


# AFFIDAVIT OF NATHANIEL E FRANK-WHITE

1. I am a Records Request Processor at the Internet Archive. I make this declaration of my own personal knowledge.

2. The Internet Archive is a website that provides access to a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public. The Internet Archive has partnered with and receives support from various institutions, including the Library of Congress.

3. The Internet Archive has created a service known as the Wayback Machine. The Wayback Machine makes it possible to browse more than 450 billion pages stored in the Internet Archive's web archive. Visitors to the Wayback Machine can search archives by URL (i.e., a website address). If archived records for a URL are available, the visitor will be presented with a display of available dates. The visitor may select one of those dates, and begin browsing an archived version of the Web. Links on archived files in the Wayback Machine point to other archived files (whether HTML pages or other file types), if any are found for the URL indicated by a given link. For instance, the Wayback Machine is designed such that when a visitor clicks on a hyperlink on an archived page that points to another URL, the visitor will be served the archived file found for the hyperlink's URL with the closest available date to the initial file containing the hyperlink.

4. The archived data made viewable and browsable by the Wayback Machine is obtained by use of web archiving software that automatically stores copies of files available via the Internet, each file preserved as it existed at a particular point in time.

5. The Internet Archive assigns a URL on its site to the archived files in the format http://web.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] aka an "extended URL". Thus, the extended URL http://web.archive.org/web/19970126045828/http://www.archive.org/ would be the URL for the record of the Internet Archive home page HTML file (http://www.archive.org/) archived on January 26, 1997 at 4:58 a.m. and 28 seconds (1997/01/26 at 04:58:28). The date indicated by an extended URL applies to a preserved instance of a file for a given URL, but not necessarily to any other files linked therein. Thus, in the case of a page constituted by a primary HTML file and other separate files (e.g., files with images, audio, multimedia, design elements, or other embedded content) linked within that primary HTML file, the primary HTML file and the other files will each have their own respective extended URLs and may not have been archived on the same dates.

6. Attached hereto as Exhibit A are true and accurate copies of screenshots of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each printout.

INTERNET ARCHIVE
archive.org

7. I declare under penalty of perjury that the foregoing is true and correct.

DATE: 09/10/2023

Nathaniel Frank-White
Nathaniel E Frank-White

**Please see attached All Purpose Jurat form for additional Notary Events**



fuboTV Media Inc.
Exhibit 1011
Page 002

# JURAT ATTACHMENT

A notary public or other officer completing this certificate verifies only the identity of the individual who signed the document to which this certificate is attached, and not the truthfulness, accuracy, or validity of that document.

STATE OF Texas }

COUNTY OF Tarrant }

The foregoing instrument was subscribed and sworn before me this date of 09/10/2023, by Nathaniel Frank-White

This notarial act was an online notarization.




(Notary Seal)

Notary's Signature _____

Registration No.: 133645929

Commission Expiration Date: March 15, 2026

# EXHIBIT A

https://web.archive.org/web/20011109111837/http://www.w3.org/TR/2001/REC-smil20-20010807/smil20.html

http://www.w3.org/TR/2001/REC-smil20-20010807/smil20.html
Go
61 captures
9 Nov 2001 - 7 Feb 202
OCT NOV OCT
09
2001 2002
About this capture

W3C Recommend

# Synchronized Multimedia Integration Language (SMIL 2.0)

## W3C Recommendation 07 August 2001

**This version:**
http://www.w3.org/TR/2001/REC-smil20-20010807/
(Other formats: single HTML file, zip archive)
**Latest version:**
http://www.w3.org/TR/smil20
**Previous version:**
http://www.w3.org/TR/2001/PR-smil20-20010605/
**Editors:**
Jeff Ayars (RealNetworks), Dick Bulterman (Oratrix), Aaron Cohen (Intel), Ken Day (Macromedia), Erik Hodge (RealNetworks), Philipp Hoschka (W3C), Eric Hyche (RealNetworks), Muriel Jourdan (INRIA), Michelle Kim (IBM), Kenichi Kubota (Panasonic), Rob Lanphier (RealNetworks), Nabil Layaïda (INRIA), Thierry Michel (W3C), Debbie Newman (Microsoft), Jacco van Ossenbruggen (CWI), Lloyd Rutledge (CWI), Bridie Saccocio (RealNetworks), Patrick Schmitz (Microsoft), Warner ten Kate (Philips).

Copyright ©2001 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

## Abstract

This document specifies the second version of the Synchronized Multimedia Integration Language (SMIL, pronounced "smile"). SMIL 2.0 has the following two design goals:

- Define an XML-based language that allows authors to write interactive multimedia presentations. Using SMIL 2.0, an author can describe the temporal behavior of a multimedia presentation, associate hyperlinks with media objects and describe the layout of the presentation on a screen.
- Allow reusing of SMIL syntax and semantics in other XML-based languages, in particular those who need to represent timing and synchronization. For example, SMIL 2.0 components are used for integrating timing into XHTML [XHTML10] and into SVG [SVG].

## Status of this document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.*

This is the W3C Recommendation of the Synchronized Multimedia Integration Language (SMIL) 2.0.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference 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.

This document has been produced by the SYMM Working Group *[members only]* as part of the W3C Synchronized Multimedia Activity. The goals of the SYMM Working Group are discussed in the SYMM Working Group charter *[members only]*.

The SYMM Working Group considers that all features in the SMIL 2.0 Recommendation have been implemented at least twice in an interoperable way. The SYMM Working Group Charter defines this as the implementations having been developed independently by different organizations and each test in the public SMIL 2.0 test suite has at least two passing implementations. The Implementation results are publicly released and are intended solely to be used as proof of SMIL 2.0 implementability. It is only a snap shot of the actual implementation behaviors at one moment of time, as these implementations may not be immediately available to the public. The interoperability data is not intended to be used for assessing or grading the performance of any individual implementation.

There are patent disclosures and license commitments associated with the SMIL 2.0 Recommendation, these may be found on the SYMM Patent Statement page in conformance with W3C policy.

This version of this document incorporates some editorial changes from earlier versions.

Please report errors in this document to www-smil@w3.org - (public archives).
The list of known errors in this specification is available at http://www.w3.org/2001/07/REC-SMIL20-20010731-errata.

The authors of this document are the SYMM Working Group members. Different parts of the document have different editors.
The W3C staff contact for work on SMIL is Thierry MICHEL.

The English version of this specification is the only normative version. Information about translations of this document is available at http://www.w3.org/AudioVideo/SMIL/translations.

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

Quick Table of Contents

1. About SMIL 2.0
2. The SMIL 2.0 Modules
3. The SMIL 2.0 Animation Modules
4. The SMIL 2.0 Content Control Modules
5. The SMIL 2.0 Layout Modules
6. The SMIL 2.0 Linking Modules

**7. The SMIL 2.0 Media Object Modules**
**8. The SMIL 2.0 Metainformation Module**
**9. The SMIL 2.0 Structure Module**
**10. The SMIL 2.0 Timing and Synchronization Module**
**11. The SMIL 2.0 Time Manipulations Module**
**12. The SMIL 2.0 Transition Effects Module**
**13. SMIL 2.0 Language Profile**
**14. SMIL 2.0 Basic Profile and Scalability Framework**
**Appendix A. SMIL 2.0 DTDs**
**Appendix B. SMIL 2.0 Schemas**
**Appendix C. Index of Elements**
**Appendix D. Index of Attributes**
**Appendix E. References**

## Full Table of Contents

**1. About SMIL 2.0**
1.1 Introduction
1.1.1 Relation to SMIL 1.0
1.1.2 Content of this Recommendation
1.2 Acknowledgements
**2. The SMIL 2.0 Modules**
2.1 Introduction
2.1.1 Modularization and Profiling
2.2 SMIL 2.0 Modules
2.3 Identifiers for SMIL 2.0 Modules and Language Profiles
2.3.1 The SMIL Mime Type
2.3.2 XML Namespace Identifier for the SMIL 2.0 Modules
2.3.3 Identifiers for SMIL 2.0 Modules and Features
2.4 SMIL Conformance
2.4.1 SMIL Host Language Conformance
2.4.2 SMIL Integration Set Conformance
2.4.3 Requirements on Identifiers for SMIL Host Language Conformant Language Profiles
2.4.4 Error Handling in SMIL Host Language Conformant Documents
2.4.5 Handling of Syntax errors in Attribute Values
2.5 Creating a DTD for a Language Profile
**3. The SMIL 2.0 Animation Modules**
3.1 Introduction
3.2 Animation Model
3.2.1 The simple animation function f(t)
3.2.2 Summary of symbols used in the semantic descriptions
3.2.3 The animation sandwich model
3.2.4 Animation elements as "continuous media"
3.2.5 The animation effect function F(t,u)
*Repeating animations*
*Examples*
*Controlling behavior of repeating animation - Cumulative animation*
*Freezing animations*
3.2.6 Additive animation
*Additive and Cumulative animation*
3.2.7 Restarting animations
3.2.8 Animation function value details
*Interpolation and indefinite simple durations*
3.3 Overview of the SMIL 2.0 BasicAnimation Module
3.4 SMIL 2.0 BasicAnimation Module Common Attributes
3.4.1 Specifying the animation target
*The target attribute*
3.4.2 Specifying the simple animation function f(t)
*Interpolation modes illustrated*
*Examples of calcMode*
3.4.3 Specifying the animation effect function F(t,u)
3.4.4 Simple animation functions specified by from, to and by
*To animation*
3.5 SMIL 2.0 BasicAnimation Elements
3.5.1 The animate element
3.5.2 The set element
3.5.3 The animateMotion element
3.5.4 The animateColor element
3.6 SMIL 2.0 BasicAnimation Module Details
3.6.1 BasicAnimation integration requirements
*Required definitions and constraints on animation targets*
*Specifying the target element*
*Target attribute issues*
*Integrating animateMotion functionality*
3.6.2 Document type definition (DTD) for the BasicAnimation module
3.7 Overview of the SMIL 2.0 SplineAnimation Module
3.7.1 SMIL 2.0 SplineAnimation Module Attributes
*Spline animation function calculation mode*
*Examples of advanced uses of calcMode*
*Interpolation with keySplines*
3.7.2 SMIL 2.0 SplineAnimation Module Elements
3.7.3 The spline animate element

3.7.4 The spline animateMotion element
3.7.5 The spline animateColor element
3.8 SMIL 2.0 SplineAnimation Module Details
3.8.1 SplineAnimation integration requirements
3.8.2 Document type definition (DTD) for the SplineAnimation module
3.9 Common Animation Integration Requirements
3.9.1 Integration requirements
*Extending Animation*
*Handling syntax errors*
*Error handling semantics*
**4. The SMIL 2.0 Content Control Modules**
4.1 Introduction
4.2 The SMIL 2.0 BasicContentControl Module
4.2.1 SMIL 2.0 BasicContentControl Module Overview
*Predefined System Test Attributes*
*The switch element*
*System Test Attribute In-Line Use*
*Examples of Switch and Test Attribute Use*
4.2.2 Elements and Attributes
*The switch element*
*Element attributes*
*Element content*
*Predefined Test Attributes*
4.2.3 Integration Requirements for the BasicContentControl Module
4.2.4 Document Type Definition (DTD) for the BasicContentControl Module
4.3 The SMIL 2.0 CustomTestAttributes Module
4.3.1 SMIL 2.0 CustomTestAttributes Module Overview
*Example Use*
*Rules for Setting Values*
4.3.2 Elements and Attributes
*The customAttributes element*
*Element attributes*
*Element content*
*The customTest element*
*Element attributes*
*Element content*
*The customTest attribute*
4.3.3 Integration Requirements for the CustomTestAttribute Module
4.3.4 Document Type Definition (DTD) for the CustomTestAttribute Module
4.4 The SMIL 2.0 PrefetchControl Module
*Examples*
4.4.2 Elements and Attributes
*The prefetch element*
*Element attributes*
*Attribute value syntax*
4.4.3 Integration Requirements for the PrefetchControl Module
4.4.4 Document Type Definition (DTD) for the PrefetchControl Module
4.5 The SMIL 2.0 SkipContentControl Module
4.5.1 SMIL 2.0 SkipContentControl Module Overview
4.5.2 Elements and Attributes
*Element definition*
*The skip-content attribute*
4.5.3 Integration Requirements for the SkipContentControl Module
**5. The SMIL 2.0 Layout Modules**
5.1 Introduction
5.2 Overview of the SMIL 2.0 BasicLayout Module
5.3 SMIL 2.0 BasicLayout Module Syntax and Semantics
5.3.1 Elements and Attributes
*The layout element*
*Element content*
*The region element*
*The root-layout element*
*The region attribute*
5.3.2 SMIL BasicLayout Module Details
5.3.3 Document Type Definition (DTD) for the BasicLayout Module
5.4 Overview of the SMIL 2.0 AudioLayout Module
5.4.1 Integration Requirements for the AudioLayout Module
5.4.2 Audio Volume Control
5.5 SMIL 2.0 AudioLayout Module Syntax and Semantics
5.5.1 Elements and Attributes
*The region element*
5.6 Overview of the SMIL 2.0 MultiWindowLayout Module
5.6.1 Integration Requirements for the MultiWindowLayout Module
5.7 SMIL 2.0 MultiWindowLayout Module Syntax and Semantics
5.7.1 Elements and Attributes
*The topLayout element*
*Element content*
*The layout element*
*Element content*
5.7.2 MultiWindowLayout Module Events
*Integration Requirements for the MultiWindowLayout Module Events*
5.7.3 Document Type Definition (DTD) for the MultiWindowLayout Module

5.8 Overview of the SMIL 2.0 HierarchicalLayout Module
5.8.1 Integration Requirements for the HierarchicalLayout Module
5.9 New Features in the SMIL 2.0 HierarchicalLayout Module
5.9.1 Hierarchical Region Layout
5.9.2 Sub-Region Layout
5.9.3 Media Object fit, z-index, and backgroundColor
5.9.4 Registration Points
5.10 SMIL 2.0 HierarchicalLayout Syntax and Semantics
5.10.1 Elements and attributes
*The layout element*
*The region element*
*The regPoint element*
5.10.2 SMIL HierarchicalLayout Positioning and Presentation Attributes
*Media specific background color*
*Sub-region Positioning Attributes*
*Registration point attributes*
*Region fit override*
*Region z-index override*
**6. The SMIL 2.0 Linking Modules**
6.1 Introduction
6.2 Relationship with Other XML Linking-related Formats
6.2.1 Relationship with XPointer
6.2.2 Relationship with XLink
6.2.4 Relationship with XHTML
6.3 Linking into SMIL 2.0 Documents
6.3.1 Handling of Links in Embedded Documents
6.3.2 Error Handling
6.4 SMIL 2.0 LinkingAttributes Module
6.5 SMIL 2.0 BasicLinking Module
6.5.1 The a Element
6.5.2 The area Element
6.6 SMIL 2.0 ObjectLinking Module
6.6.1 The fragment Attribute
**7. The SMIL 2.0 Media Object Modules**
7.1 Introduction
7.2 Definitions
7.3 SMIL BasicMedia Module
7.3.1 Media Object Elements - ref, animation, audio, img, text, textstream and video
7.3.2 Integration Requirements
7.4 SMIL MediaParam Module
7.4.1 Media object initialization: the param element
7.4.2 Element Attributes for All Media Objects
7.4.3 Integration Requirements
7.5 SMIL MediaClipping Module
7.5.1 MediaClipping Attributes
7.6 SMIL MediaClipMarkers Module
7.6.1 MediaClipMarkers Attribute Extensions
7.7 SMIL BrushMedia Module
7.7.1 The brush element
7.7.2 Integration Requirements
7.8 SMIL MediaAccessibility Module
7.8.1 MediaAccessibility Attributes
7.9 SMIL MediaDescription Module
7.9.1 MediaDescription Attributes
7.10 Appendices
7.10.1 Appendix A: Changes to SMIL 1.0 Media Object Attributes
*clipBegin, clipEnd, clip-begin, clip-end*
*Handling of new clipBegin/clipEnd syntax in SMIL 1.0 software*
*New Accessibility Attributes*
*New Advanced Media Attributes*
7.10.2 Appendix B: Changes to SMIL 1.0 Media Object Elements
*New child elements for media objects*
*The param element*
*The brush element*
**8. The SMIL 2.0 Metainformation Module**
8.1 Introduction
8.2 Overview of the SMIL 2.0 Metainformation module
8.2.1 Compatibility with SMIL 1.0
8.2.2 Extensions to SMIL 1.0
8.2.3 Multiple description schemes
8.3 SMIL 2.0 Metainformation Module Syntax and Semantics
8.3.1 The meta element
8.3.2 The metadata element
8.4 An Example
**9. The SMIL 2.0 Structure Module**
9.1 Introduction
9.2 The SMIL 2.0 Structure Module Syntax and Semantics
9.2.1 Elements and attributes
*The smil element*
*Element attributes*
*Element content*
*The head element*
*Element attributes*
*Element content*

The body element
Element attributes
Element content
9.3 Integrating the SMIL Structure Module
**10. The SMIL 2.0 Timing and Synchronization Module**
10.2 Overview of SMIL timing
10.3 Language definition
10.3.1 Attributes
The begin and dur attributes: basic timing support
Begin value semantics
Handling negative offsets for begin
Negative begin delays
Dur value semantics
Examples
The end attribute: controlling active duration
Handling negative offsets for end
The min and max attributes: more control over the active duration
The min attribute and negative begin times
Timing attribute value grammars
Begin values
End values
Parsing timing specifiers
Clock values
Offset values
SMIL 1.0 begin and end values
ID-Reference values
Syncbase values
Event values
Repeat values
Accesskey values
Media marker values
Wallclock-sync values
The endsync attribute
The repeatCount, repeatDur, and repeat attributes: repeating elements
Examples
The min attribute and restart:
SMIL 1.0 repeat (deprecated)
The fill attribute: extending an element
The fillDefault attribute
The Event sensitivity and fill
The restart attribute
Using restart for toggle activation
Controlling the default behavior of restart
Resetting element state
The syncBehavior, syncTolerance, and syncMaster attributes: controlling runtime synchronization
Controlling the default behavior
The accumulated synchronization offset
Attributes for timing integration: timeContainer and timeAction
The timeContainer attribute
The timeAction attribute
Examples:
10.3.2 Elements
The par element
Implicit duration of par
The seq element
Implicit duration of seq containers
The excl element
Implicit duration of excl containers
The priorityClass element
Examples using excl and priorityClass
Pause queue semantics
Implicit duration of media element time containers
Examples:
Media time containers of other types
10.3.3 Semantics of the Timing Model
Resolving times
Definite times
Defining the simple duration
repeatCount and unresolved simple duration
Computing the active duration
Active duration arithmetic rules
Active duration algorithm
Intermediate Active Duration Computation
Paused elements and the active duration
Evaluation of begin and end time lists
The instance times lists
Principles for building and pruning intervals
Element life-cycle
Interaction with restart semantics
Cyclic dependencies in the timegraph
Detecting Cycles
Examples
Timing and real-world clock times
Interval timing

Background rationale
Implications for the time model
Event sensitivity
User event sensitivity and timing
Link Activation compared to Event activation
Converting between local and global times
Element active time calculation
Element simple time calculation
Converting wall-clock values
Converting from event time to element time
Converting from element time to element time
Time conversions and sampling the time graph
Hyperlinks and timing
Implications of beginElement() and hyperlinking for seq and excl time containers
Propagating changes to times
Deferred elements and propagating changes to begin
Restart and propagating changes to times
Time container duration
Time container constraints on child durations
The min attribute and time container constraints on child durations
Time container constraints on sync-arcs and events
Specifics for sync-arcs
Specifics for event-based timing
Behavior of 0 duration elements
10.3.4 Clarifications and surprising results
10.4 Integrating SMIL Timing and Synchronization into a host language
10.4.1 Required host language definitions
10.4.2 Required definitions and constraints on element timing
Supported events for event-base timing
10.4.3 Error handling semantics
10.5 Document object model support
10.5.1 Element and attribute manipulation, mutation and constraints
10.5.2 Events and event model
Example 1
Example 2
Example 3
10.5.3 Reserved DOM methods
10.6 Glossary
10.6.1 General concepts
Synchronization relationship
Time graph
Descriptive terms for times
Local time and global time
Linear and Non-linear media
Scheduled timing
document begin
document end
document duration
Events and interactive timing
Syncbases
Sync arcs
Clocks
UTC: Coordinated Universal Time
Hyperlinking and timing
Activation
Discrete and continuous Media
10.6.2 Timing concepts
Time containers
Content/Media elements
Basic markup
Simple and active durations
Hard and soft sync
Pruning and cutting off an interval
10.7 Appendix A: SMIL Timing and Synchronization modules
10.8 Appendix B: Annotated examples
10.8.1 Example 1: Simple timing within a Parallel time container
10.8.2 Example 2: Simple timing within a Sequence time container
10.8.3 Example 3: excl time container with child timing variants
10.8.4 Example 4: default duration of discrete media
10.8.5 Example 5: end specifies end of active dur, not end of simple dur
10.8.6 Example 6: DOM-initiated timing
10.9 Appendix C: Differences from SMIL 1.0
10.10 Appendix D: Unifying event based and scheduled timing
10.10.1 Background
10.10.2 Modeling interactive, event-based content in SMIL
**11. The SMIL 2.0 Time Manipulations Module**
11.1 Introduction
11.1.1 Background
11.1.2 Overview of support
11.1.3 Attribute syntax
The accelerate and decelerate attributes
Examples
The autoReverse attribute
The speed attribute
Examples

11.1.4 Details of timing model arithmetic
*Timing and real-world clock times*
*Computing the element run-rate*
*Converting document time to element time*
*Filtered active time calculation*
*Filtered simple time calculation*
*Converting element time to document time*
*Computing the net cascaded speed for an element*
11.1.5 Media fallback semantics
*Ideal model*
*Fallbacks for time filters on a media element*
*Authoring considerations for the fallback semantics*
*Implications of time manipulations on time containers*
*Handling negative speeds on time containers*
**12. The SMIL 2.0 Transition Effects Module**
12.1 Introduction
12.2 Transition Model
12.3 Transition Taxonomy
12.3.1 Default Transition Subtypes
12.3.2 Required Transitions
12.4 BasicTransitions Module
12.4.1 The **transition** element
*Examples of the **transition** element*
12.4.2 The param element
12.4.3 The **transIn** and **transOut** attributes
*Rules For Applying Transitions to Media Elements*
*Use of fill="transition"*
*Slideshow example with transitions*
*Exclusive children and fill="transition"*
12.4.4 Handling Parameter Errors
12.4.5 Transition Parsing Rules
12.4.6 Extending The Set Of Transitions
12.5 InlineTransitions Module
12.5.1 The **transitionFilter** element
12.5.2 The param element
12.6 TransitionModifiers Module
12.6.1 Horizontal and Vertical Pattern Repeat
12.7 Integration
12.8 Appendix: Taxonomy Tables
12.8.1 Table 1: The Taxonomy Table
12.8.2 Table 2: SMPTE Edge Wipes
12.8.3 Table 3: SMPTE Iris Wipes
12.8.4 Table 4: SMPTE Clock Wipes
12.8.5 Table 5: SMPTE Matrix Wipes
**13. SMIL 2.0 Language Profile**
13.1 Abstract
13.2 SMIL 2.0 Profile
13.3 Normative Definition of the SMIL 2.0 Language Profile
13.3.1 Document Conformance
13.3.2 SMIL 2.0 Language Conformance
*Conforming SMIL 2.0 Documents*
*Conforming SMIL 2.0 Language User Agents*
13.3.3 The SMIL 2.0 Language Profile
13.3.4 Animation Module
13.3.5 Content Control Modules
13.3.6 Layout Modules
13.3.7 Linking Modules
13.3.8 Media Object Modules
*Widely Supported MIME Types*
*Media Object Integration Requirements*
13.3.9 Metainformation Module
13.3.10 Structure Module
13.3.11 Timing and Synchronization Modules
*Supported Event Symbols*
*Event semantics*
*Order of raising of simultaneous events:*
*Extending the set of supported events*
***Integration definitions***
13.4 Extending the SMIL 2.0 Language
13.5 Appendix A: SMIL 20 Document Type Definition
13.6 Appendix B: SMIL 20 XML Schema
**14. SMIL 2.0 Basic Profile and Scalability Framework**
14.1 Abstract
14.2 Introduction
14.3 SMIL 2.0 Basic Profile
14.3.1 Definition
*XML Namespace Declarations and Internationalization*
14.3.2 Conformance of SMIL 2.0 Basic Documents
14.3.3 Conformance of SMIL 2.0 Basic User Agents
14.4 Scalable Profiles
14.4.1 Definition
14.4.2 SMIL 2.0 Document Scalability Guidelines
14.4.3 Conformance of Scalable SMIL 2.0 Documents

