class SVGPath

The SVGPath class has the following attributes:

numberOfSegments
This attribute holds a **long**.

The SVGPath class has the following methods:

getSegment(cmdIndex)
This method returns a **int**. The cmdIndex argument is a **long**.

getSegmentParam(cmdIndex, paramIndex)
This method returns a **float**. The cmdIndex argument is a **long**. The paramIndex argument is a **long**.

moveTo(x, y)
This method has no return value. The x argument is a **float**. The y argument is a **float**.

lineTo(x, y)
This method has no return value. The x argument is a **float**. The y argument is a **float**.

quadTo(x1, y1, x2, y2)
This method has no return value. The x1 argument is a **float**. The y1 argument is a **float**. The x2 argument is a **float**. The y2 argument is a **float**.

curveTo(x1, y1, x2, y2, x3, y3)
This method has no return value. The x1 argument is a **float**. The y1 argument is a **float**. The x2 argument is a **float**. The y2 argument is a **float**. The x3 argument is a **float**. The y3 argument is a **float**.

close()
This method has no return value.

Class SVGMatrix

The SVGMatrix class has the following methods:

getComponent(index)
This method returns a **float**. The index argument is a **long**.

mMultiply(secondMatrix)
This method returns a **SVGMatrix**. The secondMatrix argument is a **SVGMatrix**.

inverse()
This method returns a **SVGMatrix**.

mTranslate(x, y)
This method returns a **SVGMatrix**. The x argument is a **float**. The y argument is a **float**.

mScale(scaleFactor)
This method returns a **SVGMatrix**. The scaleFactor argument is a **float**.

mRotate(angle)
This method returns a **SVGMatrix**. The angle argument is a **float**.

Class SVGLocatable

The SVGLocatable class has the following methods:

getBBox()

This method returns a **SVGRect**.

getScreenCTM()

This method returns a **SVGMatrix**.

getScreenBBox()

This method returns a **SVGRect**.

Class SVGLocatableElement

SVGLocatableElement inherits from SVGElement, SVGLocatable.

Class TraitAccess

The TraitAccess class has the following methods:

getTrait(name)

This method returns a **unicode**. The name argument is a **unicode**.

getTraitNS(namespaceURI, name)

This method returns a **unicode**. The namespaceURI argument is a **unicode**. The name argument is a **unicode**.

getFloatTrait(name)

This method returns a **float**. The name argument is a **unicode**.

getFloatListTrait(name)

This method returns a **Python list**. The name argument is a **unicode**.

getMatrixTrait(name)

This method returns a **SVGMatrix**. The name argument is a **unicode**.

getRectTrait(name)

This method returns a **SVGRect**. The name argument is a **unicode**.

getPathTrait(name)

This method returns a **SVGPath**. The name argument is a **unicode**.

getRGBColorTrait(name)

This method returns a **SVGRGBColor**. The name argument is a **unicode**.

getPresentationTrait(name)

This method returns a **unicode**. The name argument is a **unicode**.

getPresentationTraitNS(namespaceURI, name)

This method returns a **unicode**. The namespaceURI argument is a **unicode**. The name argument is a **unicode**.

getFloatPresentationTrait(name)

This method returns a **float**. The name argument is a **unicode**.

getFloatListPresentationTrait(name)

This method returns a **Python list**. The name argument is a **unicode**.

getMatrixPresentationTrait(name)

This method returns a **SVGMatrix**. The name argument is a **unicode**.

getRectPresentationTrait(name)

This method returns a **SVGRect**. The name argument is a **unicode**.

getPathPresentationTrait(name)

This method returns a **SVGPath**. The name argument is a **unicode**.

getRGBColorPresentationTrait(name)

This method returns a **SVGRGBColor**. The name argument is a **unicode**.

setTrait(name, value)

This method has no return value. The name argument is a **unicode**. The value argument is a **unicode**.

setTraitNS(namespaceURI, name, value)

This method has no return value. The namespaceURI argument is a **unicode**. The name argument is a **unicode**. The value argument is a **unicode**.

setFloatTrait(name, value)

This method has no return value. The name argument is a **unicode**. The value argument is a **float**.

setFloatListTrait(name, value)

This method has no return value. The name argument is a **unicode**. The value argument is a **Python list**.

setMatrixTrait(name, matrix)

This method has no return value. The name argument is a **unicode**. The matrix argument is a **SVGMatrix**.

setRectTrait(name, rect)

This method has no return value. The name argument is a **unicode**. The rect argument is a **SVGRect**.

setPathTrait(name, path)

This method has no return value. The name argument is a **unicode**. The path argument is a **SVGPath**.

setRGBColorTrait(name, color)

This method has no return value. The name argument is a **unicode**. The color argument is a **SVGRGBColor**.

Class SVGElement

SVGElement inherits from Element, EventTarget, TraitAccess.

The SVGElement class has the following attributes:

id

This attribute holds a **unicode**.

Class SVGTimedElement

SVGTimedElement inherits from SVGElement, smil::ElementTimeControl.

The SVGTimedElement class has the following attributes:

isPaused

This attribute holds a **boolean**.

The SVGTimedElement class has the following methods:

pauseElement()

This method has no return value.

resumeElement()

This method has no return value.

Class SVGAnimationElement

SVGAnimationElement inherits from SVGTimedElement.

Class SVGVisualMediaElement

SVGVisualMediaElement inherits from SVGLocatableElement, SVGTimedElement.

Class SVGTimer

SVGTimer inherits from events::EventTarget.

The SVGTimer class has the following attributes:

delay

This attribute holds a **long**.

repeatInterval

This attribute holds a **long**.

running

This attribute holds a **boolean**.

The SVGTimer class has the following methods:

start()

This method has no return value.

stop()

This method has no return value.

Class SVGGlobal

The SVGGlobal class has the following methods:

createTimer(initialInterval, repeatInterval)

This method returns a **SVGTimer**. The initialInterval argument is a **long**. The repeatInterval argument is a **long**.

getURL(iri, callback)

This method has no return value. The iri argument is a **unicode**. The callback argument is a **AsyncStatusCallback**.

postURL(iri, data, callback, type, encoding)
: This method has no return value. The iri argument is a **unicode**. The data argument is a **unicode**. The callback argument is a **AsyncStatusCallback**. The type argument is a **unicode**. The encoding argument is a **unicode**.

parseXML(data, contextDoc)
: This method returns a **Node**. The data argument is a **unicode**. The contextDoc argument is a **Document**.

Class AsyncStatusCallback

The AsyncStatusCallback class has the following methods:

operationComplete(status)
: This method has no return value. The status argument is a **AsyncURLStatus**.

Class AsyncURLStatus

The AsyncURLStatus class has the following attributes:

success
: This attribute holds a **boolean**.

contentType
: This attribute holds a **unicode**.

content
: This attribute holds a **unicode**.

Class EventListenerInitializer2

The EventListenerInitializer2 class has the following methods:

initializeEventListeners(scriptElement)
: This method has no return value. The scriptElement argument is a **Element**.

createEventListener(handlerElement)
: This method returns a **EventListener**. The handlerElement argument is a **Element**.

S References

Contents

S.1 Normative references 426
S.2 Informative references 430

S.1 Normative references

[ATAG]
Authoring Tool Accessibility Guidelines 1.0, J. Treviranus, J. Richards, I. Jacobs, C. McCathieNevile, eds. World Wide Web Consortium, 03 February 2000.
This edition of ATAG 1.0 is http://www.w3.org/TR/2000/REC-ATAG10-20000203/.
The latest edition of ATAG 1.0 is available at http://www.w3.org/TR/ATAG10/.

[AWWW]
Architecture of the World Wide Web, Volume One, I. Jacobs, N. Walsh, eds. World Wide Web Consortium, 15 December 2004.
This edition of AWWW Volume 1 is http://www.w3.org/TR/2004/REC-webarch-20041215/.
The latest edition of AWWW Volume 1 is available at http://www.w3.org/TR/webarch/.

[BCP19]
IANA Charset Registration Procedures, N. Freed, J. Postel, October 2000.
Available at http://tools.ietf.org/html/bcp19.

[BCP47]
IETF BCP 47, currently represented by a concatenation of *Tags for the Identification of Languages*, A. Phillips, M. Davis, September 2006 and *Matching of Language Tags*, A. Phillips, M. Davis, September 2006.
Available at http://tools.ietf.org/html/bcp47.