14.4.3 Conformance of Scalable SMIL 2.0 Documents
14.4.4 Conformance of Scalable User Agents
**Appendix A. SMIL 2.0 DTDs**
A.1 SMIL 2.0 Module DTDs:
A.1.1 The SMIL Animation Module
A.1.2 The SMIL Content Control Module
A.1.3 The SMIL Layout Module
A.1.4 The SMIL Linking Module
A.1.5 The SMIL Media Object Module
A.1.6 The SMIL Metainformation Module
A.1.7 The SMIL Structure Module
A.1.8 The SMIL Timing Module
A.1.9 The SMIL Transition Module
A.2 SMIL 2.0 Language Profile:
A.2.1 SMIL 2.0 Language Profile document model
A.2.2 SMIL 2.0 Language Profile DTD driver
A.3 General modularization framework:
A.3.1 SMIL 2.0 common datatypes
A.3.2 SMIL 2.0 common attributes
A.3.3 SMIL 2.0 qualified names module
A.3.4 SMIL 2.0 framework module
**Appendix B. SMIL 2.0 Schemas**
B.1 XML Schema for the SMIL 2.0 functionalities:
B.1.1 SMIL 2.0 Modules
B.1.2 The SMIL Animation Module
B.1.3 The SMIL Content Control Module
B.1.4 The SMIL Layout Module
B.1.5 The SMIL Linking Module
B.1.6 The SMIL Media Object Module
B.1.7 The SMIL Meta Module
B.1.8 The SMIL Structure Module
B.1.9 The SMIL Time manipulation
B.1.10 The SMIL Timing Module
B.1.11 The SMIL Transition Module
B.2 XML Schema for the SMIL 2.0 Language:
B.2.1 SMIL 2.0 Language Profile
B.3 XML Schema for the SMIL 2.0 modules:
B.3.1 SMIL 2.0 AccessKeyTiming
B.3.2 SMIL 2.0 AudioLayout
B.3.3 SMIL 2.0 BasicAnimation
B.3.4 SMIL 2.0 BasicContentControl
B.3.5 SMIL 2.0 BasicInlineTiming
B.3.6 SMIL 2.0 BasicLayout
B.3.7 SMIL 2.0 BasicLinking
B.3.8 SMIL 2.0 BasicMedia
B.3.9 SMIL 2.0 BasicTimeContainers
B.3.10 SMIL 2.0 BasicTransitions
B.3.11 SMIL 2.0 BrushMedia
B.3.12 SMIL 2.0 CustomTestAttributes
B.3.13 SMIL 2.0 EventTiming
B.3.14 SMIL 2.0 ExclTimeContainers
B.3.15 SMIL 2.0 FillDefault
B.3.17 SMIL 2.0 HostLanguage
B.3.18 SMIL 2.0 InlineTransitions
B.3.19 SMIL 2.0 IntegrationSet
B.3.20 SMIL 2.0 LinkingAttributes
B.3.21 SMIL 2.0 MediaAccessibility
B.3.22 SMIL 2.0 MediaClipMarkers
B.3.23 SMIL 2.0 MediaClipping
B.3.24 SMIL 2.0 MediaDescription
B.3.25 SMIL 2.0 MediaMarkerTiming
B.3.26 SMIL 2.0 MediaParam
B.3.27 SMIL 2.0 Metainformation
B.3.28 SMIL 2.0 MinMaxTiming
B.3.29 SMIL 2.0 MultiArcTiming
B.3.30 SMIL 2.0 MultiWindowLayout
B.3.31 SMIL 2.0 ObjectLinking
B.3.32 SMIL 2.0 PrefetchControl
B.3.33 SMIL 2.0 RepeatTiming
B.3.34 SMIL 2.0 RepeatValueTiming
B.3.35 SMIL 2.0 RestartDefault
B.3.36 SMIL 2.0 RestartTiming
B.3.37 SMIL 2.0 SkipContentControl
B.3.38 SMIL 2.0 SplineAnimation
B.3.39 SMIL 2.0 Structure
B.3.40 SMIL 2.0 SyncBehavior
B.3.41 SMIL 2.0 SyncBehaviorDefault
B.3.42 SMIL 2.0 SyncMaster
B.3.43 SMIL 2.0 SyncbaseTiming
B.3.44 SMIL 2.0 TimeContainerAttributes
B.3.45 SMIL 2.0 TimeManipulations
B.3.46 SMIL 2.0 TransitionModifiers
B.3.47 SMIL 2.0 WallclockTiming
B.3.48 SMIL 2.0 utility

B.3.50 SMIL 2.0 xml-mod
**Appendix C. Index of Elements**
**Appendix D. Index of Attributes**
**Appendix E. References**

# 1. About SMIL 2.0

***Editors***
Aaron Cohen (aaron.m.cohen@intel.com), Intel
Thierry Michel (tmichel@w3.org), W3C

## 1.1 Introduction

This document specifies the second version of the *Synchronized Multimedia Integration Language* (SMIL, pronounced "smile"). SMIL 2.0 has the following two design goals:

- Define an XML-based language that allows authors to write interactive multimedia presentations. Using SMIL 2.0, an author can describe the temporal behavior of a multimedia presentation, associate hyperlinks with media objects and describe the layout of the presentation on a screen.
- Allow reusing of SMIL 2.0 syntax and semantics in other XML-based languages, in particular those who need to represent timing and synchronization. For example, SMIL 2.0 components are used for integrating timing into XHTML [XHTML10] and into SVG [SVG].

SMIL 2.0 is defined as a set of markup modules, which define the semantics and an XML syntax for certain areas of SMIL functionality.

### 1.1.1 Relation to SMIL 1.0

SMIL 2.0 deprecates a small amount of SMIL 1.0 syntax in favor of more DOM friendly syntax. Most notable is the change from hyphenated attribute names to mixed case (camel case) attribute names, e.g., clipBegin is introduced in favor of clip-begin. The SMIL 2.0 modules do not require support for these SMIL 1.0 attributes so that integration applications are not burdened with them. SMIL document players, those applications that support playback of "application/smil" documents, and host language conformant document profiles must support the deprecated SMIL 1.0 attribute names as well as the new SMIL 2.0 names.

### 1.1.2 Content of this Recommendation

This Recommendation is structured as a set of sections, each defining one or more modules:

- Section 2 is an overview of SMIL 2.0 modularization and the individual modules, and presents conformance criteria.
- Section 3 defines the declarative SMIL 2.0 Animation Modules.
- Section 4 presents the SMIL 2.0 Content Control Modules.
- Section 5 describes the SMIL 2.0 Layout Modules.
- Section 6 defines the SMIL 2.0 Linking Modules.
- Section 7 presents the SMIL 2.0 Media Object Modules.
- Section 8 defines the SMIL 2.0 Metainformation Module.
- Section 9 defines the SMIL 2.0 Structure Module.
- Section 10 defines the SMIL 2.0 Timing and Synchronization Modules.
- Section 11 defines the SMIL 2.0 Time Manipulations Module.
- Section 12 presents the SMIL 2.0 Transition effects Modules.

This Recommendation also defines two profiles that are built using the above SMIL 2.0 modules:

- Section 13 defines the SMIL 2.0 Language Profile.
- Section 14 describes the SMIL 2.0 Basic Profile and Scalability Framework.

The XHTML+SMIL Profile that appeared in Working Drafts of this Recommendation is published separately, and is not part of the SMIL 2.0 Recommendation. However, one of the implementations used to validate SMIL 2.0 was based on the XHTML+SMIL Profile. All XHTML+SMIL examples in this Recommendation conform to the Profile as of the Working Draft of 06 August 2001. The latest version of this document is also available, see [XHTML+SMIL].

## 1.2 Acknowledgements

This document has been prepared by the Synchronized Multimedia Working Group (SYMM-WG) of the World Wide Web Consortium. The WG included the following individuals:

Hanan Rosenthal, Canon - Jin Yu, Compaq - Pietro Marchisio, CSELT - Lynda Hardman, CWI - Jacco van Ossenbruggen, CWI - Lloyd Rutledge, CWI - Simon Wan, Vanco Telecom - Yoo Wagoleni, Gateway (In Kiss Expo) - Masayuki Ayama, Glocom - Noboru Kamimura, Glocom, Michelle Y. Kim, IBM - Steve Wood, IBM - Jeff Boston, IBM - Nabil Layaïda, INRIA - Muriel Jourdan, INRIA - Aaron Cohen, Intel - Wayne Carr, Intel - Marcel Wong, Ericsson - Ken Day, Macromedia - Daniel Weber, Panasonic - Patrick Schmitz, Microsoft - Debbie Newman, Microsoft - Pablo Fernicola, Microsoft - Aaron Patterson, Microsoft - Kevin Gallo, Microsoft - Paul David, Microsoft - Don Cone, Netscape/AOL - Wo Chang, NIST - Didier Chanut, Nokia - Antti Koivisto, Nokia - Roberto Castagno, Nokia - Jack Jansen, Oratrix - Sjoerd Mullender, Oratrix - Dick Bulterman, Oratrix - Kenichi Kubota, Panasonic - Warner ten Kate, Philips - Ramon Clout, Philips - Jeff Ayars, RealNetworks - Erik Hodge, RealNetworks - Rob Lanphier, RealNetworks - Bridie Saccocio, RealNetworks - Eric Hyche, RealNetworks - Robin Haglund, RealNetworks - Geoff Freed, WGBH - Philipp Hoschka, W3C - Philippe Le Hégaret, W3C - Thierry Michel, W3C.

# 2. The SMIL 2.0 Modules

***Editors:***
Warner ten Kate (warner.ten.kate@philips.com), (Philips Electronics)
Aaron Cohen (aaron.m.cohen@intel.com), (Intel)
Philipp Hoschka (ph@w3.org), (W3C)

## 2.1 Introduction

***This section is informative.***

Since the publication of SMIL 1.0 [SMIL10], interest in the integration of SMIL concepts with the HTML, the HyperText Markup Language [HTML4], and other XML languages, has grown. Likewise, the W3C HTML Working Group has specified XHTML, the Extensible HyperText Markup Language [XHTML10], in preparation to subset, extend, and integrate it with other languages. The strategy considered for integrating respective functionality with other XML-based languages is based on the concepts of *modularization* and *profiling* [SMIL-MOD], [XMOD].

*Modularization* is an approach in which markup functionality is specified as a set of modules that contain semantically-related XML elements, attributes, and attribute values. *Profiling* is the creation of an XML-based language through combining these modules, in order to provide the functionality required by a particular application.

Profiling introduces the ability to tailor an XML-based language to specific needs, e.g. to optimize presentation and interaction for the client's capabilities. Profiling also adds the ability for integrating functionality from other markup languages, releasing the language designer from specifying that functionality. Moreover, it provides for consistency in markup through the use of the same model to incorporate a function. Identical constructs ease authoring, while at the user agent side there is a potential for re-use of code. For example, a scheduler supporting SMIL timing and synchronization functionality could be used for SMIL documents, XHTML+SMIL documents, and SVG documents.

Modularization enables language designers to specify dedicated markup intended for integration with other, existing, language profiles. Examples of specifications intended for such integration are MathML and XForms [MathML], [XFORMS].

Modularization and profiling use the extensibility properties of XML, and related technology like XML namespaces and XML Schema [XML10], [XML-NS], [XSCHEMA].

This part of the SMIL 2.0 specification describes the framework on which SMIL modularization and profiling is based, and specifies the SMIL 2.0 Modules, their identifiers, and the requirements for conformance within this framework.

### 2.1.1 Modularization and Profiling

***This section is informative.***

The modularization approach used in this specification derives from that set forth in XHTML Modularization [XMOD]. The framework on which SMIL modularization and profiling is based, is informally described here.

A *Module* is a collection of semantically-related XML elements, attributes, and attribute values that represents a unit of functionality. Modules are defined in coherent sets. This coherency is expressed in that the elements of these modules are associated with the same namespace.

A *Language Profile* is a combination of modules. Modules are *atomic*, i.e. they cannot be subset when included in a language profile. Furthermore, a module specification may include a set of integration requirements, to which language profiles that include the module must comply.

Commonly, there is a main language profile that incorporates nearly all the modules associated with a single namespace. For example, the SMIL 2.0 language profile uses most of the SMIL 2.0 modules. Usually, the same name is used to loosely reference both - "SMIL 2.0" in the example. Also, the name "profile" is used to mean "language profile".

Other language profiles can be specified that are subsets of the larger one, or that incorporate a mixture of modules associated with different namespaces. SMIL 2.0 Basic is an example of the first, XHTML+SMIL of the latter.

A special module in a language profile is the so-called *Structure Module*, in that it contains the root element of the language profile, e.g. <smil> or <html>. Any language profile that incorporates modules associated with a single namespace will include the Structure module associated with that namespace.

Other modules that require special mention are those that characterize the core of the functionality provided by the namespace. This is expressed by the notions of *host language* and *integration set*. Both of them relate to a set of conformance requirements for language profiles, which includes the requirement to incorporate at least the core set of modules. The set may be different for a host language and an integration set. A host language must incorporate the Structure module; an integration set need not. There may be other differences as well.

The main purpose of language profile conformance is to enhance interoperability. Preferably, the mandatory modules for host language conformance are defined in such a way that any document interchanged in a conforming language profile will yield a reasonable presentation when the document renderer, while supporting the associated mandatory module set, would ignore all other (unknown) elements and attributes. Here, "reasonable presentation" is to be understood as something intelligible, which is not necessarily a close reflection of the author's original intentions. To achieve the latter, a negotiation would have to be conducted to agree on the specific language profile to be used for the document interchange.

## 2.2 SMIL 2.0 Modules

***This section is normative.***

SMIL functionality is partitioned into ten functional areas. Within each functional area a further partitioning is applied into modules. All of these modules, and only these modules, are associated with the SMIL namespace.

The functional areas and their corresponding modules are:

1. Timing
    1. AccessKeyTiming
    2. BasicInlineTiming
    3. BasicTimeContainers
    4. EventTiming
    5. ExclTimeContainers
    6. FillDefault
    7. MediaMarkerTiming
    8. MinMaxTiming
    9. MultiArcTiming
    10. RepeatTiming
    11. RepeatValueTiming
    12. RestartDefault

13. RestartTiming
14. SyncbaseTiming
15. SyncBehavior
16. SyncBehaviorDefault
17. SyncMaster
18. TimeContainerAttributes
19. WallclockTiming

2. Time Manipulations
    1. TimeManipulations
3. Animation
    1. BasicAnimation
    2. SplineAnimation
4. Content Control
    1. BasicContentControl
    2. CustomTestAttributes
    3. PrefetchControl
    4. SkipContentControl
5. Layout
    1. AudioLayout
    2. BasicLayout
    3. HierarchicalLayout
    4. MultiWindowLayout
6. Linking
    1. BasicLinking
    2. LinkingAttributes
    3. ObjectLinking
7. Media Objects
    1. BasicMedia
    2. BrushMedia
    3. MediaAccessibility
    4. MediaClipping
    5. MediaClipMarkers
    6. MediaDescription
    7. MediaParam
8. Metainformation
    1. Metainformation
9. Structure
    1. Structure
10. Transitions
    1. BasicTransistions
    2. InlineTransitions
    3. TransitionModifiers


***This section is informative.***

Each of these modules introduces a set of semantically-related elements, properties, and attributes. Each functional area has a corresponding section in this specification document. Further details on each of the modules is specified within those sections.

The modules may be independent or complementary. For example, the SyncMaster module requires and builds upon the SyncBehavior module, but the PrefetchControl and SkipContentControl modules are independent from each other. In addition, some modules require modules from other functional areas.

Modules specify their integration requirements. When one module requires another module for basic features and as a prerequisite for integration, a language profile must include the second module in order to include the first. The first module is said to be a *dependent* of the second module. Dependency may be nested, in that a module may be dependent on a module that is dependent itself.

Table 1 presents the SMIL 2.0 modules and the modules they dependent on.

*Table 1: The SMIL 2.0 Modules and their Dependencies.*

| Module | Dependencies |
|---|---|
| AccessKeyTiming | NONE |
| AudioLayout | BasicLayout |
| BasicAnimation | BasicInlineTiming |
| BasicContentControl | NONE |
| BasicInlineTiming | NONE |
| BasicLayout | NONE |
| BasicLinking | NONE |
| BasicMedia | NONE |
| BasicTimeContainers | NONE |
| BasicTransitions | NONE |
| BrushMedia | NONE |
| CustomTestAttributes | BasicContentControl |
| EventTiming | NONE |
| ExclTimeContainers | NONE |
| FillDefault | BasicTimeContainers, and/or<br>ExclTimeContainers, and/or<br>TimeContainerAttributes |
| HierarchicalLayout | BasicLayout |
| InlineTransitions | NONE |

| | |
|---|---|
| LinkingAttributes | NONE |
| MediaAccessibility | MediaDescription |
| MediaClipMarkers | MediaClipping |
| MediaClipping | BasicMedia |
| MediaDescription | NONE |
| MediaMarkerTiming | NONE |
| MediaParam | BasicMedia |
| MetaInformation | NONE |
| MinMaxTiming | NONE |
| | AccessKeyTiming, and/or |
| MultiArcTiming | BasicInlineTiming, and/or<br>EventTiming, and/or<br>MediaMarkerTiming, and/or<br>RepeatValueTiming, and/or<br>SyncbaseTiming, and/or<br>WallclockTiming |
| MultiWindowLayout | BasicLayout |
| ObjectLinking | BasicLinking |
| PrefetchControl | NONE |
| RepeatTiming | NONE |
| RepeatValueTiming | NONE |
| RestartDefault | RestartTiming |
| RestartTiming | NONE |
| SkipContentControl | NONE |
| SplineAnimation | BasicAnimation |
| Structure | BasicContentControl, and<br>BasicInlineTiming, and<br>BasicLayout, and<br>BasicLinking, and<br>BasicMedia, and<br>BasicTimeContainers, and<br>SkipContentControl, and<br>SyncbaseTiming |
| SyncbaseTiming | NONE |
| SyncBehavior | BasicTimeContainers, and/or<br>ExclTimeContainers, and/or<br>TimeContainerAttributes |
| SyncBehaviorDefault | SyncBehavior |
| SyncMaster | SyncBehavior |
| TimeContainerAttributes | NONE |
| TimeManipulations | NONE |
| TransitionModifiers | BasicTransitions, and/or<br>InlineTransitions |
| WallclockTiming | NONE |

### 2.2.1 SMIL DOM

***This section is informative.***

SMIL is an XML-based language and conforms to the (XML) DOM Core [DOM1], [DOM2]. In the future, a SMIL-specific DOM recommendation may specify support for timing and synchronization, media integration, and other synchronized multimedia functionality.

A language profile may include DOM support. The granularity of DOM being supported corresponds to the modules being selected in that language profile. As with all modules, required support for the DOM is an option of the language profile.

## 2.3 Identifiers for SMIL 2.0 Modules and Language Profiles

***This section is informative.***

This section specifies the identifiers for the SMIL 2.0 namespace and the SMIL 2.0 modules. Each SMIL host language conformant language profile is requested to explicitly state the namespace URI that is to be used to identify it. That namespace URI must comply with the "Requirements on Identifiers for SMIL Host Language Conformant Language Profiles", defined below.

### 2.3.1 The SMIL Mime Type

***This section is normative.***

Documents authored in language profiles that include the SMIL Structure module can be associated with the "application/smil" mime type. Documents using the "`application/smil`" mime type are required to be host language conformant.

### 2.3.2 XML Namespace Identifier for the SMIL 2.0 Modules

***This section is normative.***

The XML namespace identifier for the complete set of SMIL 2.0 modules, and the elements and attributes that are contained within is:

http://www.w3.org/2001/SMIL20/

### 2.3.3 Identifiers for SMIL 2.0 Modules and Features

***This section is normative.***

Each module in this specification has a unique identifier associated with it. They are intended to uniquely and consistently identify each of them. They should be used as values in a test for whether an implementation includes a specific module, as well as in other circumstances where a need to refer to a specific SMIL 2.0 module is necessary.

Table 2 summarizes the identifiers for SMIL 2.0 modules.

*Table 2: The SMIL 2.0 Module Identifiers.*

| Module name | Identifier |
|---|---|
| AccessKeyTiming | http://www.w3.org/2001/SMIL20/AccessKeyTiming |
| AudioLayout | http://www.w3.org/2001/SMIL20/AudioLayout |
| BasicAnimation | http://www.w3.org/2001/SMIL20/BasicAnimation |
| BasicContentControl | http://www.w3.org/2001/SMIL20/BasicContentControl |
| BasicInlineTiming | http://www.w3.org/2001/SMIL20/BasicInlineTiming |
| BasicLayout | http://www.w3.org/2001/SMIL20/BasicLayout |
| BasicLinking | http://www.w3.org/2001/SMIL20/BasicLinking |
| BasicMedia | http://www.w3.org/2001/SMIL20/BasicMedia |
| BasicTimeContainers | http://www.w3.org/2001/SMIL20/BasicTimeContainers |
| BrushMedia | http://www.w3.org/2001/SMIL20/BrushMedia |
| CustomTestAttributes | http://www.w3.org/2001/SMIL20/CustomTestAttributes |
| EventTiming | http://www.w3.org/2001/SMIL20/EventTiming |
| ExclTimeContainers | http://www.w3.org/2001/SMIL20/ExclTimeContainers |
| FillDefault | http://www.w3.org/2001/SMIL20/FillDefault |
| HierarchicalLayout | http://www.w3.org/2001/SMIL20/HierarchicalLayout |
| InlineTransitions | http://www.w3.org/2001/SMIL20/InlineTransitions |
| LinkingAttributes | http://www.w3.org/2001/SMIL20/LinkingAttributes |
| MediaAccessibility | http://www.w3.org/2001/SMIL20/MediaAccessibility |
| MediaClipMarkers | http://www.w3.org/2001/SMIL20/MediaClipMarkers |
| MediaClipping | http://www.w3.org/2001/SMIL20/MediaClipping |
| MediaDescription | http://www.w3.org/2001/SMIL20/MediaDescription |
| MediaMarkerTiming | http://www.w3.org/2001/SMIL20/MediaMarkerTiming |
| MediaParam | http://www.w3.org/2001/SMIL20/MediaParam |
| Metainformation | http://www.w3.org/2001/SMIL20/Metainformation |
| MinMaxTiming | http://www.w3.org/2001/SMIL20/MinMaxTiming |
| MultiArcTiming | http://www.w3.org/2001/SMIL20/MultiArcTiming |
| MultiWindowLayout | http://www.w3.org/2001/SMIL20/MultiWindowLayout |
| ObjectLinking | http://www.w3.org/2001/SMIL20/ObjectLinking |
| PrefetchControl | http://www.w3.org/2001/SMIL20/PrefetchControl |
| RepeatTiming | http://www.w3.org/2001/SMIL20/RepeatTiming |
| RepeatValueTiming | http://www.w3.org/2001/SMIL20/RepeatValueTiming |
| RestartDefault | http://www.w3.org/2001/SMIL20/RestartDefault |
| RestartTiming | http://www.w3.org/2001/SMIL20/RestartTiming |
| SkipContentControl | http://www.w3.org/2001/SMIL20/SkipContentControl |
| SplineAnimation | http://www.w3.org/2001/SMIL20/SplineAnimation |
| Structure | http://www.w3.org/2001/SMIL20/Structure |
| SyncbaseTiming | http://www.w3.org/2001/SMIL20/SyncbaseTiming |
| SyncBehavior | http://www.w3.org/2001/SMIL20/SyncBehavior |
| SyncBehaviorDefault | http://www.w3.org/2001/SMIL20/SyncBehaviorDefault |
| SyncMaster | http://www.w3.org/2001/SMIL20/SyncMaster |
| TimeContainerAttributes | http://www.w3.org/2001/SMIL20/TimeContainerAttributes |
| TimeManipulations | http://www.w3.org/2001/SMIL20/TimeManipulations |
| TransitionModifiers | http://www.w3.org/2001/SMIL20/TransitionModifiers |
| WallclockTiming | http://www.w3.org/2001/SMIL20/WallclockTiming |

In addition to the module identifiers above, there may be different features and variances from one language profile to another that may not be expressed as the support or non-support of a particular module. These features may be expressed using the following identifiers:

**http://www.w3.org/2001/SMIL20/NestedTimeContainers**
: Profile allows nesting of the par and seq time containers.

**http://www.w3.org/2001/SMIL20/DeprecatedFeatures**
: Profile supports deprecated SMIL 1.0 features.

Implementations that support the SMIL BasicContentControl module must allow these as identifiers for use with the XML namespace mechanism, and

must allow the associated namespace identifier to be used with the systemRequired test attribute. Profiles must identify those attributes for which an implementation must return "true" (this is an integration requirement). Implementations must return "false" for modules or features which are not fully supported.

Modules can also be identified collectively. The following four module collections are defined:

`http://www.w3.org/2001/SMIL20/`
: All the modules specified by the SMIL 2.0 specification.

`http://www.w3.org/2001/SMIL20/Language`
: The modules used by the SMIL 2.0 Language profile.

`http://www.w3.org/2001/SMIL20/HostLanguage`
: The modules required for SMIL Host Language Conformance.

`http://www.w3.org/2001/SMIL20/IntegrationSet`
: The modules required for SMIL Integration Set Conformance.

## 2.4 SMIL Conformance

***This section is informative.***

In this section we specify the rules for SMIL host language and SMIL integration set conformance. First, the conformance requirements for host language conformance and integration set conformance are given. The requirements are similar to those set forth for XHTML host language document type conformance and XHTML integration set document type conformance [XMOD]. In a final section the requirements on identifiers for host language conformant language profiles are given.

Currently, there exist three language profiles using SMIL 2.0 Modules. They are the SMIL 2.0 Language Profile, the SMIL 2.0 Basic Language Profile, and the XHTML+SMIL 2.0 Language Profile [XHTML+SMIL]. The first two are SMIL host language conformant, the third is SMIL integration set conformant.

***This section is normative.***

The following two tables list names used to collectively reference certain sets of SMIL 2.0 elements and attributes. These are used in the definitions of the minimum support in the two sections below on SMIL host language conformance and SMIL integration set conformance. The term "minimum support" is used to refer to the minimum set of elements that an element can contain, and the minimum set of attributes that can be used on an element.

*Table 3: Names of SMIL 2.0 Element Collections.*

| Element Set Name | Elements |
|---|---|
| TIMING-ELMS | par, seq |
| MEDIA-ELMS | ref, animation, audio, img, video, text, textstream |
| EMPTY | no elements are required as a minimum |

*Table 4: Names of SMIL 2.0 Attribute Collections.*

| Attribute Set Name | Attributes |
|---|---|
| TIMING-ATTRS | begin, end, dur, repeatDur, repeatCount, max, min, fill, endsync |
| CONTCTRL-ATTRS | systemBitrate, systemCaptions, systemLanguage, systemRequired, systemScreenSize, systemScreenDepth, systemOverdubOrSubtitle, systemAudioDesc, systemOperatingSystem, systemCPU, systemComponent |
| MEDIA-ATTRS | src, type |
| LINKING-ATTRS | href, sourceLevel, destinationLevel, sourcePlaystate, destinationPlaystate, show, accesskey, tabindex, target, external, actuate, alt |
| COMMON-ATTRS | id, class, xml:lang, title |

### 2.4.1 SMIL Host Language Conformance

***This section is normative.***

A language profile is said to be SMIL 2.0 host language conformant if it includes the following modules:

1. Structure
2. BasicContentControl
3. BasicInlineTiming
4. BasicLayout
5. BasicLinking
6. BasicMedia
7. BasicTimeContainers
8. MinMaxTiming
9. RepeatTiming
10. SkipContentControl
11. SyncbaseTiming

1. The language profile defines what modules it collects.
2. The language profile includes all elements, attributes, and attribute values specified by the collected modules.
3. The language profile fulfills the "minimum support" requirements for elements and attributes as listed in Table 5 below.
4. The language profile complies with the integration requirements set forth by the modules it collects.
5. The language profile specifies the semantics related to the integration of the modules.
6. The language profile defines its DTD or XML Schema.

7. The language profile defines a unique identifier conforming to the requirements set forth in Requirements on Identifiers for SMIL Host Language Conformant Language Profiles.
8. The SyncbaseTiming module should be included in Host Language conformant profiles, although it is not strictly required. We strongly recommend inclusion of this module in Host Language conformant profiles to maintain a high level of consistency and interoperability with other languages that have integrated SMIL modules including the SMIL 2.0 Language, XHTML+SMIL, and SVG. Only profiles designed to operate on severely constrained devices may omit the SyncbaseTiming module.

*Table 5: Minimum Support for SMIL Host Language Conformance.*

| Element | Minimum Support | |
|---|---|---|
| | Elements | Attributes |
| smil | head, body | COMMON-ATTRS, CONTCTRL-ATTRS, xmlns |
| head | layout, switch | COMMON-ATTRS |
| body | TIMING-ELMS, MEDIA-ELMS, switch, a | COMMON-ATTRS |
| layout | root-layout, region | COMMON-ATTRS, CONTCTRL-ATTRS, type |
| root-layout | EMPTY | COMMON-ATTRS, backgroundColor, height, width, skip-content |
| region | EMPTY | COMMON-ATTRS, backgroundColor, bottom, fit, height, left, right, showBackground, top, width, z-index, skip-content, regionName |
| ref, animation, audio, img, video, text, textstream | area | COMMON-ATTRS, CONTCTRL-ATTRS, TIMING-ATTRS, repeat, MEDIA-ATTRS, region |
| a | MEDIA-ELMS | COMMON-ATTRS, LINKING-ATTRS |
| area | EMPTY | COMMON-ATTRS, LINKING-ATTRS, TIMING-ATTRS, repeat, shape, coords, nohref |
| par, seq | TIMING-ELMS, MEDIA-ELMS, switch, a | COMMON-ATTRS, CONTCTRL-ATTRS, TIMING-ATTRS, repeat |
| switch | TIMING-ELMS, MEDIA-ELMS, a, layout | COMMON-ATTRS, CONTCTRL-ATTRS |

Support of deprecated elements and attributes is required for SMIL 2.0 host language conformance for all modules the given language supports. For example, if a SMIL 2.0 host language supports the MultiArcTiming module, it must support the deprecated syntax defined in the MultiArcTiming module.

Since the SMIL 2.0 Structure module may only be used in a profile that is SMIL host language conformant, this implies that the SMIL 2.0 Structure module must at least be accompanied with the nine other modules required for host language conformance that were named above. Those modules themselves can still be used in other, non SMIL host language conformant, language profiles.

### 2.4.2 SMIL Integration Set Conformance

***This section is normative.***

A language profile is said to be SMIL 2.0 integration set conformant if it includes the following modules:

1. BasicContentControl
2. BasicInlineTiming
3. BasicMedia
4. BasicTimeContainers
5. MinMaxTiming
6. RepeatTiming
7. SyncbaseTiming

In addition, the following requirements must be satisfied:

1. The language profile defines what modules it collects.
2. The language profile includes all elements, attributes, and attribute values specified by the collected SMIL 2.0 modules.
3. The language profile fulfills the "minimum support" requirements for elements and attributes as listed in Table 6 below.
4. The language profile complies with the integration requirements set forth by the SMIL 2.0 modules it collects.
5. The SyncbaseTiming module should be included in Integration Set conformant profiles, although it is not strictly required. We strongly recommend inclusion of this module in Integration Set conformant profiles to maintain a high level of consistency and interoperability with other languages that have integrated SMIL modules including the SMIL 2.0 Language, XHTML+SMIL [XHTML+SMIL], and SVG [SVG]. Only profiles designed to operate on severely constrained devices may omit the SyncbaseTiming module.

*Table 6: Minimum Support for SMIL Integration Set Conformance.*

| Element | Minimum Support | |
|---|---|---|
| | Elements | Attributes |
| ref, animation, audio, img, video, text, textstream | | CONTCTRL-ATTRS, TIMING-ATTRS, MEDIA-ATTRS |
| par, seq | TIMING-ELMS, MEDIA-ELMS, switch, a | CONTCTRL-ATTRS, TIMING-ATTRS |
| switch | TIMING-ELMS, MEDIA-ELMS | CONTCTRL-ATTRS |

Support of deprecated elements and attributes is not required for SMIL 2.0 integration set conformance. However, when included, the above requirements also apply to these elements and attributes. Also, when supported, it is required that all the deprecated elements and attributes from all the included modules are supported as a whole.

### 2.4.3 Requirements on Identifiers for SMIL Host Language Conformant Language Profiles

***This section is informative.***

A language profile is specified through its DTD or XML Schema. The identifier of these can be used to identify the language profile. SMIL 1.0 has specified the default namespace declaration on its root element, smil, as the decisive identifier for distinguishing it from other language profiles

[SMIL10]. For that purpose SMIL 1.0 has specified

```
http://www.w3.org/TR/REC-smil
```

as the namespace identifier for SMIL 1.0.

***This section is normative.***

For the purpose of identifying the version and the language profile used, SMIL host language conformant documents must satisfy the following requirements:

1. The document should declare a default namespace [XML-NS].
2. The default namespace must be declared on the smil root element.
3. In case the SMIL host language conformant language profile has been issued as a W3C Recommendation, the default namespace identifier must satisfy the following requirements:
    1. The URI is constructed conformant to the requirements set forth by the W3C [W3C-NSURI].
    2. The default namespace identifier should identify the language profile.
       In case the language profile is a subset of a larger one, the default namespace identifier may also identify that larger language profile. The module collection that is required to be supported in the subset language profile may be indicated through the systemRequired attribute on the smil element.
       See the SMIL Basic Language Profile specification for an example.

### 2.4.4 Error Handling in SMIL Host Language Conformant Documents

***This section is normative.***

Syntax errors in a SMIL Host Language conformant document are handled according to the XML rules for well-formed or valid XML [XML10].

Semantic errors can arise at various levels. One is where the declared attribute values are of unknown value. Another is where the assembled presentation is (possibly) conflicting, as in a case where media objects are competing for display space or where they are synchronized ambiguously. These latter types, although maybe an error according to the author's intentions, are not considered an error and the user agent will present according to the resolution rules defined in this specification.

### 2.4.5 Handling of Syntax errors in Attribute Values

***This section is normative.***

Errors in attribute values might remain undetectable to the parser, because the value type is declared as CDATA, or because the value range is open ended, as in the case of events, for example. However, errors in attribute values can be detected within a given language profile, where that language profile specifies the supported value set. Specifications of language profiles are required to specify the error handling that is required when such an attribute value error occurs.

## 2.5 Creating a DTD for a Language Profile

***This section is informative.***

This section describes how language profiles could be defined using the SMIL 2.0 modular DTDs. The reader is assumed to be familiar with the mechanisms defined in "Modularization of XHTML" [XMOD], in particular Appendix D [XMOD-APPD] and Appendix E [XMOD-APPE]. In general, the SMIL 2.0 modular DTDs use the same mechanisms as the XHTML modular DTDs use. Exceptions to this are:

1. SMIL supports qualified attribute names for SMIL attributes that can appear on non-SMIL elements. This enables these attributes to use prefixes to indicate that they belong to the SMIL 2.0 namespace.
2. SMIL supports module level INCLUDE/IGNORE instead of XHTML's element/attlist level. Similar to XHTML Modularization, this prohibits profiles from importing only part of a module -- they have to support either all the elements and attributes or none.

Below, we give a short description of the files that are used to define the SMIL 2.0 modular DTDs. See the table and the end of the section for a complete list of the filenames involved.

Following the same mechanisms as the XHTML modular DTDs, the SMIL 2.0 specification places the XML element declarations (e.g. <!ELEMENT...>) and attribute list declarations (e.g. <!ATTLIST...>) of all SMIL 2.0 elements in separate files, the SMIL module files. A SMIL module file is provided for each functional area in the SMIL 2.0 specification (that is, there is a SMIL module file for animation, layout, timing, etc).

The SMIL module files are used in the normative definitions of the specification of the SMIL 2.0 Language Profile. Usage of the same module files for defining other SMIL profiles is recommended, but not required. The requirements that SMIL language profiles must follow are stated in the SMIL 2.0 specification, not in the DTD code.

To make the SMIL module files independent of each other, and independent of the language profiles, the element and attribute declarations make heavy use of XML entities. This provides profiles with the necessary hooks to define the actual content models and attributes of the SMIL elements.

The SMIL 2.0 Language Profile provides examples of how the SMIL module files can be used. Most of the DTD files are reused across the different profiles. Reused are the SMIL module files, the files that define the data types and the common attributes, the "qname" file that takes care of adding namespace prefixes if necessary, and the framework file, which takes care of including files in the appropriate order.

The files that are different for each profile are the driver file and document model file. This would, in general, also apply to new profiles: to define a new language profile, one has to write the extension module(s), the driver file that defines which modules are used, and a document model file that defines the extended document model. The driver file and document model file are described in more detail below.

**The driver file.**

This is the file that would be referenced by a document's DOCTYPE declaration. Its main job is to define which document model file and which of the SMIL module files the profile is using. It may also define an optional namespace to be used in all namespace prefixes. For example, to prefix all SMIL element names with "foobar", the following can be added to the start of the profile:

```
<!ENTITY % SMIL.prefixed "INCLUDE" >
<!ENTITY % SMIL.prefix "foobar" >
```

Elements defined in their modules as, for example, <video> will become parsed as <foobar:video>. This also applies for SMIL attributes that appear on

other elements, so, for example, "begin" becomes "foobar:begin". The default is that the qname prefix is empty -- that is, it is effectively turned off by default.

After these definitions, the driver file includes the framework file (which will subsequently include the data type, common attributes, qname and document model file), after which the SMIL module files are included that are used by this profile.

**The document model file.**

The document model file contains the XML entities that are used by the SMIL module files to define the content models and attribute lists of the elements in that profile.

Content models generally differ from profile to profile, or contain elements from other modules. To avoid these dependencies in the SMIL module files, content models need to be defined in the document model file. The (dummy) default content model as defined in the SMIL module files is "EMPTY" for all SMIL 2.0 elements.

For the same reasons, the SMIL module files only define a default attribute list for their elements. This default list only contains the SMIL 2.0 core attributes and the attributes that are defined in the same SMIL module file. All other attributes need to be added to this default list by defining the appropriate XML entities. For example, the Media Objects Module file only adds the core and media related attributes on the media objects; other attributes, such as the timing attributes, are added to this list by the document model file.

*Table 7: Formal public identifiers and system identifiers of all files used in the SMIL 2.0 modular DTDs.*

| Driver files for the predefined profiles | |
|---|---|
| -//W3C//DTD SMIL 2.0//EN | http://www.w3.org/2001/SMIL20/SMIL20.dtd |
| **Document model files for the predefined profiles** | |
| -//W3C//ENTITIES SMIL 2.0 Document Model 1.0//EN | http://www.w3.org/2001/SMIL20/smil-model-1.mod |
| **SMIL 2.0 module files** | |
| -//W3C//ELEMENTS SMIL 2.0 Animation//EN | http://www.w3.org/2001/SMIL20/SMIL-anim.mod |
| -//W3C//ELEMENTS SMIL 2.0 Content Control//EN | http://www.w3.org/2001/SMIL20/SMIL-control.mod |
| -//W3C//ELEMENTS SMIL 2.0 Layout//EN | http://www.w3.org/2001/SMIL20/SMIL-layout.mod |
| -//W3C//ELEMENTS SMIL 2.0 Linking//EN | http://www.w3.org/2001/SMIL20/SMIL-link.mod |
| -//W3C//ELEMENTS SMIL 2.0 Media Objects//EN | http://www.w3.org/2001/SMIL20/SMIL-media.mod |
| -//W3C//ELEMENTS SMIL 2.0 Document Metainformation//EN | http://www.w3.org/2001/SMIL20/SMIL-metainformation.mod |
| -//W3C//ELEMENTS SMIL 2.0 Document Structure//EN | http://www.w3.org/2001/SMIL20/SMIL-struct.mod |
| -//W3C//ELEMENTS SMIL 2.0 Timing//EN | http://www.w3.org/2001/SMIL20/SMIL-timing.mod |
| -//W3C//ELEMENTS SMIL 2.0 Transition//EN | http://www.w3.org/2001/SMIL20/SMIL-transition.mod |
| **Other utilities: data types, common attributes, qname and frame work files** | |
| -//W3C//ENTITIES SMIL 2.0 Datatypes 1.0//EN | http://www.w3.org/2001/SMIL20/smil-datatypes-1.mod |
| -//W3C//ENTITIES SMIL 2.0 Common Attributes 1.0//EN | http://www.w3.org/2001/SMIL20/smil-attribs-1.mod |
| -//W3C//ENTITIES SMIL 2.0 Qualified Names 1.0//EN | http://www.w3.org/2001/SMIL20/smil-qname-1.mod |
| -//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN | http://www.w3.org/2001/SMIL20/smil-framework-1.mod |

# 3. The SMIL 2.0 Animation Modules

***Editors***
Patrick Schmitz (cogit@ludicrum.org), (Microsoft)
Aaron Cohen (aaron.m.cohen@intel.com), (Intel)
[illegible]

## 3.1 Introduction

This section defines the SMIL 2.0 Animation Modules, which are composed of a BasicAnimation module and a SplineAnimation module. These modules contain elements and attributes for incorporating animation onto a time line, and a mechanism for composing the effects of multiple animations. Since these elements and attributes are defined in modules, designers of other markup languages can choose whether or not to include this functionality in their languages. Language designers incorporating other SMIL modules do not need to include the animation modules if animation functionality is not needed.

The examples in this document that include syntax for a host language use [SMIL10], [SVG], [HTML4] and [CSS2]. These are provided as an indication of possible integrations with various host languages.

While this document defines a base set of animation capabilities, it is assumed that host languages may build upon the support to define additional or more specialized animation elements. Animation only manipulates attributes and properties of the target elements, and so does not require any knowledge of the target element semantics beyond basic type information.

This module depends on the SMIL 2.0 BasicInlineTiming module, using elements and attributes from the Timing module for its time line. The BasicInlineTiming module is a prerequisite for any profile using SMIL Animation. The reader is presumed to have read and be familiar with the SMIL 2.0 Timing modules.

This section first presents the underlying principals of animation in SMIL 2.0, then the elements and attributes of the BasicAnimation module and of the SplineAnimation module.

## 3.2 Animation Model

This section describes the semantics underlying the SMIL 2.0 animation modules. The specific elements are not described here, but rather the common concepts and syntax that comprise the model for animation. Document issues are described, as well as the means to target an element for

common concepts and syntax that comprise the model for animation. Document issues are described, as well as the means to target an element for animation. The animation model is then defined by building up from the simplest to the most complex concepts: first the simple duration and simple animation function `f(t)`, and then the overall effect `F(t,u)`.

### 3.2.1 The simple animation function f(t)

Animation is defined as a time-based function of a *target element* (or more specifically of some *attribute* of the target element, the *target attribute*). The animation defines a mapping of time to values for the target attribute. This mapping takes into account all aspects of timing, as well as animation-specific semantics. The overall mapping is based on a *simple animation function* `f(t)` which describes the animation over the simple duration of the element. Every animation defines a simple animation function which produces a value for the target attribute for any time within the simple duration.

*Normative*

> A *target attribute* is the name of a feature of a target element as defined in a host language document.

This may be (e.g.) an XML attribute contained in the element or a CSS property that applies to the element. By default, the target element of an animation will be the parent of the animation element (an animation element is typically a child of the target element). However, the target may be any element in the document, identified either by an XML ID reference or via an XLink [XLINK] locator reference.

As a simple example, the following defines an animation of an SVG rectangle shape. The rectangle will change from being tall and thin to being short and wide.

```
<rect ...>
  <animate attributeName="width"  from="10px"  to="100px"
           begin="0s" dur="10s" />
  <animate attributeName="height" from="100px" to="10px"
           begin="0s" dur="10s" />
</rect>
```

The rectangle begins with a width of 10 pixels and increases to a width of 100 pixels over the course of 10 seconds. Over the same ten seconds, the height of the rectangle changes from 100 pixels to 10 pixels.

When an animation is running, it should not actually change the attribute values in the DOM [DOM2]. The animation runtime should maintain a *presentation value* for each animated attribute, separate from the DOM or CSS Object Model (OM). If an implementation does not support an object model, it should maintain the original value as defined by the document as well as the presentation value. The presentation value is reflected in the displayed form of the document. Animations thus manipulate the presentation value, and should not affect the *base value* exposed by DOM or CSS OM. This is detailed in The animation sandwich model.

*Normative*

> The *base value* of a target attribute `a` at time `t` is the value of `a` to which animation is applied at time `t`.

The presentation value reflects the *effect* of animation on the base value. The effect is the change to the base value of the target attribute at any given time. When an animation completes, the effect of the animation is no longer applied, and the presentation value reverts to the base value by default. The animation effect can also be extended to *freeze* the last value for the length of time determined by the semantics of the fill attribute.

An animation element defines a *simple animation function* which is evaluated as needed over time by the implementation. The resulting values are applied to the presentation value for the target attribute. Animation functions are continuous in time and can be sampled at whatever frame rate is appropriate for the rendering system. The syntactic representation of the simple animation function is independent of this model, and may be described in a variety of ways. The animation elements in this specification support syntax for a set of discrete or interpolated values, a path syntax for motion based upon SVG paths, keyframe based timing, evenly paced interpolation, and variants on these features.

In the example immediately above, the simple animation function for the width attribute, specified by `'from="10px" to="100px" ... dur="10s"'` is

> `f(t) = (10 + 90*t/10) px`, where `t` is given in seconds.

Simple animation functions may be defined which have additional parameters, or that are purely or partially algorithmic. For example, a "to" animation interpolates from the current value to the "to" value:

```
<animate attributeName="top" to="10" dur="2.5s" />
```

The animation function is a function of the current position, as well as of time:

> `f(t,u) = (u*(2.5s-t)/2.5s) + 10*(t/2.5s)`

In all cases, the animation exposes this as a function of time.

*Normative*

> The *simple animation function* defined by an animation element is a function of time, `f(t)`, defined for times `t`, `0<=t<=d`, where `d` is the simple duration of the element.
>
> The simple animation function may be defined as a function which depends on factors in addition to time. This does not affect the model of animation, beyond the trivial addition of additional parameters to `f(t)`, such as `f(t,u)` used in the "to" animation example immediately above.

Animations can be defined to either override or add to the base value of an attribute. In this context, the base value may be the DOM value, or the result of other animations that also target the same attribute. This more general concept of a base value is termed the *underlying value*. Animations that add to the underlying value are described as *additive* animations. Animations that override the underlying value are referred to as *non-additive* animations. The *animation effect function* of an element is the function which includes the affect of the underlying value and accounts for repeating and freezing of the element. Because the animation effect can be affected by repeating and freezing, it is defined over the active duration of the element rather than its simple duration.

Animations can be combined in ways which produce intermediate values outside of the domain of the target attribute, but where the presentation value produced is valid. The *type* of a target attribute is this larger set. This is detailed in The animation sandwich model.

*Normative*

> The type of a target attribute `a` is the base type of which the domain of `a` is a subset.

The *type* of a target attribute `a` is the base type of which the domain of `a` is a subset.

The *animation effect function*, `F(t,u)`, of an animation element with active duration `AD` is a function mapping times `t`: `0<=t<=AD` and values `u` of the type of the target attribute `a` into values of the type of `a`.

The *underlying value* `u` of a target attribute `a` of an animation element at time `t` is the value of `a` to which the animation effect is applied at time `t`.

The animation effect function `F(t,u)` is usually defined as a function of the simple animation function `f(t)`. `f(t)` must be defined in such a manner that `F(t,u)` produces values of the correct type.