[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.
This edition of Charmod 1.0 is http://www.w3.org/TR/2005/REC-charmod-20050215/.
The latest edition of Charmod 1.0 is available at http://www.w3.org/TR/charmod/.

[CSS2]
Cascading Style Sheets, level 2, B. Bos, H. W. Lie, C. Lilley, I. Jacobs, eds. World Wide Web Consortium, 12 May 1998.
This edition of CSS 2 is http://www.w3.org/TR/1998/REC-CSS2-19980512/.
The latest edition of CSS 2 is available at http://www.w3.org/TR/CSS2/.

[DOM2EVENTS]
Document Object Model (DOM) Level 2 Events Specification, T. Pixley, ed. World Wide Web Consortium, 13 November 2000.
This edition of DOM 2 Events is http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/.
The latest edition of DOM 2 Events is available at http://www.w3.org/TR/DOM-Level-2-Events/.

[DOM2VIEWS]
Document Object Model (DOM) Level 2 Views Specification, A. Le Hors, L. Cable, eds. World Wide Web Consortium, 13 November 2000.
This edition of DOM 2 Views is http://www.w3.org/TR/2000/REC-DOM-Level-2-Views-20001113/.
The latest edition of DOM 2 Views is available at http://www.w3.org/TR/DOM-Level-2-Views/.

[DOM3]
Document Object Model (DOM) Level 3 Core Specification, A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, eds. World Wide Web Consortium, 07 April 2004.
This edition of DOM 3 Core is http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/.
The latest edition of DOM 3 Core is available at http://www.w3.org/TR/DOM-Level-3-Core/.

[EBNF]
Information technology — Syntactic metalanguage — Extended BNF, International Organization for Standardization, 1996.
Available at http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip. This specification uses the subset of EBNF defined in the Notation section of *Extensible Markup Language (XML) 1.0 (Fourth Edition)* ([XML10], section 6).

[ET]
Element Traversal Specification, D. Schepers, ed. World Wide Web Consortium, 22 December 2008.
This edition of Element Traversal is http://www.w3.org/TR/2008/REC-ElementTraversal-20081222/.
The latest edition of Element Traversal is available at http://www.w3.org/TR/ElementTraversal/.

[FOLEY-VANDAM]
Computer Graphics: Principles and Practice, Second Edition, J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, R. L. Phillips. Addison-Wesley, 1995.

[GML]
OpenGIS Geography Markup Language (GML) Encoding Standard, version 3.2.1, C. Portele, ed. Open GIS Consortium, 27 August 2007.
Available at http://portal.opengeospatial.org/files/?artifact_id=20509.

[IEEE-754]
IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985). Institute of Electrical and Electronics Engineers, 1985.

[JPEG]
ISO/IEC 10918: Information Technology — Digital Compression And Coding Of Continuous-tone Still Images, International Organization for Standardization, September 1992.
Available at http://www.w3.org/Graphics/JPEG/itu-t81.pdf.

[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]
Information Technology — Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language: ISO/IEC FDIS 19757-4:2005(E), International Organization for Standardization, December 2005.
Available at http://www.jtc1sc34.org/repository/0694.pdf.

[PNG]
Portable Network Graphics (PNG) Specification (Second Edition): Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification, ISO/IEC 15948:2003 (E), D. Duce, ed. World Wide Web Consortium, 10 November 2003.
This edition of PNG is http://www.w3.org/TR/2003/REC-PNG-20031110/.
The latest edition of PNG is available at http://www.w3.org/TR/PNG/.

[QAF-SPEC]
QA Framework: Specification Guidelines, K. Dubost, L. Rosenthal, D. Hazaël-Massieux, L. Henderson. World Wide Web Consortium, 17 August 2005.
This edition of QA Framework: Specification Guidelines is http://www.w3.org/TR/2005/REC-qaframe-spec-20050817/.
The latest edition of QA Framework: Specification Guidelines is 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. International Organization for Standardization, 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.

[RFC1952]
GZIP file format specification version 4.3, P. Deutsch, May 1996.
Available at http://tools.ietf.org/html/rfc1952.

[RFC2046]
Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed and N. Borenstein, November 1996. (Note that this RFC obsoletes RFC 1521, RFC 1522 and RFC 1590.)
Available at http://tools.ietf.org/html/rfc2046.

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, March 1997.
Available at http://tools.ietf.org/html/rfc2119.

[RFC2397]
The "data" URL scheme, L. Masinter, August 1998.
Available at http://tools.ietf.org/html/rfc2397.

[RFC2616]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999. (Note that this RFC obsoletes RFC 2068.)
Available at http://tools.ietf.org/html/rfc2616.

[RFC2781]
UTF-16, an encoding of ISO 10646, P. Hoffman and F. Yergeau, February 2000.
Available at http://tools.ietf.org/html/rfc2781.

[RFC3023]
XML Media Types, M. Murata, S. St. Laurent, D. Kohn, January 2001. (Note that a revision to RFC 3023 is in preparation.)
Available at http://tools.ietf.org/html/rfc3023.

[RFC3629]
UTF-8, a transformation format of ISO 10646, F. Yergeau, November 2003. (Note that this RFC obsoletes RFC 2044 and RFC 2279.)
Available at http://tools.ietf.org/html/rfc3629.

[RFC3986]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, January 2005. (Note that RFC 3986 updates RFC 1738 and obsoletes RFC 2732, RFC 2396 and RFC 1808.)
Available at http://tools.ietf.org/html/rfc3986.

[RFC3987]
Internationalized Resource Identifiers (IRIs), M. Dürst, M. Suignard, January 2005.
Available at http://tools.ietf.org/html/rfc3987.

[RFC4329]
Scripting Media Types, B. Höhrmann, April 2006.
Available at http://tools.ietf.org/html/rfc4329.

[SMIL21]
Synchronized Multimedia Integration Language (SMIL 2.1), D. Bulterman, G. Grassel, *et. al.*. World Wide Web Consortium, 13 December 2005.
This edition of SMIL 2 is http://www.w3.org/TR/2005/REC-SMIL2-20051213/.
The latest edition of SMIL 2 is available at http://www.w3.org/TR/SMIL2/.

[SMILANIM]
SMIL Animation, P. Schmitz, A. Cohen. World Wide Web Consortium, 04 September 2001.
This edition of SMIL Animation is http://www.w3.org/TR/2001/REC-smil-animation-20010904/.
The latest edition of SMIL Animation is available at http://www.w3.org/TR/smil-animation/.

[SRGB]
IEC 61966-2-1: 1999: Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB, International Electrotechnical Commission, 1999. ISBN 2-8318-4989-6. (See also *Using the sRGB_v4_ICC_preference.icc profile* and *sRGB profiles* [USING-SRGB, SRGB-PROFILES]).
Available at http://webstore.iec.ch/webstore/webstore.nsf/artnum/025408.

[SVGM11]
Mobile SVG Profiles: SVG Tiny and SVG Basic, T. Capin, ed. World Wide Web Consortium, 14 January 2003.
This edition of SVG Mobile 1.1 is http://www.w3.org/TR/2001/REC-SVGmobile-20030114/.
The latest edition of SVG Mobile 1.1 is available at http://www.w3.org/TR/SVGMobile/.

[UAAG]
User Agent Accessibility Guidelines 1.0, I. Jacobs, J. Gunderson, E. Hansen, eds. World Wide Web Consortium, 17 December 2002.
This edition of UAAG 1.0 is http://www.w3.org/TR/2000/REC-UAAG10-20021217/.
The latest edition of UAAG 1.0 is available at http://www.w3.org/TR/UAAG10/.

[UNICODE]
The Unicode Standard, Version 4.1.0 or later. The Unicode Consortium. Note that *The Unicode Standard, Version 4.1.0* is defined by *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1) as amended by Unicode 4.0.1, (available at http://www.unicode.org/versions/Unicode4.0.1/) and Unicode 4.1.0, (available at http://www.unicode.org/versions/Unicode4.1.0/). Refer to http://www.unicode.org/unicode/standard/versions/ for details.

[UAX9]
Unicode Bidirectional Algorithm, The Unicode Standard Annex #9. The Unicode Consortium, 2008.
The latest version of this annex is available at http://www.unicode.org/reports/tr15/.