### 3.2.2 Summary of symbols used in the semantic descriptions

`a`
: The target attribute of an animation element.

`d`
: The simple duration of the element.

`AD`
: The active duration of the element.

`t`
: A time. Depending on the context, `t` may be in user-perceived time, an element's active duration, or its simple duration.

`u`
: The underlying value of the target attribute, as affected by all animations specified earlier in document order.

`f(t)`
: The simple animation function of times within the simple duration. Note that while `F(t,u)` defines the mapping for the entire animation, `f(t)` has a simplified model that just handles the simple duration.

`F(t,u)`
: The effect of an animation for any point in the active duration of the animation. This maps times within the active duration (`t`: `0<=t<=AD`) and an underlying value to a value for the target attribute. A time value of 0 corresponds to the time at which the animation begins. `F(t,u)` combines the simple animation function `f(t)` with all the other aspects of animation and timing controls.

### 3.2.3 The animation sandwich model

When an animation is running, it does not actually change the attribute values in the DOM. The animation runtime should ideally maintain a *presentation value* for any target attribute, separate from the DOM, CSS, or other object model (OM) in which the target attribute is defined. The presentation value is reflected in the display form of the document. The effect of animations is to manipulate this presentation value, and not to affect the underlying DOM or CSS OM values.

The remainder of this discussion uses the generic term OM for both the XML DOM [DOM2] as well as the CSS-OM. If an implementation does not support an object model, it should ideally maintain the original value as defined by the document as well as the presentation value; for the purposes of this section, we will consider this original value to be equivalent to the value in the OM.

In some implementations of DOM, it may be difficult or impractical to main a presentation value as described. CSS values should always be supported as described, as the CSS-OM provides a mechanism to do so. In implementations that do not support separate presentation values for general XML DOM properties, the implementation must at least restore the original value when animations no longer have an effect.

The rest of this discussion assumes the recommended approach using a separate presentation value.

The model accounting for the OM and concurrently active or frozen animations for a given attribute is described as a "sandwich", a loose analogy to the layers of meat and cheeses in a "submarine sandwich" (a long sandwich made with many pieces of meats and cheese layered along the length of the bread). In the analogy, time is associated with the length of the sandwich, and each animation has its duration represented by the length of bread that the layer covers. On the bottom of the sandwich is the base value taken from the OM. Each active (or frozen) animation is a layer above this. The layers (i.e. the animations) are placed on the sandwich both in time along the length of the bread, as well as in order according to *priority*, with higher priority animations placed above (i.e. on top of) lower priority animations. At any given point in time, you can take a slice of the sandwich and see how the animation layers stack up.

Note that animations apply into the presentation value service of the OM in which the attribute is defined, and pass the resulting value on to the next layer of document processing. This does not replace or override any of the normal document OM processing cascade.

Specifically, animating an attribute defined in XML will modify the presentation value before it is passed through the style sheet cascade, using the XML DOM value as its base. Animating an attribute defined in a style sheet language will modify the presentation value passed through the remainder of the cascade.

In CSS2 and the DOM 2 CSS-OM, the terms "specified", "computed" and "actual" are used to describe the results of evaluating the syntax, the cascade and the presentation rendering. When animation is applied to CSS properties of a particular element, the base value to be animated is read using the (readonly) `getComputedStyle()` method on that element. The values produced by the animation are written into an override stylesheet for that element, which may be obtained using the `getOverrideStyle()` method. These new values then affect the cascade and are reflected in a new computed value (and thus, modified presentation). This means that the effect of animation overrides all style sheet rules, except for user rules with the `!important` property. This enables `!important` user style settings to have priority over animations, an important requirement for accessibility. Note that the animation may have side effects upon the document layout. See also section 6.1, "Specified, computed, and actual values," of [CSS2] and section 5.2.1, "Override and computed style sheet," of [DOM2CSS].

Within an OM, animations are prioritized according to when each begins. The animation first begun has lowest priority and the most recently begun animation has highest priority. When two animations start at the same moment in time, the activation order is resolved as follows:

- If one animation is a *time dependent* of another (e.g. it is specified to begin when another begins), then the time dependent is considered to activate *after* the syncbase element, and so has higher priority. Time dependency is further discussed in Propagating changes to times. This rule applies independent of the timing described for the syncbase element - i.e. it does not matter whether the syncbase element begins on an offset, relative to another syncbase, relative to an event-base, or via hyperlinking. In all cases, the syncbase is begun before any time dependents are begun, and so the syncbase has lower priority than the time dependent.
- If two animations share no time dependency relationship (e.g. neither is defined relative to the other, even indirectly) the element that appears first in the document has lower priority. This includes the cases in which two animation elements are defined relative to the same syncbase or event-base.

Note that if an animation is restarted (see also Restarting animations), it will always move to the top of the priority list, as it becomes the most recently activated animation. That is, when an animation restarts, its layer is pulled out of the sandwich, and added back on the very top. In contrast, when an element repeats the priority is not affected (repeat behavior is not defined as restarting).

Each additive animation adds its effect to the result of all sandwich layers below. A non-additive animation simply overrides the result of all lower sandwich layers. The end result at the top of the sandwich is the presentation value that must be reflected in the document view.

Some attributes that support additive animation have a defined legal range for values (e.g. an opacity attribute may allow values between 0 and 1). In some cases, an animation function may yield out of range values. It is recommended that implementations clamp the results to the legal range as late as possible, before applying them to the presentation value. Ideally, the effect of all the animations active or frozen at a given point should be combined, before any clamping is performed. Although individual animation functions may yield out of range values, the combination of additive animations may still be legal. Clamping only the final result and not the effect of the individual animation functions provides support for these cases. Intermediate results may be clamped when necessary although this is not optimal. The host language must define the clamping semantics for each attribute that can be animated. As an example, this is defined for animateColor element.

Initially, before any animations for a given attribute are active, the presentation value will be identical to the original value specified in the document (the OM value).

When all animations for a given attribute have completed and the associated animation effects are no longer applied, the presentation value will again be equal to the OM value. Note that if any animation is defined with fill="freeze", the effect of the animation will be applied as long as the animation element remains in the frozen state, and so the presentation value will continue to reflect the animation effect. Refer also to the section "Freezing animations".

Some animations (e.g. animateMotion) will *implicitly* target an attribute, or possibly several attributes (e.g. the "posX" and "posY" attributes of some layout model). These animations must be combined with any other animations for each attribute that is affected. Thus, e.g. an animateMotion animation may be in more than one animation sandwich (depending upon the layout model of the host language). For animation elements that implicitly target attributes, the host language designer must specify which attributes are implicitly targeted, and the runtime must accordingly combine animations for the respective attributes.

Note that any queries (via DOM interfaces) on the target attribute will reflect the OM value, and will not reflect the effect of animations. Note also that the OM value may still be changed via the OM interfaces (e.g. using script). While it may be useful or desired to provide access to the final presentation value after all animation effects have been applied, such an interface is not provided as part of SMIL Animation. A future version may address this.

Although animation does not manipulate the OM values, the document display must reflect changes to the OM values. Host languages can support script languages that can manipulate attribute values directly in the OM. If an animation is active or frozen while a change to the OM value is made, the behavior is dependent upon whether the animation is defined to be additive or not, as follows: (see also the section Additive animation).

- If only additive animations are active or frozen (i.e. no non-additive animations are active or frozen for the given attribute) when the OM value is changed, the presentation value must reflect the changed OM value as well as the effect of the additive animations. When the animations complete and the effect of each is no longer applied, the presentation value will be equal to the changed OM value.
- If any non-additive animation is running when the OM value is changed, the presentation value will not reflect the changed OM value, but will only reflect the effect of the highest priority non-additive animation, and any still higher priority additive animations. When all non-additive animations complete and the effect of each is no longer applied, the presentation value will reflect the changed OM value and the effect of any additive animations that are active or frozen.

### 3.2.4 Animation elements as "continuous media"

Within the timing model, animation is considered to be "continuous" media. The animation elements defined in SMIL Animation do not have a natural intrinsic duration, so they are assigned an intrinsic duration of *indefinite*.

This has several consequences, which are noted in various sections below. In particular, most animation elements will have an explicit duration set with the dur attribute, since a finite, known duration is required for interpolation.

### 3.2.5 The animation effect function F(t,u)

As described above, the simple animation function $f(t)$ defines the animation for the simple duration $d$. However, SMIL Timing allows the author to repeat the simple duration. SMIL Timing also allows authors to specify whether the animation should simply end when the active duration completes, or whether it should be *frozen* at the last value. SMIL Animation specifies what it means for an animation to be frozen. In addition, the author can specify how each animation should be combined with other animations and the original DOM value.

This section describes the semantics for the additional functionality, including a detailed model for combining animations. This is presented as a sequence of functions building on the simple animation function:

- The *repeated animation function*, $f_r(t)$, defines the effect of repeating an animation element.
- The *cumulative animation function*, $f_c(t)$, defines the effect of accumulating values from one iteration to the next of a repeated animation element.
- The *frozen animation function*, $f_f(t)$, includes the effect of freezing an animation element at the end of its active duration.
- The *animation effect function*, $F(t,u)$, defines how a an animation element depends on the underlying value $u$ of the target attribute.

Since these functions describe the animation outside of the simple duration, they are defined for any non-negative time $t$.

**Repeating animations**

As described in the section Interval timing of the BasicInlineTiming module, repeating an element causes the element to be "played" several times in sequence. The repeated period is 0 to the simple duration of the element. Animation follows this model, where "playing" the animation means applying the simple animation function $f(t)$ repeatedly.

*Normative*

The *repeated animation function*, $f_r(t)$, for any simple animation function $f(t)$ is

$$f_r(t) = f( \text{REMAINDER}( t, d ) ),$$

where $t>=0$, $d$ is the simple duration , and $\text{REMAINDER}( t, d )$ is defined as $(t - d*\text{floor}(t/d))$.

This formulation follows the end-point exclusive model described in Interval timing. As an animation repeats, it starts at $f(0)$, is sampled and applied up to but not including the end-point $f(d)$. At the end of the simple duration, i.e. at the beginning of the next iteration, it starts back at $f(0)$. $f(d)$ may never actually be applied.

*Examples*

In the following example, the 2.5 second animation function will be repeated twice; the active duration will be 5 seconds. The attribute top will go from 0 to (almost) 10, return to 0 at 2.5 seconds, and repeat.

```
<animate attributeName="top" from="0" to="10" dur="2.5s"
         repeatCount="2" />
```

In the following example, the animation function will be repeated two full times and then the first half is repeated once more; the active duration will be 7.5 seconds.

```
<animate attributeName="top" from="0" to="10" dur="3s"
         repeatCount="2.5" />
```

In the following example, the animation function will repeat for a total of 7 seconds. It will play fully two times, followed by a fractional part of 2 seconds. This is equivalent to a repeatCount of 2.8. The last (partial) iteration will apply values in the range "0" to "8".

```
<animate attributeName="top" from="0" to="10" dur="2.5s"
         repeatDur="7s" />
```

In the following example, the simple duration is longer than the duration specified by repeatDur, and so the active duration will effectively cut short the simple duration. However, animation function still uses the specified simple duration. The effect of the animation is to interpolate the value of "top" from 10 to 15, over the course of 5 seconds.

```
<animate attributeName="top" from="10" to="20"
         dur="10s" repeatDur="5s" />
```

Note that if the simple duration is not defined (e.g. it is indefinite), repeat behavior is not defined (but repeatDur still defines the active duration). In the following example the simple duration is indefinite, and so the repeatCount is effectively ignored. Nevertheless, this is not considered an error: the active duration is also indefinite. The effect of the animation is to to just use the value for `f(0)`, setting the fill color to red for the remainder of the animate element's duration.

```
<animate attributeName="fill" from="red" to="blue" repeatCount="2" />
```

In the following example, the simple duration is indefinite, but the repeatDur still determines the active duration. The effect of the animation is to set the fill color to red for 10 seconds.

```
<animate attributeName="fill" from="red" to="blue" repeatDur="10s" />
```

In the following example, the simple duration is longer than the duration specified by repeatDur, and so the active duration will effectively cut short the simple duration. However, the animation function still interpolates using the specified simple duration. The effect of the animation is to interpolate the value of "top" from 10 to 17, over the course of 7 seconds.

```
<animate attributeName="top" from="10" to="20"
         dur="10s" repeatDur="7s" />
```

*Controlling behavior of repeating animation - Cumulative animation*

The author may also select whether a repeating animation should repeat the original behavior for each iteration, or whether it should build upon the previous results, accumulating with each iteration. For example, a motion path that describes an arc can repeat by moving along the same arc over and over again, or it can begin each repeat iteration where the last left off, making the animated element bounce across the window. This is called *cumulative* animation.

*Normative*

> Every animation element must be defined as either *cumulative* or *non-cumulative*. An animation element may be defined as cumulative only if addition is defined for the target attribute. The *cumulative animation function*, $f_c(t)$, for any simple animation function $f(t)$ is
>
> $f_c(t) = f_r(t)$, if the element is non-cumulative.
>
> If the element is cumulative:
>
> Let $f_i(t)$ represent the cumulative animation function for a given iteration $i$.
>
> The first iteration $f_0(t)$ is unaffected by accumulate, and so is the same as the original simple animation function definition. Each subsequent iteration adds to the result of the previous iterations:
>
> $f_0(t) = f(t)$
>
> $f_i(t) = (f(d) * i) + f(t - (i*d))$ for any integer $i > 0$.
>
> The cumulative animation function is then
>
> $f_c(t) = f_i(t)$, where $i = floor(t/d)$.

Note that $f_{i+1}(t)$starts at $f(d)*i + f(0)$. To avoid jumps, authors will typically choose animation functions which start at 0.

For example, the path notation for a simple arc (detailed in The animateMotion element) can be used to describe a bouncing motion:

```
<img ...>
  <animateMotion path="m 0 0 c 30 50 70 50 100 0 z" dur="5s"
     accumulate="sum" repeatCount="4" />
</img>
```

The image moves from the original position along the arc over the course of 5 seconds. As the animation repeats, it builds upon the previous value and begins the second arc where the first one ended, as illustrated in Figure 1, below. In this way, the image "bounces" across the screen. The same animation could be described as a complete path of 4 arcs, but in the general case the path description can get quite large and cumbersome to edit.

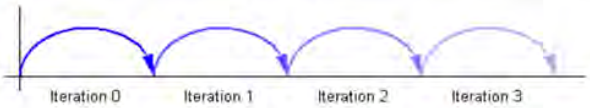



*Figure 1 - A cumulative repeating animation. Each repeat iteration builds upon the previous.*

Note that cumulative animation only controls how a single animation accumulates the results of the simple animation function as it repeats. It specifically does not control how one animation interacts with other animations to produce a presentation value. This latter behavior is described in the section Additive animation. Similarly, if an element restarts, the accumulate from the first run is not applied to the second. See Restarting animations.

Any numeric attribute that supports addition can support cumulative animation. For example, we can define a "pulsing" animation that will grow the "width" of an SVG `<rect>` element by 100 pixels in 50 seconds.

```
<rect width="20px" ...>
   <animate attributeName="width" dur="5s"
      values="0; 15; 10" additive="sum"
      accumulate="sum" repeatCount="10" />
</rect>
```

Each simple duration causes the rectangle width to bulge by 15 pixels and end up 10 pixels larger. The shape is 20 pixels wide at the beginning, and after 5 seconds is 30 pixels wide. The animation repeats, and builds upon the previous values. The shape will bulge to 45 pixels and then end up 40 pixels wide after 10 seconds, and will eventually end up 120 (20 + 100) pixels wide after all 10 repeats.

**Freezing animations**

Animation elements follow the definition of fill in the Timing module. This section extends that specification to cover animation-specific semantics.

By default when an animation element ends, its effect is no longer applied to the presentation value for the target attribute. For example, if an animation moves an image and the animation element ends, the image will "jump back" to its original position.

```
<img top="3" ...>
   <animate begin="5s" dur="10s" attributeName="top" by="100"/>
```

As shown in Figure 2, the image will appear stationary at the top value of "3" for 5 seconds, then move 100 pixels down in 10 seconds. 15 seconds after the document begin, the animation ends, the effect is no longer applied, and the image jumps back from 103 to 3 where it started (i.e. to the underlying DOM value of the `top` attribute).

***Figure 2 - Illustration of animation without freezing.***

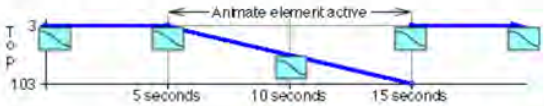


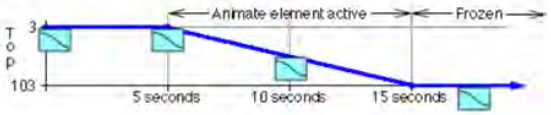

*Figure 2 - Simple animation without freezing. After the animate element ends, the effect of the animation is removed.*

The fill attribute can be used to maintain the value of the animation after the active duration of the animation element ends:

```
<img top="3" ...>
   <animate begin= "5s" dur="10s" attributeName="top" by="100"
         fill="freeze" />
</img>
```

The animation ends 15 seconds after the document begin, but the image remains at the top value of 103 (Figure 3). The attribute *freezes* the last value of the animation for the duration of the freeze effect. This duration is controlled by the time container (for details, see SMIL Timing and Synchronization).

***Figure 3 - Illustration of animation with `fill="freeze"`.***

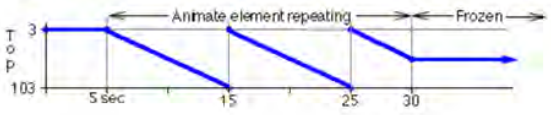



*of the animation is retained.*

If the active duration cuts short the simple duration (including the case of partial repeats), the effect value of a *frozen* animation is defined by the shortened simple duration. In the following example, the simple animation function repeats two full times and then again for one-half of the simple duration. In this case, the value while *frozen* will be 53:

```
<img top="3" ...>
   <animate begin= "5s" dur="10s" attributeName="top" by="100"
         repeatCount="2.5" fill="freeze" />
</img>
```

***Figure 4 - Illustration of animation combining a partial `repeat` and `fill="freeze"`.***



In the following example, the dur attribute is missing, and so the simple duration is indefinite. The active duration is constrained by end to be 10 seconds. Interpolation is not defined, and the value while *frozen* will be the from value, 10:

```
<animate from="10" to="20" end="10s" fill="freeze" .../>
```

Stating this formally:

*Normative*

The *frozen animation function*, $f_f(t)$, for an element with active duration AD, $0<=t<=AD$ is given by

$f_f(t) = f_c(t)$ for all times t: $0<=t<AD$

If AD is not an even multiple of the simple duration d,

$f_f(AD) = f_c(AD)$.

$f_f(AD) = f(d)$.

If AD is an even multiple of d, i.e. $AD = d*i$ for some positive integer i, and the animation is cumulative,

$f_f(AD) = f(d) * i$.

### 3.2.6 Additive animation

In addition to repeating and accumulating values of a single animation, an animation may be expressed as a delta to an attribute's value, rather than as an absolute value. This can be used in a single animation to modify the underlying DOM value, or complex animations can be produced by combining several simple ones.

For example, a simple "grow" animation can increase the width of an object by 10 pixels:

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
    additive="sum"/>
</rect>
```

The width begins at 20 pixels, and increases to 30 pixels over the course of 10 seconds. If the animation were declared to be non-additive, the same from and to values would make the width go from 0 to 10 pixels over 10 seconds.

Many complex animations are best expressed as combinations of simpler animations. A "vibrating" path, for example, can be described as a repeating up and down motion added to any other motion:

```
<img ...>
  <animateMotion from="0,0" to="100,0" dur="10s" />
  <animateMotion values="0,0; 0,5; 0,0" dur="1s"
                 repeatDur="10s" additive="sum"/>
</img>
```

The *animation effect function*, captures the semantics of this for a single animation element:

*Normative*

0<=t<=AD and u, a value of the type of the target attribute of the animation into that same type.

Every animation element must be defined as either *additive* or *non-additive*. An element may be defined as additive only if addition is defined for type type of the target attribute.

If the animation is additive, $F(t,u) = u + f_f(t)$.

If the animation is non-additive, $F(t,u) = f_f(t)$.

When there are multiple animations defined for a given attribute that overlap at any moment, the two either add together or one overrides the other. Animations overlap when they are both either active or frozen at the same moment. The ordering of animations (e.g. which animation overrides which) is determined by a priority associated with each animation. The animations are prioritized according to when each begins. The animation first begun has lowest priority and the most recently begun animation has highest priority.

Higher priority animations that are not additive will override all earlier (lower priority) animations, and simply set the attribute value. Animations that are additive apply (i.e. add to) to the result of the earlier-activated animations. For details on how animations are combined, see The animation sandwich model.

Additive animation is defined for numeric attributes and other data types for which an addition function is defined. This includes numeric attributes for concepts such as position, widths and heights, sizes, etc. This also includes color (refer to The animateColor element), and may include other data types as specified by the host language.

It is often useful to combine additive animations and fill behavior, for example when a series of motions are defined that should build upon one another:

```
<img ...>
  <animateMotion begin="0" dur="5s" path="[some path]"
          additive="sum" fill="freeze" />
  <animateMotion begin="5s" dur="5s" path="[some path]"
          additive="sum" fill="freeze" />
  <animateMotion begin="10s" dur="5s" path="[some path]"
          additive="sum" fill="freeze" />
</img>
```

The image moves along the first path, and then starts the second path from the end of the first, then follows the third path from the end of the second, and stays at the final point.

While many animations of numerical attributes will be additive, this is not always the case. As an example of an animation that is defined to be non-

additive, consider a hypothetical extension animation "mouseFollow" that causes an object to track the mouse.

```
<img ...>
  <animateMotion dur="10s" repeatDur="indefinite"
          path="[some nice path]" />
  <mouseFollow begin="mouseover" dur="5s"
          additive="replace" fill="remove" />
</img>
```

The mouse-tracking animation runs for 5 seconds every time the user mouses over the image. It cannot be additive, or it will just offset the motion path in some odd way. The `mouseFollow` needs to override the **animateMotion** while it is active. When the `mouseFollow` completes, its effect is no longer applied and the **animateMotion** again controls the presentation value for position.

In addition, some numeric attributes (e.g. a telephone number attribute) may not sensibly support addition. It is left to the host language to specify which attributes support additive animation. Attribute types for which addition is not defined, such as strings and Booleans, cannot support additive animation.

**Additive and Cumulative animation**

The accumulate attribute should not be confused with the additive attribute. The additive attribute defines how an animation is combined with other animations and the base value of the attribute. The accumulate attribute defines only how the simple animation function interacts with itself, across repeat iterations.

Typically, authors expect cumulative animations to be additive (as in the examples described for accumulate above), but this is not required. The following example is cumulative but not additive.