[UAX14]
Unicode Line Breaking Algorithm, The Unicode Standard Annex #14. The Unicode Consortium, 2008.
The latest version of this annex is available at http://www.unicode.org/reports/tr14/.

[UAX15]
Unicode Normalization Forms, The Unicode Standard Annex #15. The Unicode Consortium, 2008.
The latest version of this annex is available at http://www.unicode.org/reports/tr15/.

[XLINK10]
XML Linking Language (XLink) Version 1.0, S. DeRose, E. Maler, D. Orchard, eds. World Wide Web Consortium, 27 June 2001.
This edition of XLink 1.0 is http://www.w3.org/TR/2001/REC-xlink-20010627/.
The latest edition of XLink 1.0 is available at http://www.w3.org/TR/xlink/.

[XML10]
Extensible Markup Language (XML) 1.0 (Fourth Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, eds. World Wide Web Consortium, 04 February 2004.
This edition of XML 1.0 is http://www.w3.org/TR/2006/REC-xml-20060816/.
The latest edition of XML 1.0 is available at http://www.w3.org/TR/REC-xml/.

[XML11]
Extensible Markup Language (XML) 1.1 (Second Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan, eds. World Wide Web Consortium, 15 April 2004.
This edition of XML 1.1 is http://www.w3.org/TR/2006/REC-xml11-20060816/.
The latest edition of XML 1.1 is 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, 09 September 2005.
This edition of xml:id is http://www.w3.org/TR/2005/REC-xml-id-20050909/.
The latest edition of xml:id is available at http://www.w3.org/TR/xml-id/.

[XML-BASE]
XML Base, J. Marsh, ed. World Wide Web Consortium, 27 June 2001.
This edition of XML Base is http://www.w3.org/TR/2001/REC-xmlbase-20010627/.
The latest edition of XML Base is available at http://www.w3.org/TR/xmlbase/.

[XML-EVENTS]
XML Events, S. McCarron, S. Pemberton, T. V. Raman, eds. World Wide Web Consortium, 14 October 2003.
This edition of XML Events is http://www.w3.org/TR/2003/REC-xml-events-20031014/.
The latest edition of XML Events is available at http://www.w3.org/TR/xml-events/.

[XML-NS10]
Namespaces in XML 1.0 (Second Edition), T. Bray, D. Hollander, A. Layman, eds. World Wide Web Consortium, 16 August 2006.
This edition of Namespaces in XML 1.0 is http://www.w3.org/TR/2006/REC-xml-names-20060816/.
The latest edition of Namespaces in XML 1.0 is available at http://www.w3.org/TR/xml-names/.

[XML-NS]
Namespaces in XML 1.1 (Second Edition), T. Bray, D. Hollander, A. Layman, R. Tobin, eds. World Wide Web Consortium, 16 August 2006.
This edition of Namespaces in XML 1.1 is http://www.w3.org/TR/2006/REC-xml-names11-20060816/.
The latest edition of Namespaces in XML 1.1 is 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.
This edition of XPointer Framework is http://www.w3.org/TR/2003/REC-xptr-framework-20030325/.
The latest edition of XPointer Framework is available at http://www.w3.org/TR/xptr-framework/.

[XSL]
Extensible Stylesheet Language (XSL) Version 1.1, A. Berglund, ed. World Wide Web Consortium, 05 December 2006.
This edition of XSL 1.1 is http://www.w3.org/TR/2006/REC-xsl11-20061205/.
The latest edition of XSL 1.1 is available at http://www.w3.org/TR/xsl11/.

S.2 Informative references

[ARIA]
Accessible Rich Internet Applications (WAI-ARIA) Version 1.0, M. Cooper, R. Schwerdtfeger, L. Seeman, L. Pappas, eds. World Wide Web Consortium, *work in progress*, 06 August 2008.
This edition of WAI-ARIA is http://www.w3.org/TR/2008/WD-wai-aria-20080806/.
The latest edition of WAI-ARIA is available at http://www.w3.org/TR/wai-aria/.

[CC]
Creative Commons.
Available at http://creativecommons.org/.

[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.
This edition of Charmod Normalization is http://www.w3.org/TR/2005/WD-charmod-norm-20051027/.
The latest edition of Charmod Normalization is available at http://www.w3.org/TR/charmod-norm/.

[CODECS]
WAVE and AVI Codec Registries, Internet Assigned Numbers Authority.
Available at http://www.iana.org/assignments/wave-avi-codec-registry.

[CSS21]
Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, B. Bos, T. Çelik, I. Hickson, H. W. Lie, eds. World Wide Web Consortium, *work in progress*, 19 July 2007.
This edition of CSS 2.1 is http://www.w3.org/TR/2007/CR-CSS21-20070719.
The latest edition of CSS 2.1 is available at http://www.w3.org/TR/CSS21/.

[DCORE]
Dublin Core Metadata Initiative.
Available at http://dublincore.org/.

[DOM3EVENTS]
Document Object Model (DOM) Level 3 Events Specification, B. Höhrmann, P. Le Hégaret, T. Pixley, D. Schepers, eds. World Wide Web Consortium, *work in progress*, 21 December 2007.
This edition of DOM 3 Events is http://www.w3.org/TR/2007/WD-DOM-Level-3-Events-20071221/.
The latest edition of DOM 3 Events is available at http://www.w3.org/TR/DOM-Level-3-Events/.

[ECMA-262]
ECMAScript Language Specification, 3rd Edition, M. Cowlishaw, ed. Ecma International, December 1999.
Available at http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[HTML4]
HTML 4.01 Specification, D. Raggett, A. Le Hors, I. Jacobs. World Wide Web Consortium, 24 December 1999.
This edition of HTML 4 is http://www.w3.org/TR/1999/REC-html401-19991224/.
The latest edition of HTML 4 is available at http://www.w3.org/TR/html4/.

[HTML5]

HTML 5, I. Hickson, D. Hyatt, eds. World Wide Web Consortium, *work in progress*, 10 June 2008.
This edition of HTML 5 is http://www.w3.org/TR/2008/WD-html5-20080610/.
The latest edition of HTML 5 is available at http://www.w3.org/TR/html5/.

[ITS]

Internationalization Tag Set (ITS) 1.0, C. Lieske, F. Sasaki. World Wide Web Consortium, 3 April 2007.
This edition of ITS 1.0 is http://www.w3.org/TR/2007/REC-its-20070403/.
The latest edition of ITS is available at http://www.w3.org/TR/its/.

[JAVA]

The Java Language Specification, J. Gosling, B. Joy and G. Steele. 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. Java Community Process, 21 December 2004.
Available at http://forum.nokia.com/info/sw.nokia.com/id/bcffc227-cacc-4e41-9829-d1979db4dde3/JSR-226-spec-fr-1_0.zip.html.
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.
This edition of MathML 2 is http://www.w3.org/TR/2003/REC-MathML2-20031021/.
The latest edition of MathML 2 is available at http://www.w3.org/TR/MathML2/.

[MF]

Microformats.
Available at http://microformats.org/.

[MIME-RESPECT]

Authoritative Metadata, R. Fielding, I. Jacobs, eds. World Wide Web Consortium, 12 April 2006.
This edition of Authoritative Metadata is http://www.w3.org/2001/tag/doc/mime-respect-20060412.
The latest edition of Authoritative Metadata is available at http://www.w3.org/2001/tag/doc/mime-respect.

[MIMETYPES]

MIME Media Types, Internet Assigned Numbers Authority.
Available at http://www.iana.org/assignments/media-types/.

[NSState]

The Disposition of Names in an XML Namespace, N. Walsh, ed. World Wide Web Consortium, 09 January 2006.
This edition of Namespace State is http://www.w3.org/2001/tag/doc/namespaceState-2006-01-09.
The latest edition of Namespace State is available at http://www.w3.org/2001/tag/doc/namespaceState.

[PROGRESSEVENTS]

Progress Events 1.0, C. McCathieNevile, ed. World Wide Web Consortium, *work in progress*, 21 May 2008.
This edition of ProgressEvents is http://www.w3.org/TR/2008/WD-progress-events-20080521/.
The latest edition of Progress Events is available at http://www.w3.org/TR/progress-events/.