```
<img ...>
  <animate dur="10s" repeatDur="indefinite"
          attributeName="top" from="20" by="10"
          additive="replace" accumulate="sum" />
</img>
```

The animation overrides whatever original value was set for "top", and begins at the value 20. It moves down by 10 pixels to 30, then repeats. It is cumulative, so the second iteration starts at 50 (the value of 30 from the previous iteration plus the from value, 20) and moves down by another 10 to 60, and so on.

When a cumulative animation is also defined to be additive, the two features function normally. The accumulated effect for `F(t,u)` is used as the value for the animation, and is added to the underlying value for the target attribute. For example:

```
<img top="10" ... >
  <animate dur="10s" repeatdur="indefinite"

          attributename="top" from="20" by="10"

          additive="sum" accumulate="sum" />
</img>
```

The animation adds to the original value of 10 that was set for "top", and begins at the value 30. It moves down by 10 pixels to 40, then repeats. It is cumulative, so the second iteration starts at 60 (the value of 40 from the previous iteration plus 20) and moves down by another 10 to 70, and so on.

Refer also to The animation sandwich model.

### 3.2.7 Restarting animations

Animation elements follow the definition of restart in the SMIL Timing module. This section is descriptive.

When an animation restarts, the defining semantic is that it behaves as though this were the first time the animation had begun, independent of any earlier behavior. The animation effect function `F(t,u)` is defined independent of the restart behavior. Any effect of an animation playing earlier is no longer applied, and only the current animation effect `F(t,u)` is applied.

If an additive animation is restarted while it is active or frozen, the previous effect of the animation (i.e. before the restart) is no longer applied to the attribute. Note in particular that cumulative animation is defined only within the active duration of an animation. When an animation restarts, all accumulated context is discarded, and the animation effect `F(t,u)` begins accumulating again from the first iteration of the restarted active duration.

### 3.2.8 Animation function value details

Many animations specify the simple animation function `f(t)` as a sequence of values to be applied over time. For some types of attributes (e.g. numbers), it is also possible to describe an interpolation function between values.

As a simple form of describing the values, animation elements can specify a *from* value and a *to* value. If the attribute takes values that support interpolation (e.g. a number), the simple animation function can interpolate values in the range defined by *from* and *to*, over the course of the simple duration. A variant on this uses a *by* value in place of the *to* value, to indicate an additive change to the attribute.

More complex forms specify a list of values, or even a path description for motion. Authors can also control the timing of the values, to describe "keyframe" animations, and even more complex functions.

In all cases, the animation effect function, `F(t,u)`, must yield legal values for the target attribute. Three classes of values are described:

1. **Unitless scalar values**. These are simple scalar values that can be parsed and set without semantic constraints. This class includes integers (base 10) and floating point (format specified by the host language).
2. **String values**. These are simple strings.
3. **Language abstract values**. These are values like CSS-length and CSS-angle values that have more complex parsing, but that can yield values that may be interpolated.

The animate element can interpolate unitless scalar values, and both animate and set elements can handle String values without any semantic knowledge of the target element or attribute. The animate and set elements must support unitless scalar values and string values. The host language must define which additional language abstract values should be handled by these elements. Note that the animateColor element implicitly handles the abstract values for color values, and that the animateMotion element implicitly handles position and path values.

In order to support interpolation on attributes that define numeric values with some sort of units or qualifiers (e.g. "10px", "2.3feet", "$2.99"), some additional support is required to parse and interpolate these values. One possibility is to require that the animation framework have built-in knowledge of the unit-qualified value types. However, this violates the principle of encapsulation and does not scale beyond CSS to XML languages that define

new attribute value types of this form.

The recommended approach is for the animation implementation for a given host environment to support two interfaces that abstract the handling of the language abstract values. These interfaces are not formally specified, but are simply described as follows:

1. The first interface converts a string (the animation function value) to a unitless, canonical number (either an integer or a floating point value). This allows animation elements to interpolate between values without requiring specific knowledge of data types like CSS-length. The interface will likely require a reference to the target attribute, to determine the legal abstract values. If the passed string cannot be converted to a unitless scalar, the animation element will treat the animation function values as strings, and the calcMode will default to "discrete".
2. The second interface converts a unitless canonical number to a legal string value for the target attribute. This may, for example, simply convert the number to a string and append a suffix for the canonical units. The animation element uses the result of this to actually set the presentation value.

Support for these two interfaces ensures that an animation engine need not replicate the parser and any additional semantic logic associated with language abstract values.

This is not an attempt to specify how an implementation provides this support, but rather a requirement for how values are interpreted. Animation behaviors should not have to understand and be able to convert among all the CSS-length units, for example. In addition, this mechanism allows for application of animation to new XML languages, if the implementation for a language can provide parsing and conversion support for attribute values.

The above recommendations notwithstanding, it is sometimes useful to interpolate values in a specific unit-space, and to apply the result using the specified units rather than canonical units. This is especially true for certain relative units such as those defined by CSS (e.g. em units). If an animation specifies all the values in the same units, an implementation may use knowledge of the associated syntax to interpolate in the unit space, and apply the result within the animation sandwich, in terms of the specified units rather than canonical units. As noted above, this solution does not scale well to the general case. Nevertheless, in certain applications (such as CSS properties), it may be desirable to take this approach.

**Interpolation and indefinite simple durations**

If the simple duration of an animation is indefinite (e.g. if no dur value is specified), interpolation is not generally meaningful. While it is possible to define an animation function that is not based upon a defined simple duration (e.g. some random number algorithm), most animations define the function in terms of the simple duration. If an animation function is defined in terms of the simple duration and the simple duration is indefinite, the first value of the animation function (i.e. `f(0)`) should be used (effectively as a constant) for the animation function.

## 3.3 Overview of the SMIL 2.0 BasicAnimation Module

The SMIL 2.0 BasicAnimation module provides

- Syntax for identifying target elements and attributes,
- Simple animation functions defined by sets of points to be visited in sequence, with the function defined between those using discrete, linear or paced interpolation.
- The animate element for generic animation.
- The set element for simply setting the value of the target attribute to a constant value
- The animateMotion element for defining motion on a path. And
- The animateColor element for defining color changes over time.

The BasicAnimation module defines attributes and elements following the model presented in the Animation Model section.

## 3.4 SMIL 2.0 BasicAnimation Module Common Attributes

The elements of the BasicAnimation module have in common the attributes used to identify the target attribute and, less universally, the attributes by which the animation functions are specified.

### 3.4.1 Specifying the animation target

The animation target is defined as a specific attribute of a particular element. The means of specifying the target attribute and the target element are detailed in this section.

**The target attribute**

The target attribute to be animated is specified with attributeName. The value of this attribute is a string that specifies the name of the target attribute, as defined in the host language.

The attributes of an element that can be animated are often defined by different languages, and/or in different namespaces. For example, in many XML applications, the position of an element (which is a typical target attribute) is defined as a CSS property rather than as XML attributes. In some cases, the same attribute name is associated with attributes or properties in more than one language, or namespace. To allow the author to disambiguate the name mapping, an additional attribute attributeType is provided that specifies the intended namespace.

The attributeType attribute is optional. By default, the animation runtime will resolve the names according to the following rule: If there is a name conflict and attributeType is not specified, the list of CSS properties supported by the host language is matched first (if CSS is supported in the host language); if no CSS match is made (or CSS does not apply) the default namespace for the target element will be matched.

If a target attribute is defined in an XML Namespace other than the default namespace for the target element, the author must specify the namespace of the target attribute using the associated namespace prefix as defined in the scope of the animation element. The prefix is prepended to the value for attributeName.

For more information on XML namespaces, see [XML-NS].

**Target attribute attributes**

Specifies the name of the target attribute. An xmlns prefix may be used to indicate the XML namespace for the attribute [XML-NS]. The prefix will be interpreted in the scope of the animation element.

attributeType

Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following (values are case-sensitive):

`CSS`
: This specifies that the value of attributeName is the name of a CSS property, as defined for the host document. This argument value is only meaningful in host language environments that support CSS.

`XML`
: This specifies that the value of "attributeName" is the name of an XML attribute defined in the default XML namespace for the target element. Note that if the value for attributeName has an XMLNS prefix, the implementation must use the associated namespace as defined in the scope of the animation element.

`auto`
: The implementation should match the attributeName to an attribute for the target element. The implementation must first search through the list of CSS properties for a matching property name, and if none is found, search the default XML namespace for the element. This is the default.

**The target element**

An animation element can define the target element of the animation either explicitly or implicitly. An explicit definition uses an attribute to specify the target element. The syntax for this is described below.

If no explicit target is specified, the implicit target element is the parent element of the animation element in the document tree. It is expected that the common case will be that an animation element is declared as a child of the element to be animated. In this case, no explicit target need be specified.

If an explicit target element reference cannot be resolved (e.g. if no such element can be found), the animation has no effect. In addition, if the target element (either implicit or explicit) does not support the specified target attribute, the animation has no effect. See also Handling syntax errors.

The following two attributes can be used to identify the target element explicitly:

**Target element attributes**

targetElement
: This attribute specifies the target element to be animated. The attribute value must be the value of an XML identifier attribute of an element (i.e. an "IDREF") within the host document. For a formal definition of IDREF, refer to XML 1.0 [XML10].

href
: This attribute specifies the target element to be animated. The attribute value must be an XLink locator, referring to the target element to be animated.

When integrating animation elements into the host language, the language designer should avoid including both of these attributes. If however, the host language designer chooses to include both attributes in the host language, then when both are specified for a given animation element the XLink href attribute takes precedence over the targetElement attribute.

The advantage of using the targetElement attribute is the simpler syntax of the attribute value compared to the href attribute. The advantage of using the XLink href attribute is that it is extensible to a full linking mechanism in future versions of SMIL Animation, and the animation element can be processed by generic XLink processors. The XLink form is also provided for host languages that are designed to use XLink for all such references. The following two examples illustrate the two approaches.

This example uses the simpler targetElement syntax:

```
<animate targetElement="foo" attributeName="bar" .../>
```

This example uses the more flexible XLink locater syntax, with the equivalent target:

```
<foo xmlns:xlink="http://www.w3.org/1999/xlink">
  ...
  <animate xlink:href="#foo" attributeName="bar" .../>
  ...
</foo>
```

When using an XLink href attribute on an animation element, the following additional XLink attributes need to be defined in the host language. These may be defined in a DTD, or the host language may require these in the document syntax to support generic XLink processors. For more information, refer to [XLINK].

The following XLink attributes are required by the XLink specification. The values are fixed, and so may be specified as such in a DTD. All other XLink attributes are optional, and do not affect SMIL Animation semantics.

**XLink attributes for href**

type
: Must be `simple`. Identifies the type of XLink being used.

actuate
: Must be `onLoad`. Indicates that the link to the target element is followed automatically (i.e., without user action).

show
: Must be `embed`. Indicates that the reference does not include additional content in the file.

Additional details on the target element specification as relates to the host document and language are described in Required definitions and constraints on animation targets.

### 3.4.2 Specifying the simple animation function f(t)

Every animation function defines the value of the attribute at a particular moment in time. The time range for which the animation function is defined is the simple duration. The animation function does not produce defined results for times outside the range of 0 to the simple duration.

An animation is described either as a list of *values*, or in a simplified form that describes the *from*, *to* and *by* values. The from/to/by form is defined in Simple animation functions defined by from, to and by.

**Simple animation function attributes**

values

values

A semicolon-separated list of one or more values, each of which must be a legal value for the specified attribute. Vector-valued attributes are supported using the vector syntax of the attributeType domain. Leading and trailing white space, and white space before and after semi-colon separators, will be ignored.

If any values are not legal, the animation will have no effect (see also Handling Syntax Errors).

calcMode

Specifies the interpolation mode for the animation. If the target attribute does not support linear interpolation (e.g. for strings), or if the values attribute has only one value, the calcMode attribute is ignored and `discrete` interpolation is used. The calcMode attribute can take any of the following values:

`discrete`
This specifies that the animation function will jump from one value to the next without any interpolation.

`linear`
Simple linear interpolation between values is used to calculate the animation function.
This is the default.

`paced`
Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.).

The animation will apply the values in order over the course of the animation. For `discrete` and `linear` animations, values in the values attribute are equally spaced through the animation duration. For `paced` animations, the values are spaced so that a uniform rate of change is obtained.

The following example using the values syntax animates the width of an SVG shape over the course of 10 seconds, interpolating from a width of 40 to a width of 100 and back to 40.

```
<rect ...>
  <animate attributeName="width" values="40;100;40" dur="10s"/>
</rect>
```

The simple animation function for this example (with time in seconds) is

`f(t) = 40 + 60*t/5,  0 <= t < 5`, and
`f(t) = 100 - 60*(t-5)/5,  5 <= t <= 10`.

The simple animation function defined by the values and calcMode attributes can be formally specified:

*Normative*

With `i = floor((t*n)/d)`, `d` the simple duration of the animation element, `n` the number of entries in the values attribute and `value[i]` the $i^{th}$ entry (counting from 0):

- For `discrete` animation, the duration is divided into `n` equal time periods, one per value. The animation function takes on the values in order, one value for each time period:

  `f(t) = value[i]`

  associated times:

  $f(t) = value[i] + (value[i+1]-value[i]) * (t-t_i)/d$.

- For `paced` animations, the duration is divided into periods of lengths such that the rate of change of the attribute remains constant. If the distance between two values is $dist(v_1,v_2)$, the total distance traversed `D(i)` up to and including `value[i]` is

  $D(0) = 0$, and

  $D(i) = dist(value[0],value[1]) + dist(value[1],value[2]) + ... + dist(value[i-1],value[i])$, for integers $i$ with $0<i<=n$.

  The animation function takes on the values in the values attribute at times determined by these distances:

  $T(i) = (D(i)/D(n)) * d$, for integers $i$ with $0<=i<=n$.

  $f(t) = value[i] + (value[i+1]-value[i]) * (t-T(i))/d$,

  where $i$ is the largest non-negative integer such that $T(i)<=t$.

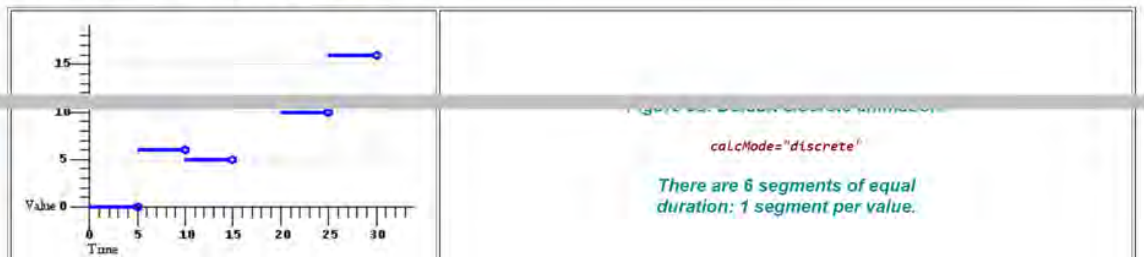
Note that a `linear` or `paced` animation will be a smoothly closed loop if the first value is repeated as the last.

**Interpolation modes illustrated**

The three figures 5a, 5b and 5c below show how the same basic animation will change a value over time, given different interpolation modes. All examples are based upon the following example, but with different values for calcMode:

```
<animate dur="30s" values="0; 6; 5; 11; 10; 16" calcMode="[as specified]" />
```

***Figure 5 - Discrete, linear and paced animation***



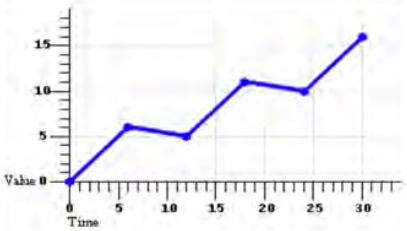



*Figure 5b: Default linear animation.*

*calcMode="linear"*

*There are 5 segments of equal duration: n-1 segments for n values.*

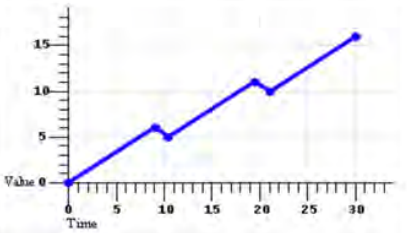



*Figure 5c: Default paced animation.*

*calcMode="paced"*

*There are 5 segments of varying duration: n-1 segments for n values, computed to yield a constant rate of change in the value.*

**Examples of calcMode**

The following example describes a simple discrete animation:

```
<animate attributeName="foo" dur="8s"
    values="bar; fun; far; boo" />
```

The value of the attribute "foo" will be set to each of the four strings for 2 seconds each. Because the string values cannot be interpolated, only discrete animation is possible; any calcMode attribute would be ignored.

The following example describes a simple linear animation:

```
<animate attributeName="x" dur="10s" values="0; 10; 100"
    calcMode="linear"/>
```

The value of "x" will change from 0 to 10 in the first 5 seconds, and then from 10 to 100 in the second 5 seconds. Note that the values in the values attribute are spaced evenly in time; in this case the result is a much larger actual change in the value during the second half of the animation. Contrast this with the same example changed to use "paced" interpolation:

```
<animate attributeName="x" dur="10s" values="0; 10; 100"
    calcMode="paced"/>
```

To produce an even pace of change to the attribute "x", the second segment defined by the values list gets most of the simple duration: The value of "x" will change from 0 to 10 in the first second, and then from 10 to 100 in the next 9 seconds. While this example could be easily authored as a *from-to animation* without paced interpolation, many examples (such as motion paths) are much harder to author without the `paced` value for calcMode.

### 3.4.3 Specifying the animation effect function F(t,u)

As described in The animation effect function F(t,u), the simple animation function may be

- Repeated or frozen, using the attributes repeatCount, repeatDur and fill of the BasicInlineTiming module.
- Defined as cumulative or non-cumulative when repeated.
- Defined as additive or non-additive when combined with the underlying value.

The animation effect function F(t,u) defines the semantics of these attributes, and give examples. This section gives only the syntax.

See the BasicInlineTiming module for definitions of the attributes repeatCount, repeatDur and fill.

The additive and cumulative behavior of repeating animations is controlled with the additive and accumulate attributes, respectively:

**Animation effect function attributes**

accumulate
: Controls whether or not the animation is cumulative. May be either of the following two values:

  `sum`
  : Specifies that the animation is *cumulative*, i.e. each repeat iteration after the first builds upon the last value of the previous iteration.

  `none`
  : Specifies that the animation is *non-cumulative*, i.e. repeat iterations simply repeat the animation function `f(t)`. This is the default.

  This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.

additive
: Controls whether or not the animation is additive.

  `sum`
  : Specifies that the animation is *additive*, i.e. will add to the underlying value of the attribute and other lower priority animations.

  `replace`
  : Specifies that the animation is *non-additive*, i.e. will override the underlying value of the attribute and other lower priority animations. This is the default.

  This attribute is ignored if the target attribute does not support additive animation.

### 3.4.4 Simple animation functions specified by from, to and by

An animation is described either as a list of *values*, as described earlier, or in a simplified form that uses *from*, *to* and *by* values.

**From/to/by attributes for simple animation functions**

from
: Specifies the starting value of the animation. Must be a legal value for the specified attribute. Ignored if the values attribute is specified.

to
: Specifies the ending value of the animation. Must be a legal value for the specified attribute. Ignored if the values attribute is specified.

by
: Specifies a relative offset value for the animation. Must be a legal value of a domain for which addition to the attributeType domain is defined and which yields a value in the attributeType domain. Ignored if the values attribute is specified.

The simpler *from*/*to*/by syntax provides for several variants. To use one of these variants, one of by or to must be specified; a from value is optional. It is not legal to specify both by and to attributes; if both are specified, only the to attribute will be used (the by will be ignored). The combinations of attributes yield the following classes of animation.

***from-to animation***
: Specifying a `from` value and a `to` value defines a simple animation. The animation function is defined to start with the `from` value, and to finish with the `to` value.

  *Normative*: A *from-to animation* with a `from` value $v_f$ and a `to` value $v_t$ is equivalent to the same animation with a `values` list with 2 values, $v_f$ and $v_t$.

***from-by animation***
: Specifying a `from` value and a `by` value defines a simple animation in which the animation function is defined to start with the `from` value, and to change this over the course of the simple duration by a *delta* specified with the `by` attribute. This may only be used with attributes that support addition (e.g. most numeric attributes).

  *Normative*: A *from-by* animation with a `from` value $v_f$ and a `by` value $v_b$ is equivalent to the same animation with a `values` list with 2 values, $v_f$ and $(v_f+v_b)$.

***by animation***
: Specifying only a by value defines a simple animation in which the animation function is defined to offset the underlying value for the attribute, using a delta that varies over the course of the simple duration, starting from a delta of 0 and ending with the delta specified with the `by` attribute. This may only be used with attributes that support additive animation.

  *Normative*: A *by* animation with a `by` value $v_b$ is equivalent to the same animation with a `values` list with 2 values, `0` and $v_b$, and `additive="sum"`. Any other specification of the `additive` attribute in a *by* animation is ignored.

***to animation***
: This describes an animation in which the animation function is defined to start with the underlying value for the attribute, and finish with the value specified with the `to` attribute. Using this form, an author can describe an animation that will start with any current value for the attribute, and will end up at the desired `to` value.

  A normative definition of a *to animation* is given below in *To animation*

**Examples**

The following "*from-to animation*" example animates the width of an SVG shape over the course of 10 seconds from a width of 50 to a width of 100.

```
<rect ...>
   <animate attributeName="width" from="50" to="100" dur="10s"/>
</rect>
```

The following "*from-by animation*" example animates the width of an SVG shape over the course of 10 seconds from a width of 50 to a width of 75.

```
<rect ...>
   <animate attributeName="width" from="50" by="25" dur="10s"/>
</rect>
```

The following "*by animation*" example animates the width of an SVG shape over the course of 10 seconds from the original width of 40 to a width of 70.

```
<rect width="40"...>
   <animate attributeName="width" by="30" dur="10s"/>
</rect>
```

*From-to* and *from-by* animations also support cumulative animation, as in the following example:

```
<rect width="20px"...>
   <animate attributeName="width" dur="5s" from="10px" to="20px"
      accumulate="sum" repeatCount="10" />
</rect>
```

The rectangle will grow from 10 to 20 pixels in the first 5 seconds, and then from 20 to 30 in the next 5 seconds, and so on up to 110 pixels after 10 repeats. Note that since the default value for additive is **`replace`**, the original value is ignored. The following example makes the animation explicitly additive:

```
<rect width="20px"...>
   <animate attributeName="width" dur="5s" from="10px" to="20px"
      accumulate="sum" additive="sum" repeatCount="10" />
</rect>
```

The results are the same as before, except that all the values are shifted up by the original value of 20. The rectangle is 30 pixels wide after 5 seconds, and 130 pixels wide after 10 repeats.

A *to animation* of an attribute which supports addition is a kind of mix of additive and non-additive animation. The underlying value is used as a starting point as with additive animation, however the ending value specified by the to attribute overrides the underlying value as though the animation was non-additive.

The following "*to animation*" example animates the width of an SVG shape over the course of 10 seconds from the original width of 40 to a width of 100.

```
<rect width="40"...>
```

```
</rect>
```

Since a *to animation* has only 1 value, a discrete *to animation* will simply set the to value for the simple duration. In the following example, the rect will be blue for the 10 second duration of the animate element.

```
<rect color="red"...>
  <animate attributeName="color" to="blue" dur="10s" calcMode="discrete"/>
</rect>
```

The semantics of *to animation* fit into the general animation model, but with a few special cases. The normative definition given here parallels the definition for other types of animation presented in the Animation Model section.

*Normative*

The simple animation function $f(t,u)$ for a *to animation* with `to` value $v_t$ is a linear interpolation between the underlying value, $u$, and the `to` value:

$f(t,u) = (u * (d-t)/d) + (v_t * t/d)$, for $t: 0<=t<=d$ where $d$ is the simple duration.

If no other (lower priority) animations are active or frozen, this defines simple interpolation. However if another animation is manipulating the underlying value, the *to animation* will initially add to the effect of the lower priority animation, and increasingly dominate it as it nears the end of the simple duration, eventually overriding it completely. The value for $f(t,u)$ at the end of the simple duration is just the `to` value.

Repeating *to animations* is the same as repeating other animations:

*Normative*

The *repeated animation function*, $f_r(t,u)$, has the standard definition:

$f_r(t,u) = f( \text{REMAINDER}(t,d), u )$.

Because *to animation* is defined in terms of absolute values of the target attribute, cumulative animation is not defined:

*Normative*

The *cumulative animation function*, $f_c(t)$, for a *to animation* is

$f_c(t,u) = f_r(t,u)$.

A frozen *to animation* takes on the value at the time it is frozen, masking further changes in the underlying value. This matches the dominance of the `to` value at the end of the simple duration. Even if other, lower priority animations are active while a *to animation* is frozen, the value does not change.

*Normative*

The *frozen animation function*, $f_f(t)$, for a *to animation* is

$f_f(t,u) = f_c(t,u)$, if the animation is not frozen at time $t$, and

$f_f(t,u) = v_f$, if the animation is frozen at time $t$, where $v_f$ is the value of $f_f(t,u)$ at the moment the animation was frozen.

For example, consider

```
<rect width="40"...>
  <animate attributeName="width" to="100" dur="10s" repeatCount="2.5" fill="freeze"/>
</rect>
```

The width will animate from 40 to 100 pixels in the first 10 seconds, repeat 40 to 100 in the second 10 seconds, go from 40 to 70 in the final 5 seconds, and freeze at 70.

*To animation* defines its own kind of additive semantics, so the additive attribute is ignored.

The *animation effect function*, $F(t,u)$ for a *to animation* is

$F(t,u) = f_f(t,u)$.

Multiple *to animations* will also combine according to these semantics. As the animation progresses, the higher-priority animation will have greater and greater effect, and the end result will be to set the attribute to the final value of the higher-priority *to animation*.

For an example of additive *to animation*, consider the following two additive animations. The first, a *by-animation* applies a delta to attribute "x" from 0 to -10. The second, a *to animation* animates to a final value of 10.