[RFC2361]

WAVE and AVI Codec Registries. E. Fleischman, June 1998.
Available at http://tools.ietf.org/html/rfc2361.

[RDF]

RDF Primer, F. Manola, E. Miller, eds. World Wide Web Consortium, 10 February 2004.
This edition of RDF Primer is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.
The latest edition of RDF Primer is available at http://www.w3.org/TR/rdf-primer/.

[RDFA]

RDFa in XHTML: Syntax and Processing, B. Adida, M. Birbeck, S. McCarron, S. Pemberton, eds. World Wide Web Consortium, *work in progress*, 04 September 2008.
This edition of RDFa Syntax is http://www.w3.org/TR/2008/PR-rdfa-syntax-20080904/.
The latest edition of RDFa Syntax is available at http://www.w3.org/TR/rdfa-syntax/.
An RDFa Primer is also available at http://www.w3.org/TR/xhtml-rdfa-primer/.

[ROLE]

XHTML Role Attribute Module, M. Birbeck, S. McCarron, S. Pemberton, T. V. Raman, R. Schwerdtfeger, eds. World Wide Web Consortium, *work in progress*, 07 April 2008.

This edition of RDFa Syntax is http://www.w3.org/TR/2008/PR-rdfa-syntax-20080904/.
The latest edition of the XHTML Role Attribute Module is available at http://www.w3.org/TR/xhtml-role/.

[SCHEMA2]
XML Schema Part 2: Datatypes Second Edition. P. Biron, A. Malhotra, eds. World Wide Web Consortium, 28 October 2004. (See also *Processing XML 1.1 documents with XML Schema 1.0 processors* [XML11-SCHEMA].)
This edition of XML Schema Part 2 is http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.
The latest edition of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2/.

[SRGB-PROFILES]
sRGB profiles, International Color Consortium. Available at http://www.color.org/srgbprofiles.xalter.

[SVG11]
Scalable Vector Graphics (SVG) 1.1, J. Ferraiolo, 藤沢 淳 (FUJISAWA Jun), D. Jackson, eds. World Wide Web Consortium, 14 January 2003.
This edition of SVG 1.1 is http://www.w3.org/TR/2003/REC-SVG11-20030114/.
The latest edition of SVG 1.1 is available at http://www.w3.org/TR/SVG11/.

[SVG-ACCESS]
Accessibility Features of SVG, C. McCathieNevile, M. Koivunen, eds. World Wide Web Consortium, 07 August 2000.
This edition of Accessibility Features of SVG is http://www.w3.org/TR/2000/NOTE-SVG-access-20000807/.
The latest edition of Accessibility Features of SVG is available at http://www.w3.org/TR/SVG-access/.

[USING-SRGB]
Using the sRGB_v4_ICC_preference.icc profile, International Color Consortium. Available at http://www.color.org/ICC_White_Paper_26_Using_the_V4_sRGB_ICC_profile.pdf.

[WAI]
The Web Accessibility Initiative.
Available at http://www.w3.org/WAI/.

[WCAG]
Web Content Accessibility Guidelines 1.0, W. Chisholm, G. Vanderheiden, I. Jacobs, eds. World Wide Web Consortium, 05 May 1999.
This edition of WCAG 1.0 is http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/.
The latest edition of WCAG 1.0 is available at http://www.w3.org/TR/WAI-WEBCONTENT/.

[WCAG2]
Web Content Accessibility Guidelines 2.0, B. Caldwell, W. Chisholm, J. Slatin, G. Vanderheim, eds. World Wide Web Consortium, *work in progress*, 30 April 2008.
This edition of WCAG 2.0 is http://www.w3.org/TR/2008/CR-WCAG20-20080430/.
The latest edition of WCAG 2.0 is available at http://www.w3.org/TR/WCAG20/.

[WICD]
WICD Core 1.0, T. Mehrvarz, L. Pajunen, J. Quint, D. Appelquist, eds. World Wide Web Consortium, *work in progress*, 18 July 2007.
This edition of WICD Core 1.0 is http://www.w3.org/TR/2007/CR-WICD-20070718/.
The latest edition of WICD Core 1.0 is available at http://www.w3.org/TR/WICD/.

[WINDOW]
Window Object 1.0, I. Davis, M. Stachowiak, eds. World Wide Web Consortium, *work in progress*, 07 April 2006.
This edition of Window 1.0 is http://www.w3.org/TR/2006/WD-Window-20060407/.
The latest edition of Window 1.0 is available at http://www.w3.org/TR/Window/.

[WEBCGM]
WebCGM 1.0 Second Release, D. Cruikshank, J. Gebhardt, L. Henderson, R. Platon, D. Weidenbrueck, eds. World Wide Web Consortium, 17 December 2001.
This edition of WebCGM is http://www.w3.org/TR/2001/REC-WebCGM-20011217/.
The latest edition of WebCGM is available at http://www.w3.org/TR/REC-WebCGM/.

[XBL2]
XML Binding Language (XBL) 2.0, I. Hickson, ed. World Wide Web Consortium, *work in progress* 16 March 2007.
This edition of XBL 2 is http://www.w3.org/TR/2007/CR-xbl-20070316/.
The latest edition of XBL 2 is available at http://www.w3.org/TR/xbl/.

[XHTML]

XHTML™ 1.0: The Extensible HyperText Markup Language (Second Edition). World Wide Web Consortium, 01 August 2002.
This edition of XHTML 1 is http://www.w3.org/TR/2002/REC-xhtml1-20020801/.
The latest edition of XHTML 1 is available at http://www.w3.org/TR/xhtml1/.

[XHTMLVOCAB]

"XHTML Vocabulary namespace". XHTML 2 Working Group, World Wide Web Consortium.
The latest edition is available at http://www.w3.org/1999/xhtml/vocab.

[XI18N-BP]

Best Practices for XML Internationalization, Y. Savourel, J. Kosek, R. Ishida. World Wide Web Consortium, 13 February 2008.
This edition of Best Practices for XML Internationalization is http://www.w3.org/TR/2008/NOTE-xml-i18n-bp-20080213/.
The latest edition of Best Practices for XML Internationalization is available at http://www.w3.org/TR/xml-i18n-bp/.

[XLINK11]

XML Linking Language (XLink) Version 1.1, S. DeRose, E. Maler, D. Orchard, N. Walsh, eds. World Wide Web Consortium, *work in progress*, 31 March 2008.
This edition of XLink 1.1 is http://www.w3.org/TR/2008/WD-xlink11-20080331/.
The latest edition of XLink 1.1 is available at http://www.w3.org/TR/xlink11/.

[XML11-SCHEMA]

Processing XML 1.1 documents with XML Schema 1.0 processors, H. S. Thompson, ed. World Wide Web Consortium, 11 May 2005.
This edition of Processing XML 1.1 with XML Schema 1.0 is http://www.w3.org/TR/2005/NOTE-xml11schema10-20050511/.
The latest edition of Processing XML 1.1 with XML Schema 1.0 is available at http://www.w3.org/TR/xml11schema10/.

[XSLT]

XSL Transformations (XSLT) Version 1.0, J. Clark, ed. World Wide Web Consortium, 16 November 1999.
This edition of XSLT 1.0 is http://www.w3.org/TR/1999/REC-xslt-19991116.
The latest edition of XSLT 1.0 is available at http://www.w3.org/TR/xslt.

[XSLT2]

XSL Transformations (XSLT) Version 2.0, M. Kay, ed. World Wide Web Consortium, 23 January 2007.
This edition of XSLT 2.0 is http://www.w3.org/TR/2007/REC-xslt20-20070123/.
The latest edition of XSLT 2.0 is available at http://www.w3.org/TR/xslt20/.

T Change History

Contents

T.1 Changes over the whole document 434

This appendix is informative.

Listed in this section are all the changes have been made to this Recommendation relative to the previous Proposed Recommendation Working Draft of SVG Tiny 1.2. Cosmetic changes (i.e. correction of typos, changes related to the styling of the document and addition of links) need not be listed.

A detailed difference log of each change is available from our public CVS repository.

T.1 Changes over the whole document

As a high-level summary, some typographical or grammatical errors were corrected during Proposed Recommendation phase. A note regarding future specifications was made to the Status of this Document, and the Abstract was reworded slightly for clarity and correctness. No substantive changes were made from Proposed Recommendation to Recommendation phase.