```
<foo x="0" ...>
  <animate id="A1" attributeName="x"
      by="-10" dur="10s" fill="freeze" />
  <animate id="A2" attributeName="x"
      to="10"  dur="10s" fill="freeze" />
</foo>
```

The presentation value for "x" in the example above, over the course of the 10 seconds is presented in Figure 6 below. These values are simply computed using the formula described above. Note that the value for $F(t,u)$ for A2 is the presentation value for "x", since A2 is the higher-priority animation.

***Figure 6 - Effect of Additive to animation example***

| Time | $F(t,u)$ for A1 | $F(t,u)$ for A2 |
|---|---|---|
| 0 | 0 | 0 |

| | | |
|---|---|---|
| 1 | -1 | 0.1 |
| 2 | -2 | 0.4 |
| 3 | -3 | 0.9 |
| 4 | -4 | 1.6 |
| 5 | -5 | 2.5 |
| 6 | -6 | 3.6 |
| 7 | -7 | 4.9 |
| 8 | -8 | 6.4 |
| 9 | -9 | 8.1 |
| 10 | -10 | 10 |

## 3.5 SMIL 2.0 BasicAnimation Elements

The SMIL BasicAnimation module defines four elements, animate, set, animateMotion and animateColor.

### 3.5.1 The animate element

The animate element introduces a generic attribute animation that requires little or no semantic understanding of the attribute being animated. It can animate numeric scalars as well as numeric vectors. It can also animate a single non-numeric attribute through a discrete set of values. The animate element is an empty element; it cannot have child elements.

This element supports from/to/by and values descriptions for the animation function, as well as all of the calculation modes. It supports all the described timing attributes. These are all described in respective sections above.

**Element attributes**

attributeName **and**
attributeType
: The attribute to be animated. See The target attribute. attributeName is required; attributeType is optional.

targetElement,
href,
actuate,
show, **and**
type
: The target element. See The target element. All are optional.

from,
to,
by,
values,
calcMode,
accumulate, **and**
additive

: Specify the animation function and effect. See Specifying the simple animation function f(t) and Specifying the animation effect F(t,u).

Numerous examples are provided above, as are normative definitions of the semantics of all attributes supported by animate.

### 3.5.2 The set element

The set element provides a simple means of just setting the value of an attribute for a specified duration. As with all animation elements, this only manipulates the presentation value, and when the animation completes, the effect is no longer applied. That is, set does not *permanently* set the value of the attribute.

The set element supports all attribute types, including those that cannot reasonably be interpolated and that more sensibly support semantics of simply setting a value (e.g. strings and Boolean values). The set element is non-additive. The additive and accumulate attributes are not allowed, and will be ignored if specified.

The set element supports all the timing attributes to specify the simple and active durations. However, the repeatCount and repeatDur attributes will just affect the active duration of the set, extending the effect of the set (since it is not really meaningful to "repeat" a static operation). Note that using

The set element supports a more restricted set of attributes than the animate element. Only one value is specified, and neither interpolation control nor additive or cumulative animation is supported:

**Element attributes**

attributeName **and**
attributeType
: The attribute to be animated. See The target attribute. attributeName is required; attributeType is optional.

targetElement,
href,

actuate,
show, and
type
The target element. See The target element. All are optional.

to
Specifies the value for the target attribute during the active duration of the set element. The argument value must match the target attribute type.

*Normative*

The simple animation function defined by a set element is

```
f(t) = v
```

were v is the value of the to attribute.

The set element is non-cumulative and non-additive.

**Examples**

The following changes the stroke-width of an SVG rectangle from the original value to 5 pixels wide. The effect begins at 5 seconds and lasts for 10 seconds, after which the original value is again used.

```
<rect>
  <set attributeName="stroke-width" to="5px"
       begin="5s" dur="10s" fill="remove" />
</rect>
```

The following example sets the `class` attribute of the text element to the string "highlight" when the mouse moves over the element, and removes the effect when the mouse moves off the element.

```
<text>This will highlight if you mouse over it...
  <set attributeName="class" to="highlight"
       begin="mouseover" end="mouseout" />
</text>
```

### 3.5.3 The animateMotion element

The **animateMotion** element will move an element along a path. The element abstracts the notion of motion and position across a variety of layout mechanisms - the host language defines the layout model and must specify the precise semantics of position and motion. The path can be described in either of two ways:

- Specifying x,y pairs for the from/to/by attributes. These will define a straight line motion path.
- Specifying x,y pairs for the values attribute. This will define a motion path of straight line segments, or points (if calcMode is set to discrete). This will override any from/to/by attribute values.

All values must be x, y value pairs. Each x and y value may specify any units supported for element positioning by the host language. The host language defines the default units. In addition, the host language defines the *reference point* for positioning an element. This is the point within the element that is aligned to the position described by the motion animation. The reference point defaults in some languages to the upper left corner of the element bounding box; in other languages the reference point may be implicit, or may be specified for an element.

The syntax for the x, y value pairs is:

```
coordinate-pair ::= ( coordinate comma-wsp coordinate )
coordinate      ::= num
num             ::= Number
```

Coordinate values are separated by at least one white space character or a comma. Additional white space around the separator is allowed. The values of `coordinate` must be defined as some sort of number in the host language.

The attributeName and attributeType attributes are not used with **animateMotion**, as the manipulated position attribute(s) are defined by the host language. If the position is exposed as an attribute or attributes that can also be animated (e.g. as "top" and "left", or "posX" and "posY"), implementations must combine **animateMotion** animations with other animations that manipulate individual position attributes. See also The animation sandwich model.

If none of the from, to, by and values attributes are specified, the animation will have no effect.

The default calculation mode (calcMode) for **animateMotion** is **paced**. This will produce constant velocity motion along the specified path. Note that while animateMotion elements can be additive, the addition of two or more **paced** (constant velocity) animations may not result in a combined motion animation with constant velocity.

**Element attributes**

targetElement,
href,
actuate,
show, and
type
The target element. See The target element. All are optional.

from,
to,
by,
values,
accumulate, and
additive
Specify the animation function and effect. See Specifying the simple animation function f(t) and Specifying the animation effect F(t,u).

calcMode
Defined as above in Specifying the simple animation function f(t), but note that the default calcMode for **animateMotion** is **paced**. This will produce constant velocity motion.

The use of `linear` for the calcMode with more than 2 points described in the values attribute may result in motion with varying velocity. The `linear` calcMode specifies that time is evenly divided among the segments defined by the values. The use of `linear` does not specify that time is divided evenly according to the *distance* described by each segment.

For motion with constant velocity, calcMode should be set to `paced`.

origin
: Specifies the origin of motion for the animation. The values and semantics of this attribute are dependent upon the layout and positioning model of the host language. In some languages, there may be only one option, `default`. However, in CSS positioning for example, it is possible to specify a motion path relative to the container block, or to the layout position of the element. It is often useful to describe motion relative to the position of the element as it is laid out (e.g. from off screen left to the layout position, specified as `from="(-100,0)"` and `to="(0,0)"`. Authors must be able to describe motion both in this manner, as well as relative to the container block. The origin attribute supports this distinction. Nevertheless, because the host language defines the layout model, the host language must also specify the "default" behavior, as well as any additional attribute values that are supported.
Note that the definition of the layout model in the host language specifies whether containers have bounds, and the behavior when an element is moved outside the bounds of the layout container. In CSS2 [CSS2], for example, this can be controlled with the "clip" property.
Note that for additive animation, the origin distinction is not meaningful. This attribute only applies when additive is set to `replace`.

### 3.5.4 The animateColor element

The **animateColor** element specifies an animation of a color attribute. The host language must specify those attributes that describe color values and can support color animation.

All values must represent [sRGB] color values. Legal value syntax for attribute values is defined by the host language.

Interpolation is defined on a per-color-channel basis.

**Element attributes**

attributeName **and**
attributeType
: The attribute to be animated. See The target attribute. attributeName is required; attributeType is optional.

targetElement,
href,
actuate,
show, **and**
type

from,
to,
by,
values,
calcMode,
accumulate, **and**
additive

: Specify the animation function and effect. See Specifying the simple animation function f(t), Specifying the animation effect F(t,u).

The values in the from/to/by and values attributes may specify negative and out of gamut values for colors. The function defined by an individual **animateColor** may yield negative or out of gamut values. The implementation must correct the resulting presentation value, to be legal for the destination (display) colorspace. However, as described in The animation sandwich model, the implementation should only correct the final combined result of all animations for a given attribute, and should not correct the effect of individual animations.

Values are corrected by "clamping" the values to the correct range. Values less than the minimum allowed value are clamped to the minimum value (commonly 0, but not necessarily so for some color profiles). Values greater than the defined maximum are clamped to the maximum value (defined by the host language) .

Note that color values are corrected by clamping them to the gamut of the destination (display) colorspace. Some implementations may be unable to process values which are outside the source (sRGB) colorspace and must thus perform clamping to the source colorspace, then convert to the destination colorspace and clamp to its gamut. The point is to distinguish between the source and destination gamuts; to clamp as late as possible, and to realize that some devices, such as inkjet printers which appear to be RGB devices, have non-cubical gamuts.

Note to implementers: When **animateColor** is specified as a *to animation*, the animation function should assume Euclidean RGB-cube distance where deltas must be computed. See also Specifying the simple animation function f(t) and Simple animation functions specified by from, to and by. Similarly, when the calcMode attribute for **animateColor** is set to `paced`, the animation function should assume Euclidean RGB-cube distance to compute the distance and pacing.

## 3.6 SMIL 2.0 BasicAnimation Module Details

### 3.6.1 BasicAnimation integration requirements

This section describes what a language designer must actually do to specify the integration of SMIL Animation into a host language. This includes basic definitions and constraints upon animation.

In addition to the requirements listed in this section, those listed in Common animation integration requirements must be satisfied.

**Required definitions and constraints on animation targets**

*Specifying the target element*

The host language designer must choose whether to support the targetElement attribute or the XLink attributes for specifying the target element. Note that if the XLink syntax is used, the host language designer must decide how to denote the XLink namespace for the associated attributes. The namespace can be fixed in a DTD, or the language designer can require colonized attribute names (*qnames*) to denote the XLink namespace for the

attributes. The required XLink attributes have fixed values, and so may also be specified in a DTD, or can be required on the animation elements. Host language designers may require that the optional XLink attributes be specified. These decisions are left to the host language designer - the syntax details for XLink attributes do not affect the semantics of SMIL Animation.

In general, target elements may be any element in the document. Host language designers must specify any exceptions to this. Host language designers are discouraged from allowing animation elements to target elements outside of the document in which the animation element is defined. The XLink syntax for the target element could allow this, but the SMIL timing and animation semantics of this are not defined in this version of SMIL Animation.

*Target attribute issues*

The definitions in this module can be used to animate any attribute of any element in a host document. However, it is expected that host language designers integrating SMIL Animation may choose to constrain which elements and attributes can support animation. For example, a host language may choose not to support animation of the `language` attribute of a `script` element. A host language which included a specification for DOM functionality might limit animation to the attributes which may legally be modified through the DOM.

Any attribute of any element not specifically excluded from animation by the host language may be animated, as long as the underlying data type (as defined by the host language for the attribute) supports discrete values (for discrete animation) and/or addition (for interpolated, additive and cumulative animation).

All constraints upon animation must be described in the host language specification or in an appropriate schema, as the DTD alone cannot reasonably express this.

The host language must define which language abstract values should be handled for animated attributes. For example, a host language that incorporates CSS may require that CSS length values be supported. This is further detailed in Animation function value details.

The host language must specify the interpretation of relative values. For example, if a value is specified as a percentage of the size of a container, the host language must specify whether this value will be dynamically interpreted as the container size is animated.

The host language must specify the semantics of clamping values for attributes. The language must specify any defined ranges for values, and how out of range values will be handled.

The host language must specify the formats supported for numeric attribute values. This includes both integer values and floating point values. As a reasonable minimum, host language designers are encouraged to support the format described in section 4.3.1, "Integers and real numbers," of [CSS2].

**Integrating animateMotion functionality**

The host language specification must define which elements can be the target of **animateMotion**. In addition, the host language specification must describe the positioning model for elements, and must describe the model for **animateMotion** in this context (i.e. the semantics of the `default` value for the origin attribute must be defined). If there are different ways to describe position, additional attribute values for the origin attribute should be defined to allow authors control over the positioning model.

### 3.6.2 Document type definition (DTD) for the BasicAnimation module

See the full DTD for the SMIL Animation Modules.

## 3.7 Overview of the SMIL 2.0 SplineAnimation Module

This section defines the functionality of the SMIL 2.0 SplineAnimation module. This module adds attributes for spline interpolation and for uneven spacing of points in time. These attributes may be used in **animate**, **animateMotion** and **animateColor** elements.

### 3.7.1 SMIL 2.0 SplineAnimation Module Attributes

**Spline animation function calculation mode**

The SplineAnimation module extends the `discrete`, `linear` and `paced` calculation modes of the BasicAnimation module, providing additional control over interpolation and timing:

- The calcMode attribute is extended to support splines.
- A path attribute is added to the **animateMotion** element, allowing authors to define a motion path using a subset of the SVG [SVG] path syntax, and providing smooth path motion.
- The keyTimes attribute provides additional control over the timing of the animation function, associating a time with each value in the values list (or the points in a path description for the **animateMotion** element).
- The keySplines attribute provides a means of controlling the pacing of interpolation *between* the values in the values list.

**Calculation mode attributes**

calcMode
: In addition to the values `discrete`, `linear` and `paced` of the BasicAnimation module, the AdvanceAnimation module supports the value

  `spline`
  : Interpolates from one value in the values list to the next according to a time function defined by a cubic Bezier spline. The points of the spline are defined in the keyTimes attribute, and the control points for each interval are defined in the keySplines attribute.

  The use of `discrete` for the calcMode together with a path specification is allowed, but will simply jump the target element from point to point. The times are derived from the points in the path specification, as described in the path attribute, immediately below.

keyTimes
: 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 should be 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 keyTime 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 there are any errors in the keyTimes specification (bad values, too many or too few values), the animation will have no effect.

If the simple duration is indefinite and the interpolation mode is **linear** or **spline**, any keyTimes specification will be ignored.

keySplines
: A set of Bezier control points associated with the keyTimes list, defining a cubic Bezier function that controls interval pacing. The attribute value is a semicolon separated list of control point descriptions. Each control point description is a set of four floating point values: `x1 y1 x2 y2`, describing the Bezier control points for one time segment. The keyTimes values that define the associated segment are the Bezier "anchor points", and the keySplines values are the control points. Thus, there must be one fewer sets of control points the keySplines attribute than there are keyTimes.

The values must all be in the range 0 to 1.

This attribute is ignored unless the calcMode is set to **spline**.

If there are any errors in the keySplines specification (bad values, too many or too few values), the animation will have no effect.

This semantic (the duration is divided into `n-1` even periods) applies as well when the keySplines attribute is specified, but keyTimes is not. The times associated to the keySplines values are determined as described above.

The syntax for the control point sets in keySplines lists is:

Using:

```
fpval          ::= Floating point number
S              ::= spacechar*
comma-wsp      ::= S (spacechar(",") S
spacechar      ::= (#x20 | #x9 | #xD | #xA)
```

Control point values are separated by at least one white space character or a comma. Additional white space around the separator is allowed. The allowed syntax for floating point numbers must be defined in the host language.

If the argument values for keyTimes or keySplines are not legal (including too few or too many values for either attribute), the animation will have no effect (see also Handling syntax errors).

In the calcMode, keyTimes and keySplines attribute values, leading and trailing white space and white space before and after semicolon separators will be ignored.

*Examples of advanced uses of calcMode*

Discrete animation can be used with keyTimes, as in the following example:

```
<animateColor attributeName="color" dur="10s" calcMode="discrete"
    values="green; yellow; red" keyTimes="0.0; 0.5;" />
```

This example also shows how keyTimes values can interact with an indefinite duration. The value of the "color" attribute will be set to green for 5 seconds, and then to yellow for 5 seconds, and then will remain red for the remainder of the document, since the (unspecified) duration defaults to "indefinite".

The following example illustrates the use of keyTimes:

```
<animate attributeName="x" dur="10s" values="0; 50; 100"
    keyTimes="0; .8; 1" calcMode="linear"/>
```

The keyTimes values cause the "x" attribute to have a value of "0" at the start of the animation, "50" after 8 seconds (at 80% into the simple duration) and "100" at the end of the animation. The value will change more slowly in the first half of the animation, and more quickly in the second half.

*Interpolation with keySplines*

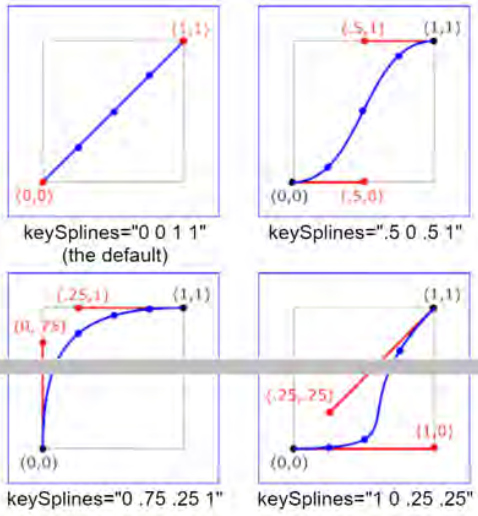
For some attributes, the *pace* of change may not be easily discernable by viewers. However for animations like motion, the ability to make the *speed* of the motion change gradually, and not in abrupt steps, can be important. The keySplines attribute provides this control.

Extending the above example to use keySplines:

```
<animate attributeName="x" dur="10s" values="0; 50; 100"
    keyTimes="0; .8; 1" calcMode="spline"
    keySplines=".5 0 .5 1; .8 0 1 1" />
```

The keyTimes still cause the "x" attribute to have a value of "0" at the start of the animation, "50" after 8 seconds and "100" at the end of the animation. However, the keySplines values define a curve for pacing the interpolation between values. In the example above, the spline causes an ease-in and ease-out effect between time 0 and 8 seconds (i.e. between keyTimes 0 and .8, and values "0" and "50"), but a strict linear interpolation between 8 seconds and the end (i.e. between keyTimes .8 and 1, and values "50" and "100"). Figure 7 shows the curves that these keySplines values define.

***Figure 7 - Illustration of keySplines effect***




keySplines="0 0 1 1" (the default)

keySplines=".5 0 .5 1"

keySplines="0 .75 .25 1"

keySplines="1 0 .25 .25"

Each diagram in Figure 7 illustrates the effect of keySplines settings for a single interval (i.e. between the associated pairs of values in the keyTimes and values lists.). The horizontal axis can be thought of as the input value for the *unit progress* of interpolation within the interval - i.e. the pace with which interpolation proceeds along the given interval. The vertical axis is the resulting value for the *unit progress*, yielded by the keySplines function. Another way of describing this is that the horizontal axis is the input *unit time* for the interval, and the vertical axis is the output *unit time*. See also the section Timing and real-world clock times.

To illustrate the calculations, consider the simple example:

```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
    calcMode="spline" keySplines="as in table" />
```

Using the keySplines values for each of the four cases above, the approximate interpolated values as the animation proceeds are:

| keySplines values | Initial value | After 1s | After 2s | After 3s | Final value |
|---|---|---|---|---|---|
| 0 0 1 1 | 10.0 | 12.5 | 15.0 | 17.5 | 20.0 |
| .5 0 .5 1 | 10.0 | 11.0 | 15.0 | 19.0 | 20.0 |
| 0 .75 .25 1 | 10.0 | 18.0 | 19.3 | 19.8 | 20.0 |
| 1 0 .25 .25 | 10.0 | 10.1 | 10.6 | 16.9 | 20.0 |

For a formal definition of Bezier spline calculation, see [COMP-GRAPHICS], pages 488-491.

The keyTimes and keySplines attributes can also be used with the *from/to/by* shorthand forms for specifying values, as in the following example:

```
<animate attributeName="foo" from="10" to="20"
    calcMode="spline" keySplines=".5 0 .5 1" />
```

The value will change from 10 to 20, using an "*ease-in/ease-out*" curve specified by the keySplines values. The keyTimes values cause the value of 20 to be reached at 7 seconds, and to hold there for the remainder of the 10 second simple duration.

The following example describes a somewhat unusual usage, a *from-to* animation with discrete animation. The `stroke-linecap` attribute of SVG elements takes a string, and so implies a calcMode of **`discrete`**. The animation will set the `stroke-linecap` attribute to **`round`** for 5 seconds (half the simple duration) and then set the `stroke-linecap` to **`square`** for 5 seconds.

```
<rect stroke-linecap="butt"...>
   <animate attributeName="stroke-linecap"
      from="round" to="square" dur="10s"/>
</rect>
```

### 3.7.2 SMIL 2.0 SplineAnimation Module Elements

The SplineAnimation module extends the BasicAnimation elements **animate**, **animateMotion** and **animateColor**, adding the attributes keyTimes and keySplines, and the value **`spline`** for the calcMode attribute.

### 3.7.3 The spline animate element

The SplineAnimation module extends the **animate** element defined by the BasicAnimation module, adding the following attributes and values.

**Element attributes**

**all attributes and associated elements of the animate element in BasicAnimation.**
See The animate element.
keyTimes,
keySplines, **and**
calcMode

Extend the specification animation function and effect. See The animate element and Spline animation function calculation mode.

Examples are provided above, as are normative definitions of the semantics of all attributes supported by animate.

### 3.7.4 The spline animateMotion element

The SplineAnimation module extends the animateMotion element defined by the BasicAnimation module, adding the following attributes and values.

**Element attributes**

**all attributes and associated elements of the animateMotion element in BasicAnimation.**
See The animateMotion element.
keyTimes,
keySplines, **and**
calcMode

Extend the specification animation function and effect. See The animateMotion element and Spline animation function calculation mode.

path
Specifies the curve that describes the attribute value as a function of time. The supported syntax is a subset of the SVG path syntax. Support includes commands to describes lines ("MmLlHhVvZz") and Bezier curves ("Cc"). For details refer to the path specification in SVG [SVG].
Note that SVG provides two forms of path commands, "absolute" and "relative". These terms may appear to be related to the definition of additive animation and/or to the from attribute, but they are orthogonal. The terms "absolute" and "relative" apply only to the definition of the path itself, and not to the operation of the animation. The "relative" commands define a path point relative to the previously specified point. The terms "absolute" and "relative" are unrelated to the definitions of both "additive" animation and any specification of origin.

- For the "absolute" commands ("MLHVZC"), the host language must specify the coordinate system of the path values.
- If the "relative" commands ("mlhvzc") are used, they simply define the point as an offset from the previous point on the path. This does not affect the definition of "additive" or origin for the animateMotion element.

A path data segment must begin with either one of the "moveto" commands.

**Move To commands - "M <x> <y>" or "m <dx> <dy>"**
Start a new sub-path at the given (x,y) coordinate. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands.
**Line To commands - "L <x> <y>" or "l <dx> <dy>"**
Draw a line from the current point to the given (x,y) coordinate which becomes the new current point. A number of coordinate pairs may be specified to draw a polyline.
**Horizontal Line To commands - "H <x>" or "h <dx>"**
Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). Multiple x values can be provided.
**Vertical Line To commands - "V <y>" or "v <dy>"**
Draws a vertical line from the current point (cpx, cpy) to (cpx, y). Multiple y values can be provided.
**Closepath commands - "Z" or "z"**
The "closepath" causes an automatic straight line to be drawn from the current point to the initial point of the current subpath.
**Cubic Bezier Curve To commands -**
**"C <x1> <y1> <x2> <y2> <x> <y>" or**
**"c <dx1> <dy1> <dx2> <dy2> <dx> <dy>"**
Draws a cubic Bezier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. Multiple sets of coordinates may be specified to draw a polybezier.

For all calcMode settings, the definition of the simple animation function, `f(t)`, uses the number of values in the values attribute to determine how the simple duration is `d` is divided into segments. When a path attribute is used, the number of values is defined to be the number of points defined by the path, unless there are "move to" commands within the path. A "move to" command does not define an additional "segment" for the purposes of timing or interpolation. A "move to" command does not count as an additional point when dividing up the duration. When a path is combined with a `paced` calcMode setting, all "move to" commands are considered to have 0 duration (i.e. they always happen instantaneously), and should not be considered in computing the pacing.

If the path attribute is is specified, any from/to/by or values attribute values will be ignored.

Examples are provided above, as are normative definitions of the semantics of all attributes supported by animate.

For complete velocity control, calcMode can be set to `spline` and the author can specify a velocity control spline with keyTimes and keySplines.

### 3.7.5 The spline animateColor element

The SplineAnimation module extends the animateColor element defined by the BasicAnimation module, adding the following attributes and values.

**Element attributes**

**all attributes and associated elements of the animateColor element in BasicAnimation.**
See The animateColor element.
keyTimes,
keySplines, **and**
calcMode

Extend the specification animation function and effect. See The animateColor element and Spline animation function calculation mode.

## 3.8 SMIL 2.0 SplineAnimation Module Details

### 3.8.1 SplineAnimation integration requirements

To specify the integration of the SMIL 2.0 SplineAnimation module into a host language, the language designer must integrate SMIL 2.0 BasicAnimation into the language, satisfying all the requirements listed in BasicAnimation integration requirements.

In addition to integrating BasicAnimation, the requirements listed in Common animation integration requirements must be satisfied for the SplineAnimation module.

### 3.8.2 Document type definition (DTD) for the SplineAnimation module

See the full DTD for the SMIL Animation Modules.

## 3.9 Common Animation Integration Requirements

This section presents host-language-integration issues which are the same for the BasicAnimation and SplineAnimation modules.

### 3.9.1 Integration requirements

The host language profile must integrate the SMIL 2.0 BasicInlineTiming module into the host language, satisfying all requirements of that module. In the host language must be available on BasicAnimation elements.

In particular, the fill attribute is supported on animation elements only if the host language integrates the SMIL 2.0 BasicTimeContainers module in addition to the BasicInlineTiming module.

The host langauge profile may add additional attributes to Animation elements. Attributes added to any Animation element must be added to all Animation elements. In particular, this module does not define an XML ID attribute. It is expected that the host language profile will add an XML ID attribute to the Animation elements.

**Extending Animation**

Language designers integrating SMIL Animation are encouraged to define new animation elements where such additions will be of convenience to authors. The new elements must be based on SMIL Animation and SMIL Timing and Synchronization, and must stay within the framework provided by SMIL Timing and Synchronization and SMIL Animation.

Language designers are also encouraged to define support for additive and cumulative animation for non-numeric data types where addition can sensibly be defined.

**Constraints on manipulating animation elements**

Language designers integrating SMIL Animation are encouraged to disallow manipulation of attributes of the animation elements after the document has begun. This includes both the attributes specifying targets and values, as well as the timing attributes. In particular, the `id` attribute (of type ID) on all animation elements must not be mutable (i.e. should be read-only). Requiring animation runtimes to track changes to `id` values introduces considerable complexity, for what is at best a questionable feature.

It is recommended that language specifications disallow manipulation of animation element attributes through DOM interfaces after the document has begun. It is also recommended that language specifications disallow the use of animation elements to target other animation elements.

Note in particular that if the attributeName attribute can be changed (either by animation or script), problems may arise if the target attribute has a namespace qualified name. Current DOM specifications do not include a mechanism to handle this binding.

Dynamically changing the attribute values of animation elements introduces semantic complications to the model that are not yet sufficiently resolved. This constraint may be lifted in a future version of SMIL Animation.

**Handling syntax errors**

The specific error handling mechanisms for each attribute are described with the individual syntax descriptions. Some of these specifications describe the behavior of an animation with syntax errors as "having no effect". This means that the animation will continue to behave normally with respect to timing, but will not manipulate any presentation value, and so will have no visible impact upon the presentation.

In particular, this means that if other animation elements are defined to begin or end relative to an animation that "has no effect", the other animation elements will begin and end as though there were no syntax errors. The presentation runtime may indicate an error, but need not halt presentation or animation of the document.

Some host languages and/or runtimes may choose to impose stricter error handling (see also Error handling semantics for a discussion of host language issues with error handling). Authoring environments may also choose to be more intrusive when errors are detected.

**Error handling semantics**

The host language designer may impose stricter constraints upon the error handling semantics. That is, in the case of syntax errors, the host language may specify additional or stricter mechanisms to be used to indicate an error. An example would be to stop all processing of the document, or to halt all animation.

Host language designers may not relax the error handling specifications, or the error handling response (as described in Handling syntax errors). For example, host language designers may not define error recovery semantics for missing or erroneous values in the values or keyTimes attribute values.

# 4. The SMIL 2.0 Content Control Modules

***Editors***
Dick Bulterman, (Dick.Bulterman@oratrix.com), Oratrix
Jeffrey Ayars (jeffa@real.com), RealNetworks

## 4.1 Introduction

This section defines the SMIL 2.0 content control modules. These modules contain elements and attributes which provide for runtime content choices and optimized content delivery. SMIL content control functionality is partitioned across four modules:

- BasicContentControl, containing content selection elements and predefined system test attributes;
- CustomTestAttributes, containing author-defined custom test elements and attributes;
- PrefetchControl, containing presentation optimization elements and attributes; and
- SkipContentControl, containing attributes that support selective attribute evaluation.

Since all of the content control elements and attributes are defined in modules, designers of other markup languages can reuse this functionality on a module by module basis when they need to include media content control in their language.

The functionality in the CustomTestAttributes module builds on the functionality of the BasicContentControl module; profiles implementing the CustomTestAttributes module must also implement the BasicContentControl module. The PrefetchControl and SkipContentControl modules have no prerequisites.

In some of the module descriptions for content control, the concept of "user preference" may be present. User preferences are usually set by the playback engine using a preferences dialog box, but this specification does not place any restrictions on how such preferences are communicated from the user to the SMIL player.

It is implementation dependent when content control attributes are evaluated. Attributes may be evaluated multiple times. Dynamic reevaluation is allowed but not required.

## 4.2 The SMIL 2.0 BasicContentControl Module

### 4.2.1 SMIL 2.0 BasicContentControl Module Overview

SMIL 1.0 provides a "test-attribute" mechanism to process an element only when certain conditions are true, for example when the language preference specified by the user matches that of a media object. One or more test attributes may appear on media object references or timing structure elements; if the attribute evaluates to `true`, the containing element is played, and if the attribute evaluates to `false` the containing element is ignored. SMIL 1.0 also provides the switch element for expressing that a set of document parts are alternatives, and that the first one fulfilling certain conditions should be chosen. This is useful to express that different language versions of an audio file are available, and that the client may select one of them.

The SMIL 2.0 BasicContent module includes the test attribute functionality from SMIL 1.0 and extends it by supporting new system test attributes. This section will describe the use of the predefined system test attributes, the switch element and test attribute in-line placement. A mechanism for extending test attributes is presented in the CustomTestAttributes module.

**Predefined System Test Attributes**

This specification defines a list of test attributes that can be added to language elements, as allowed by the language designer. In SMIL 1.0, these elements are synchronization and media elements. Conceptually, these attributes represent Boolean tests. When any of the test attributes specified for an element evaluates to `false`, the element carrying this attribute is ignored.

SMIL 2.0 supports the full set of SMIL 1.0 system attributes. The SMIL 1.0 compatible system test attributes are:

- systemBitrate
- systemCaptions
- systemLanguage
- system-overdub-or-caption (note: this attribute has been deprecated in favor of systemCaptions or systemOverdubOrSubtitle)
- systemRequired
- systemScreenDepth
- systemScreenSize

Note that, with the exception of system-overdub-or-caption, the names of these attributes have been changed to reflect SMIL 2.0's *camelCase* conventions. The SMIL 1.0 hyphenated names are deprecated in this release.

New to SMIL 2.0 are system test attributes that define additional characteristics of the system environment. These are:

- systemAudioDesc
- systemCPU
- systemComponent
- systemOperatingSystem
- systemOverdubOrSubtitle

The complete definition of each attribute is given in the attributes definition section.

**The switch element**

The switch element allows an author to specify a set of alternative elements from which only the first acceptable element is chosen.

An example of the use of the switch is:

```
...
<par>
  <video src="anchor.mpg" ... />
  <switch>
    <audio src="dutchHQ.aiff" systemBitrate="56000" ... />
    <audio src="dutchMQ.aiff" systemBitrate="28800" ... />
    <audio src="dutchLQ.aiff" ... />
  </switch>
</par>
...
```

In this example, one audio object is selected to accompany the video object. If the system bitrate is 56000 or higher, the object *dutchHQ.aiff* is selected. If the system bitrate is at least 28800 but less than 56000, the object *dutchMQ.aiff* is selected. If no other objects are selected, the alternative *dutchLQ.aiff* is selected, since it has no test attribute (thus is always acceptable) and no other test attributes evaluated to `true`.

Authors should order the alternatives from the most desirable to the least desirable. Furthermore, authors may wish to place a relatively fail-safe alternative as the last item in the switch so that at least one item within the switch is chosen (unless this is explicitly not desired).

Note that some network protocols, e.g. HTTP and RTSP, support content-negotiation, which may be an alternative to using the switch element in some cases.

It is the responsibility of the SMIL 2.0 player to determine the setting for system test attribute values. Such settings may be determined statically based on configuration settings, or they may be determined (and re-evaluated) dynamically, depending on the player implementation. Players may not select members of a switch at random.

**System Test Attribute In-Line Use**

To allow more flexibility in element selection, test attributes may also be used outside of the switch element.

In the following example of in-line test attribute use, captions are shown only if the user wants captions `on`.

```
...
<par>
   <audio src="audio.rm"/>
   <video src="video.rm"/>
   <textstream src="stockticker.rt"/>
   <textstream src="closed-caps.rt" systemCaptions="on"/>
</par>
...
```

The alternatives indicated by the in-line construct could be represented as a set of switch statements, although the resulting switch could become explosive in size. Use of an in-line test mechanism significantly simplifies the specification of adaptive content, especially in those cases where many independent alternatives exist. Note, however, that there is no fail-safe alternative mechanism (such as defining an element without a test attribute inside of a switch) when using test attributes in-line.

**Examples of Switch and Test Attribute Use**

1. Choosing between content with different total bitrates

   In a common scenario, implementations may wish to allow for selection via a systemBitrate attribute on elements. The SMIL 2.0 player evaluates each of the elements within the switch one at a time, looking for an acceptable bitrate value.

   ```
   ...
   <par>
      <text .../>
      <switch>
         <par systemBitrate="40000">
         ...
         </par>
         <par systemBitrate="24000">
         ...
         </par>
         <par systemBitrate="10000">
         ...
         </par>
      </switch>
   </par>
   ...
   ```

   In this example, if the system bitrate has been determined to be less than 10000 (in mobile telephone cases, for example), then none of the par constructs would be included.

2. Choosing between audio resources with different bitrates

   The elements within the switch may be any combination of elements. For instance, one could specify an alternate audio track:

   ```
   ...
   <switch>
      <audio src="joe-audio-better-quality" systemBitrate="16000" />
      <audio src="joe-audio" />
   </switch>
   ...
   ```

   If the system bitrate was less than 16000, the standard-quality audio would be presented by default.

3. Choosing between audio resources in different languages

   In the following example, an audio resource is available both in Dutch and in English. Based on the user's preferred language, the player can choose one of these audio resources.

   ```
   ...
   <switch>
      <audio src="joe-audio-nederlands" systemLanguage="nl"/>
      <audio src="joe-audio-english" systemLanguage="en"/>
   </switch>
   ...
   ```

   In this example, if the system language setting was anything other than Dutch or English, no audio would be presented. To make a choice the default, it should appear as the last item in the list and not contain a test attribute. In the following fragment, English is used as the default:

   ```
   ...
   <switch>
      <audio src="joe-audio-nederlands" systemLanguage="nl"/>
      <audio src="joe-audio-english" />
   </switch>
   ...
   ```

4. Choosing between content written for different screens

   In the following example, the presentation contains alternative parts designed for screens with different resolutions and bit-depths. Depending on the particular characteristics of the screen, the player must use the first alternative in which all of the test attributes evaluate to `true`.

   ```
   ...
      <text .../>
      <switch>
         <par systemScreenSize="1024X1280" systemScreenDepth="16">
         ...
         </par>
         <par systemScreenSize="480X640" systemScreenDepth="32">
         ...
         </par>
         <par systemScreenSize="480X640" systemScreenDepth="16">
   ```

```
      ...
      </par>
    </switch>
  </par>
  ...
```

5. Supporting multiple options via in-line use

   This example shows a video that is accompanied by zero or more media objects. If the system language has been set to either Dutch or English, then the appropriate audio object will play. In addition, if the system language has been set to either Dutch or English and systemCaptions has also been set to `on`, the appropriate text files will also be displayed.

```
...
<par>
  <video src="anchor.mpg" ... />
  <audio src="dutch.aiff"   systemLanguage="nl"  ... />
  <audio src="english.aiff" systemLanguage="en" ... />
  <text  src="dutch.html"   systemLanguage="nl" systemCaption="on"... />
  <text  src="english.html" systemLanguage="en" systemCaption="on"... />
</par>
...
```

   If system language is set to something other than Dutch or English, no objects will be rendered (except the video). Note that there is no catch-all default mechanism when using test attributes for in-line evaluation.

6. Choosing the language of overdub and subtitle tracks

   In the following example, a French-language movie is available with English, German, and Dutch overdub and subtitle tracks. The following SMIL segment expresses this, and switches on the alternatives that the user prefers.

```
...
<par>
  <switch>
    <audio src="movie-aud-en.rm" systemLanguage="en"
           systemOverdubOrSubtitle="overdub"/>
    <audio src="movie-aud-de.rm" systemLanguage="de"
           systemOverdubOrSubtitle="overdub"/>
    <audio src="movie-aud-nl.rm" systemLanguage="nl"
           systemOverdubOrSubtitle="overdub"/>
    <!-- French for everyone else -->
    <audio src="movie-aud-fr.rm"/>
  </switch>
  <video src="movie-vid.rm"/>
  <switch>
    <textstream src="movie-sub-en.rt" systemLanguage="en"
           systemOverdubOrSubtitle="subtitle"/>
    <textstream src="movie-sub-de.rt" systemLanguage="de"
           systemOverdubOrSubtitle="subtitle"/>
    <textstream src="movie-sub-nl.rt" systemLanguage="nl"
           systemOverdubOrSubtitle="subtitle"/>
    <!-- French captions for those that really want them -->
    <textstream src="movie-caps-fr.rt" systemCaptions="on"/>
  </switch>
</par>
...
```

### 4.2.2 Elements and Attributes

SMIL 2.0 BasicContentControl defines the switch element and a set of predefined system test attributes.

**The switch element**

The switch element allows an author to specify a set of alternative elements. An element is selected as follows: the player evaluates the elements in the order in which they occur in the switch element. The first acceptable element is selected at the exclusion of all other elements within the switch. Implementations must NOT arbitrarily pick an object within a switch when test attributes for all child elements fail.

*Element attributes*

This element does not have attributes beyond those required of all elements in the profile.

*Element content*

The content of the element is language implementation dependent.

In the SMIL 2.0 language profile, if the switch is used as a direct or indirect child of a body element, it may contain any media object or timing structure container, or it may contain nested switch elements. All of these elements may appear multiple times inside the switch. If the switch is used as a direct or indirect child of a head element, it may contain one or more layout elements.

**Predefined Test Attributes**

SMIL 2.0 defines the following system test attributes. When any of the test attributes specified for an element evaluates to `false`, the element carrying this attribute is ignored. Note that most hyphenated test attribute names from SMIL 1.0 have been deprecated in favor of names using the current SMIL *camelCase* convention. For these, the deprecated SMIL 1.0 name is shown in parentheses after the preferred name.

systemAudioDesc
: values: `on` | `off`
  This test attribute specifies whether or not closed audio descriptions should be rendered. This is intended to provide authors with the ability to support audio descriptions in the same way that systemCaptions provides text captions. The value of systemAudioDesc is used to control object rendering in conjunction with the user's preference for receiving audio descriptions of a media object if and when these are available. A value of `on` indicates a preference to have such descriptions rendered when available. A value of `off` indicates a preference not to render such descriptions.
  Authors should place systemAudioDesc = `on` only on elements that they wish to render when the user has indicated they want audio descriptions. Authors should place systemAudioDesc = `off` only on elements that they wish to render when the user has indicated they DON'T want audio descriptions.
  Evaluates to `true` if the user preference matches this attribute value. Evaluates to `false` if they do not match.

systemBitrate (system-bitrate)

value: `the approximate bandwidth, in bits-per-second, available to the system.`
The measurement of bandwidth is application specific, meaning that applications may use sophisticated measurement of end-to-end connectivity, or a simple static setting controlled by the user. In the latter case, this could for instance be used to make a choice based on the user's connection to the network. Typical values for modem users would be 14400, 28800, 56000 bit/s etc. Evaluates to `true` if the available system bitrate is equal to or greater than the given value. Evaluates to `false` if the available system bitrate is less than the given value.
The attribute can assume any integer value greater than 0. If the value exceeds an implementation-defined maximum bandwidth value, the attribute always evaluates to `false`.

systemCaptions (system-captions)

values: `on | off`

This attribute allows authors to to specify a redundant text equivalent of the audio portion of the presentation. Examples of intended use are: audiences with hearing disabilities, those learning to read, or anyone who wants or needs this information.
Evaluates to `true` if the user preference matches this attribute value. Evaluates to `false` if they do not match.

systemComponent

value: `an XML CDATA string` containing one or more white-space separated `URI`'s.
Each URI identifies a component of the playback system, e.g. user agent component/feature, number of audio channels, codec, HW MPEG decoder, etc. The URI is implementation dependent. Each implementation is encouraged to publish a list of component URIs which can be used to identify and resolve the presence of implementation-dependent components.
Evaluates to `true` if the resulting value is `true`. Evaluates to `false` otherwise.

systemCPU

value: `an XML NMTOKEN` [XML10].
This test attribute specifies the CPU on which a user agent may be running. An implementation must allow the user the ability to set the system value to `unknown` for privacy.
The following list contains the suggested values for this test attribute (additional names may be supported by an implementation): `alpha, arm, arm32, hppa1.1, m68k, mips, ppc, rs6000, vax, x86, unknown.`

*These values come from the _PR_SI_ARCHITECTURE constants defined by the mozilla project.*

Evaluates to `true` if the user preference matches this attribute value. Evaluates to `false` if they do not match. The value is case-sensitive.

systemLanguage (system-language)

values: `a comma-separated list of language names as defined in [RFC1766], or an empty/null string`
This attribute evaluates to `true` (1) if one of the languages indicated by user preferences exactly equals one of the languages given in the value of this parameter, or (2) if one of the languages indicated by user preferences exactly matches a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-". Evaluates to `false` otherwise. For example, if generic English is specified in the user preferences, it will match both "en" by case (1) and "en-gb" (British English) by case (2), but if British English is specified in the user preferences, it will match only "en-gb" by case (1).
If a null or empty string is specified, the test attribute evaluates to `false`.

The syntax of the systemLanguage and the deprecated system-language attributes are defined using EBNF notation (as defined in [XML10]) as list of XML namespace prefixes [XML-NS], separated by the ',' character:

```
systemLanguageArgumentValue ::= (languageTag (S? ',' S? languageTag)*)?
```

Where allowed white space is indicated as "S", defined as follows (taken from the [XML10] definition for 'S'):

```
S ::= (#x20 | #x9 | #xD | #xA)+
```

*Implementation: When making the choice of linguistic preference available to the user, implementers should take into account the fact that most users are not familiar with the details of language matching as described above, and should provide appropriate guidance. As an example, users may mistakenly 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.*

systemOperatingSystem

value: `an XML NMTOKEN` [XML10].
This test attribute specifies the operating system on which a user agent may be running. An implementation must allow the user the ability to set the user preference to `unknown` for privacy.
The following list contains the suggested values for this test attribute (additional names may be supported by an implementation): `aix, beos, bsdi, dgux, freebsd, hpux, irix, linux, macos, ncr, nec, netbsd, nextstep, nto, openbsd, openvms, os2, osf, palmos, qnx, sinix, rhapsody, sco, solaris, sonly, sunos, unixware, win16, win32, win9x, winnt, wince, unknown.`

*These values come from the _PR_SI_SYSNAME constants defined by the mozilla project.*

Evaluates to `true` if the user preference matches this attribute value. Evaluates to `false` if they do not match. The value is case-sensitive.

systemOverdubOrSubtitle

values: `overdub | subtitle`
This attribute specifies whether subtitles or overdub is rendered. `overdub` selects for substitution of one voice track for another, and `subtitle` means that the user prefers the display of text in a language other than that which is being used in the audio track.
Evaluates to `true` if the user preference matches this attribute value. Evaluates to `false` if they do not match.

system-overdub-or-caption

values: `caption | overdub`
This test attribute has been deprecated in favor of using systemOverdubOrSubtitle and systemCaptions.
This attribute is a setting which determines if users prefer overdubbing or captioning when the option is available.
Evaluates to `true` if the user preference matches this attribute value. Evaluates to `false` if they do not match.

systemRequired (system-required)

value: `list of namespace prefix language extensions`
This attributes provides an extension mechanism for new elements or attributes. Evaluates to `true` if all of the extensions in the list are supported by the implementation, otherwise, this evaluates to `false`. The syntax of the systemRequired and the deprecated system-required attributes are defined using EBNF notation (as defined in [XML10]) as list of XML namespace prefixes [XML-NS], separated by the '+' character:

```
systemRequiredArgumentValue := NMTOKEN (S? '+' S? NMTOKEN)*
```

Where allowed white space is indicated as "S", defined as follows (taken from the [XML10] definition for 'S'):

```
S ::= (#x20 | #x9 | #xD | #xA)+
```

systemScreenDepth (system-screen-depth)

values: `a number greater than 0`
This attribute specifies the depth of the screen color palette in bits required for displaying the element. Typical values are `1 | 4 | 8 | 24 | 32`.
Evaluates to `true` if the playback engine is capable of displaying images or video with the given color depth. Evaluates to `false` if the playback

engine is only capable of displaying images or video with a smaller color depth.

systemScreenSize (system-screen-size)

value: `a screen size`

Attribute values have the following syntax:

```
screen size ::= screen-height"X"screen-width
```

Each of these is a pixel value, and must be an integer value greater than 0.

Evaluates to `true` if the playback engine is capable of displaying a presentation of the given size. Evaluates to `false` if the playback engine is only capable of displaying smaller presentations.

It is the responsibility of the SMIL 2.0 Player to determine the settings for each predefined test variable. These values may be determined by static configuration settings, or they may be evaluated dynamically during runtime. Such setting and (re)evaluation behavior is implementation dependent.

For this version of SMIL elements with specified test attributes that evaluate to false, or elements within a switch that are not selected, are considered to be ignored and will behave as though they were not specified in the document. Any references to these elements will be as if the elements were not in the document. In particular, any ID references to the element will act as if there was no element with that ID. Languages that integrate this module must specify any additional behavior related to these ignored elements. In the SMIL 2.0 Language profile, timing attributes that reference invalid IDs are treated as being indefinite.

*Authors should be aware that this model for handling ignored elements may be revised in a future version of SMIL, and the related semantics may well change. These changes should not affect implementations that only support parse-time (or equivalent) evaluation of test attributes and/or the switch element. However, the semantics of dynamic re-evaluation (i.e. re-evaluation during document presentation) of test attributes and/or switch elements are not defined in this version of SMIL; this will be addressed in a future version.*

*Authors should realize that if several alternative elements are enclosed in a **switch**, and none of them evaluate to true, this may lead to situations such as a media object being shown without one or more companion objects. It is thus recommended to include a "catch all" choice at the end of a **switch** which is acceptable in all cases.*

### 4.2.3 Integration Requirements for the BasicContentControl Module

The functionality in this module does not build on functionality defined in other SMIL 2.0 modules.

### 4.2.4 Document Type Definition (DTD) for the BasicContentControl Module

See the full DTD for the SMIL Content Control modules.

## 4.3 The SMIL 2.0 CustomTestAttributes Module

### 4.3.1 SMIL 2.0 CustomTestAttributes Module Overview

The use of predefined system test attributes in the SMIL BasicContentControl module provides a selection mechanism based on attributes that are fixed within the module's definition. The CustomTestAttribute module extends this facility with the definition of author-defined custom test attributes. Custom test attributes allow presentation authors to define their own test attributes for use in a specific document. Custom test attributes may be shared among application documents using the uid attribute.

As with system test attributes, custom test attributes can be used within timing structure and media object elements; if they evaluate to `true`, the containing element is activated and if they evaluate to `false`, the containing element is ignored. In this version of SMIL, an ignored element will be treated as if it were not part of the source document. As a result, any element referencing the ID of the ignored node will, in effect, reference an invalid ID. Languages that integrate this module must specify any additional behavior related to these ignored elements.

Since custom test attributes are application/document specific, they need a mechanism to allow attribute definition and attribute setting. Attribute definition is done via the customAttributes and customTest elements. The initial state of any custom test attribute can be set at author-time with the defaultState attribute, which takes a value of either `true` or `false`. This module provides an override attribute with a value `hidden` that gives an author the ability to discourage runtime resetting of any attributes using these mechanisms.

The state of the attribute can be changed in one of three ways:

1. by modifying the value of the default state attribute in the document source before delivery to the player;
2. by using the unique identifier given in the uid attribute as a reference to a runtime value for the custom test;
3. by an interface presented to the user (or the user agent) through the document player at runtime.

The exact rules for setting and modifying the values associated with custom test attributes are given below.

An implementation may support either, both, or none of methods 2 and 3. If method 2 is supported, the URI value in uid is simply a unique identifier and does not imply that the runtime value must be fetched over the Web. The value may be stored and retrieved locally, and simply identified by the uid. The precise manner in which this is done is implementation dependent. If method 3 is supported, the custom test attribute facility does not require any specific UI support for direct user manipulation of the custom test attributes.

**Example Use**

The following example shows one way in which custom test attributes can be applied within a SMIL 2.0 Language profile document:

```
<smil>
  <head>
    <layout>
      <!-- define projection regions -->
    </layout>
    <customAttributes>
      <customTest id="west-coast" title="West Coast Edition"
        defaultState="false" override="visible"
        uid="http://defs.example.org/user-settings/west-coast" />
      <customTest id="east-coast" title="East Coast Edition"
        defaultState="false" override="visible"
        uid="http://defs.example.org/user-settings/east-coast" />
      <customTest id="far-north"  title="Northern Edition"
        defaultState="false" override="visible"
        uid="http://defs.example.org/user-settings/far-north" />
      <customTest id="the-rest"   title="National Edition"
        defaultState="true"  override="hidden" />
    </customAttributes>
  </head>
  <body>
```

```
    ...
    <par>
      <img src="background.png" region="a"/>
      <video src="story_lv.rm" region="b" />
      <switch>
        <audio src="story_lw.rm" region="c" customTest="west-coast"/>
        <audio src="story_le.rm" region="c" customTest="east-coast"/>
        <audio src="story_ln.rm" region="c" customTest="far-north"/>
        <audio src="story_lr.rm" region="c" customTest="the-rest"/>
      </switch>
      ...
    </par>
    ...
  </body>
</smil>
```

The customAttributes element in the header contains the definition of the available custom test attributes. Each custom test attribute, defined by the customTest element, contains an identifier and a title (which can be used by a user agent, if available, to label the attribute), as well as an (optional) initial state definition, a UID that contains a unique identifier for the value setting for this attribute and an override flag.

The custom test variables named "west-coast", "east-coast" and "far-north" are defined with a default rendering state of `false`. They each contain a reference to a URI which is used to define local settings for the respective variables.

The custom test variable "the-rest" is defined with a default rendering setting of `true`.

Inside the body, a SMIL switch construct is used to select media objects for inclusion in a presentation depending on the values of the various custom test attributes. The first object that contains a value of `true` will be rendered, and since in this example the last option will always resolve `true`, it will be rendered if no other objects resolve to `true`.

While this example shows switch-based use of custom test attributes, the facility could also be applied as test attributes in in-line use.

**Rules for Setting Values**

The setting of the value associated with a custom test attribute proceeds as follows:

1. The initial setting is taken from the value of the defaultState attribute, if present. If no default state is explicitly defined, a value of `false` is used.
2. Next, if a URI mechanism is supported by the implementation, the URI defined by the uid attribute is checked to see if a persistent value has been defined for the custom test attribute with the associated id. If such a value is present, it is used instead of the default state defined in the document (if any). Otherwise, the existing initial state is maintained.
3. Next, if a UI-based mechanism (either via the SMIL DOM, a player GUI or some other means) is available and a value has been set by the user, the value associated with the custom test attribute is set to the user-specified value. If no user preference has been defined, either the UID-based value or the default value from the document text (in that order) is used.

Note that a user setting of the custom test attribute will take precedence over a URI setting. If the user has not specified a value for the attribute then the URI setting takes precedence. As with predefined system test attributes, this evaluation will occur in an implementation-defined manner. The value may be (re)evaluated dynamically, but this is not required. Note also that not all implementations need support uid or UI setting of attributes.

### 4.3.2 Elements and Attributes

This section defines the elements and attributes that make up the functionality in the SMIL CustomTestAttributes module. The customAttributes and customTest elements are used to define custom test attribute variables and the customTest attribute is used in-line on media object and timing structure references to control evaluation of the containing elements.

**The customAttributes element**

The customAttributes element contains definitions of each of the custom test attributes. The contained elements define a collection of author-specified test attributes that can be used in switch statements or as in-line test attributes in the document.

*Element attributes*

This element does not have attributes beyond those required of all elements in the profile.

*Element content*

The customAttributes element may contain one or more customTest elements.

**The customTest element**

The customTest element defines an author-specified name that will be used as the test argument in the switch element or in-line on media object and timing structure elements. The customTest elements are defined within the section delineated by the customAttributes elements that make up part of the document header.

*Element attributes*

defaultState

The initial state for the named custom test variable is given in the value of this attribute. If unspecified, it defaults to `false`.
The run-time state for the named custom test variable may be set according to the rules for uid and/or override attribute processing, if present.
The values are not case-sensitive.

override

values: `visible` | `hidden`
This attribute allows the author to choose whether the ability to override the initial state of a custom test variable should be presented to the typical user, or whether that choice should be reserved for users that specifically express a preference for this access. If the value of the override attribute is `visible`, then the user agent should make available to the user a means to set the custom test variable value in its default configuration either directly, via the SMIL DOM, or by some other mechanism. If the value of the override attribute is `hidden`, then the user agent should not present to the user a means to set the custom attribute value unless the user has expressed a preference for this access. The values are not case-sensitive. The default value is `hidden`.

uid

values: A `URI`

The URI identifies the associated custom test for persistent use. The user agent should use this as the key to store and retrieve values associated with the custom test attribute, and take care that privacy and security issues are regarded. If the permitted by the override attribute, a resolved reference to a setting via the uid attribute defines the initial setting of the custom test value; this value may be overridden by the user/user-agent if permitted by the override attribute. It is up to the runtime environment to enforce this attribute.

*The actual evaluation mechanism associated with the URI is implementation dependent. It can vary from a simple lookup in a local file or registry, to a secure reference via a capabilities database, and may be influenced by other configuration settings provided by the implementation.*

*Element content*

None.

**The customTest attribute**

In addition to the **customAttributes** and **customTest** elements, this module provides a customTest attribute that can be applied by language designers to media objects and timing structure elements requiring selection. In all operational aspects, the custom test attribute is similar to the predefined system test attribute facility of the Basic Content Control module.

customTest

value: `a list of XML identifiers`

The identifiers, defined in the **customTest** elements, define variables that are evaluated as test attributes. If the variables all evaluate to `true`, the associated element is evaluated, otherwise it and its content are skipped. customTest attributes whose values don't match the identifier of a **customTest** element evaluate to `false`.

The syntax of the customTest is defined using EBNF notation (as defined in [XML10]) as list of **customTest** element identifier references, separated by the '+' character:

```
CustomTestArgumentValue := IDREF (S? '+' S? IDREF)*
```

Where allowed white space is indicated as "S", defined as follows (taken from the [XML10] definition for 'S'):

```
S ::= (#x20 | #x9 | #xD | #xA)+
```

### 4.3.3 Integration Requirements for the CustomTestAttribute Module

The functionality in this module builds on functionality defined in the BasicContentControl module, which is a required prerequisite for inclusion of the CustomTestAttribute module.

The profile implementing the custom test elements and attributes must provide a means of associating a unique XML identifier with a customTest element, so that it can be used by the customTest attribute. And the profile should provide a means of associating descriptive text with a customTest element, which may be used in a GUI or other selection mechanism that may be presented to the user. For the SMIL 2.0 Language Profile, the element's id and title attributes serve this purpose.

### 4.3.4 Document Type Definition (DTD) for the CustomTestAttribute Module

See the full DTD for the SMIL Content Control modules.

## 4.4 The SMIL 2.0 PrefetchControl Module

### 4.4.1 SMIL 2.0 PrefetchControl Module Overview

This module defines an element and attributes that can be used to control the fetching of content from a server in a manner that will improve the rendering performance of the document.

This element will give a suggestion or hint to a user agent that a media resource will be used in the future and the author would like part or all of the resource fetched ahead of time to make the document playback smoother. User-agents can ignore **prefetch** elements, though doing so may cause an interruption in the document playback when the resource is needed. It gives authoring tools or savvy authors the ability to schedule retrieval of resources when they think that there is available bandwidth or time to do it. A **prefetch** element is contained within the body of an XML document, and its scheduling is based on its lexical order unless explicit timing is present.

*Prefetching data from a URL that changes the content dynamically is potentially dangerous: if the entire resource isn't prefetched, a subsequent request for the remaining data may yield data from a newer resource. A user agent should respect any appropriate caching directives applied to the content, e.g. no-cache 822 headers in HTTP. More specifically, content marked as non-cacheable would have to be refetched each time it was played, where content that is cacheable could be prefetched once, with the results of the prefetch cached for future use.*

**Examples**

1. Prefetch an image so it can be displayed immediately after a video ends:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <seq>
      <par>
        <prefetch id="endimage"
          src="http://www.example.org/logo.gif"/>
        <text id="interlude"
          src="http://www.example.org/pleasewait.html" fill="freeze"/>
      </par>
      <video id="main-event" src="rtsp://www.example.org/video.mpg"/>
      <img src="http://www.example.org/logo.gif" dur="5s"/>
    </seq>
  </body>
</smil>
```

The example starts with a prefetch in parallel with the rendering of a text object. The text is discrete media so it ends immediately, the prefetch is defaulted to prefetch the entire image at full available bandwidth and the prefetch element ends when the image is downloaded. That ends the

<par> and the video begins playing. When the video ends the image is shown.

2. Prefetch the images for a button so that rollover occurs quickly for the end user:

```
<body>
   <prefetch id="upimage" src="http://www.example.org/up.gif"/>
   <prefetch id="downimage" src="http://www.example.org/down.gif"/>
   ....
   <!-- script will change the graphic on rollover -->
   <img src="http://www.example.org/up.gif"/>
 </body>
</html>
```

### 4.4.2 Elements and Attributes

**The prefetch element**

The prefetch gives authors a mechanism to influence the scheduling of media object transfers from a server to the player.

Documents must still playback even when the prefetch elements are ignored, although rebuffering or pauses in presentation of the document may occur. If the prefetch for a prefetch element is ignored, any timing on the element is still respected, e.g. if a prefetch element has a `dur="5s"`, elements that depend on the prefetch element's timing behave as if the prefetch took 5 seconds.

The intrinsic duration of a prefetch element is either the duration of the media fetch, if the prefetch operation is supported by the implementation, or zero if prefetch is not supported.

If a prefetch element is repeated, due to restart or repeat on a parent element the prefetch operation should occur again. This insures appropriately "fresh" data is displayed if, for example, the prefetch is for a banner ad to a URL whose content changes with each request.

*Element attributes*

The prefetch element supports the following attributes:

mediaSize
: values: **bytes-value | percent-value**
Defines how much of the resource to fetch as a function of the file size of the resource. To fetch the entire resource without knowing its size, specify 100%. The default is 100%.

mediaTime
: values: **clock-value | percent-value**
Defines how much of the resource to fetch as a function of the duration of the resource. To fetch the entire resource without knowing its duration, specify 100%. The default is 100%.
For discrete media (non-time based media like text/html or image/png) using this attribute causes the entire resource to be fetched.

bandwidth
: values: **bitrate-value | percent-value**
Defines how much network bandwidth the user agent should use when doing the prefetch. To use all that is available, specify 100%. The default is 100%.

Any attribute with a value of "0%" is ignored and treated as if the attribute wasn't specified.

If both mediaSize and mediaTime are specified, mediaSize is used and mediaTime is ignored.

If the clipBegin or clipEnd in the media object are different from the prefetch, an implementation can use any data that was fetched but the result may not be optimal.

*Attribute value syntax*

**bytes-value**
The bytes-value value has the following syntax:

```
bytes-value ::= Digit+; any positive number
```

**percent-value**
The percent-val value has the following syntax:

```
percent-value ::= Digit+ "%"; any positive number in the range 0 to 100
```

**clock-value**
The clock-value value has the following syntax:

```
Clock-val          ::= ( Hms-val | Smpte-val )
Smpte-val          ::= ( Smpte-type )? Hours ":" Minutes ":" Seconds
                       ( ":" Frames ( "." Subframes )? )?
Smpte-type         ::= "smpte" | "smpte-30-drop" | "smpte-25"
Hms-val            ::= ( "npt=" )? (Full-clock-val | Partial-clock-val
                       ( "=" Timecount ) ? )
Full-clock-val     ::= Hours ":" Minutes ":" Seconds ( "." Fraction)?
Partial-clock-val  ::= Minutes ":" Seconds ("." Fraction)?
Timecount-val      ::= Timecount ("." Fraction)? (Metric)?
Metric             ::= "h" | "min" | "s" | "ms"
Hours              ::= DIGIT+; any positive number
Minutes            ::= 2DIGIT; range from 00 to 59
Seconds            ::= 2DIGIT; range from 00 to 59
Frames             ::= 2DIGIT; smpte range = 00-29, smpte-30-drop range = 00-29, smpte-25 range = 00-24
Subframes          ::= 2DIGIT; smpte range = 00-01, smpte-30-drop range = 00-01, smpte-25 range = 00-01
Fraction           ::= DIGIT+
Timecount          ::= DIGIT+
2DIGIT             ::= DIGIT DIGIT
DIGIT              ::= [0-9]
```

For Timecount values, the default metric suffix is "s" (for seconds).

**bitrate-value**
The bitrate-value value specifies a number of bits per second. It has the following syntax:

```
bitrate-value ::= Digit+; any positive number
```

### 4.4.3 Integration Requirements for the PrefetchControl Module

A profile integrating the PrefetchControl module must add the attributes necessary to specify the media to be fetched. In general, these will be the same resource specifying attributes as those on the media elements themselves. In addition, the profile must add any necessary attributes to control the timing of the prefetch element.

### 4.4.4 Document Type Definition (DTD) for the PrefetchControl Module

See the full DTD for the SMIL Content Control modules.

## 4.5 The SMIL 2.0 SkipContentControl Module

### 4.5.1 SMIL 2.0 SkipContentControl Module Overview

This module contains one attribute, skip-content attribute, that can be used to selectively control the evaluation of the element on which this attribute

### 4.5.2 Elements and Attributes

**Element definition**

The SkipContentControl module does not contain any element definitions.

**The skip-content attribute**

skip-content
value: `true` | `false`
This attribute controls whether the content of an element is evaluated or should be skipped.

- If a new element is introduced in a future version of language allowing markup from a previous version of the language as element content, the skip-content attribute controls whether this content is processed by the user agent.
- If an empty element in a version of a language becomes non-empty in a future SMIL version, the skip-content attribute controls whether this content is ignored by a user agent, or results in a syntax error.

If the value of the skip-content attribute is `true`, and one of the cases above apply, the content of the element is ignored. If the value is `false`, the content of the element is processed.
The default value for skip-content is `true`.

### 4.5.3 Integration Requirements for the SkipContentControl Module

It is the responsibility of the language profile to specify which elements have skip-content attributes to enable this expansion mechanism.

# 5. The SMIL 2.0 Layout Modules

***Editors***

Dick Bulterman (Dick.Bulterman@oratrix.com), Oratrix
Erik Hodge (ehodge@real.com), RealNetworks

## 5.1 Introduction

***This section is informative.***

This section defines the SMIL 2.0 Layout Modules, which are composed of a BasicLayout module and three modules with additional functionality that build on top of the BasicLayout module: the AudioLayout, MultiWindowLayout, and HierarchicalLayout modules. The modules contain elements and attributes allowing for positioning of media elements on the visual rendering surface, and control of audio volume. Since these elements and attributes are defined in modules, designers of other markup languages can choose whether or not to include this functionality in their languages. Therefore, language designers incorporating other SMIL modules do not need to include the layout modules if sufficient layout functionality is already present.

## 5.2 Overview of the SMIL 2.0 BasicLayout Module

***This section is informative.***

The functionality in this module is essentially identical with the layout functionality in [SMIL10].

Like SMIL 1.0, SMIL 2.0 BasicLayout module includes a layout model for organizing media elements into regions on the visual rendering surface. The layout element is used in the document head to declare a set of regions on which media elements are rendered. Media elements declare which region they are to be rendered into with the region attribute.

Each region has a set of CSS2 compatible properties such as top, left, height, width, and backgroundColor. These properties can be declared using a syntax defined by the type attribute of the layout element. In this way, media layout can be described using the either SMIL basic layout syntax or CSS2 syntax (note that these are not functionally identical). Other layout types are possible as well.

For example, to describe a region with the id "r" at location 15,20 that is 100 pixels wide by 50 pixels tall using the SMIL BasicLayout module:

```
<layout>
<region id="r" top="15px" left="20px" width="100px" height="50px"/>
</layout>
```

To create the same region using CSS2 syntax:

```
<layout type="text/css">
[regionе="r"] { top: 15px; left: 20px; width: 100px; height:50px; }
</layout>
```

To display a media element in the region declared above, specify the region's id as the region attribute of the media element:

```
<ref region="r" src="http://..." />
```

Additionally, implementations may choose to allow using the CSS syntax to set the media layout directly. This can be done by using the selector syntax to set layout properties on the media elements. For example, to display all video and image elements in a rectangle at the same size and position as the examples above:

```
<layout type="text/css">
video, img { top:15px; left:20px; width:100px; height=50px; }
</layout>
```

Note that multiple layout models could be specified within a control structure such as the SMIL switch element, each with a different type. The first layout with a type supported by the implementation will be the one used.

## 5.3 SMIL 2.0 BasicLayout Module Syntax and Semantics

***This section is normative.***

### 5.3.1 Elements and Attributes

This section defines the elements and attributes that make up the functionality in the SMIL BasicLayout module.

**The layout element**

The layout element determines how the elements in the document's body are positioned on an abstract rendering surface (either visual or acoustic).

The layout element must appear before any of the declared layout is used in the document. If present, the layout element must appear in the head section of the document. If a document contains no layout element, the positioning of the body elements is implementation-dependent.

[illegible] support multiple alternative layouts by enclosing several layout elements within the SMIL switch element. This could also be used to describe the document's layout using different layout languages. Support for the system test attributes in the SMIL BasicContentControl module also enables greater author flexibility as well as user accessibility.

Default layout values can be assigned to all renderable elements by selecting the empty layout element `<layout></layout>`. If the document does not include a layout element, then the positioning of media elements is implementation-dependent.

**Element attributes**

type
: This attribute specifies which layout language is used in the layout element. If the user agent does not understand this language, it must skip the element and all of its content up until the next `</layout>` tag. The default value of the type attribute is "`text/smil-basic-layout`". This identifier value supports SMIL 1.0 and SMIL 2.0 BasicLayout module layout semantics.

*Element content*

If the type attribute of the layout element has the value "`text/smil-basic-layout`", it may contain the following elements:

region
root-layout

Languages incorporating the BasicLayout module need to define what additional elements are allowed as children. If the type attribute of the layout element has another value, the element contains character data.

**The region element**

The region element controls the position, size and scaling of media object elements.

In the following example fragment, the position of a text element is set to a 5 pixel distance from the top border of the rendering window:

```
<smil xmlns="http://www.w3.org/2000/SMIL20/">
 <head>
  <layout>
   <root-layout width="320" height="480" />
    <region id="a" top="5" />
  </layout>
 </head>
 <body>
  <text region="a" src="text.html" dur="10s" />
 </body>
</smil>
```

The position of a region, as specified by its top, bottom, left, and right attributes, is always relative to the parent geometry, which is defined by the parent element. For the SMIL BasicLayout module, all region elements must have as their immediate parent a layout element, and the region position is defined relative to the root window declared in the sibling root-layout element. The intrinsic size of a region is equal to the size of the parent geometry.

When region sizes, as specified by width and height attributes are declared relative with the "%" notation, the size of a region is relative to the size of the parent geometry. Sizes declared as absolute pixel values maintain those absolute values.

Conflicts between the region size and position attributes width, height, bottom, left, right, and top are resolved according to the rules for placeholder [illegible]

elements as detailed below. The default values of region position and size attributes is specified as `auto`. This attribute value has the same meaning here that it does in [CSS2], when there is no distinction drawn between replaced and non-replaced element.

A placeholder element is one which has no intrinsic width or height, but does have a bounding-box which has a width and height. SMIL BasicLayout regions are placeholder elements. Placeholder elements are clipped to the bounding box.

The governing equation for the horizontal dimension is:

bbw (bounding-box-width) = left + width + right

Given that each of these three parameters can have either a value of "auto" or a defined value not "auto", then there are 8 possibilities:

| *Attribute values* | | | *Result before clipping to the bounding box* | | |
|---|---|---|---|---|---|
| **left** | **width** | **right** | **left** | **width** | **right** |
| auto | auto | auto | 0 | bbw | 0 |
| auto | auto | defined | 0 | bbw - right | right |
| auto | defined | auto | 0 | width | bbw - width |
| auto | defined | defined | bbw - right - width | width | right |
| defined | auto | auto | left | bbw - left | 0 |
| defined | auto | defined | left | bbw - right - left | right |
| defined | defined | auto | left | width | bbw - left - width |
| defined | defined | defined | left | width | bbw - left - width |

The vertical attributes height, bottom, and top are resolved similarly. The governing equation for the vertical dimension is:

bbh (bounding-box-height) = top + height + bottom

Given that each of these three parameters can have either a value of "auto" or a defined value not "auto", then there are 8 possibilities:

| *Attribute values* | | | *Result before clipping to the bounding box* | | |
|---|---|---|---|---|---|
| **top** | **height** | **bottom** | **top** | **height** | **bottom** |
| auto | auto | auto | 0 | bbh | 0 |
| auto | auto | defined | 0 | bbh - bottom | bottom |
| auto | defined | auto | 0 | height | bbh - height |
| auto | defined | defined | bbh - bottom - height | height | bottom |
| defined | auto | auto | top | bbh - top | 0 |
| defined | auto | defined | top | bbh - bottom - top | bottom |
| defined | defined | auto | top | height | bbh - top - height |
| defined | defined | defined | top | height | bbh - top - height |

**Element attributes**

backgroundColor
: The use and definition of this attribute are identical to the "background-color" property in the CSS2 specification. This attribute specifies the background color used to fill the area of a region displaying media that is not filled by the media. The display of the background color when the region is not in use by a media element is controlled by the showBackground attribute. Unlike SMIL 1.0, this module requires support for CSS2 system colors.
The backgroundColor attribute may take on the CSS value `inherit`. This means that the background color will be that of the parent element. If the parent element does not have an applicable background color property, the default value of `transparent` is used instead.

background-color
: Deprecated. Equivalent to backgroundColor, which replaces this attribute. The language profile must define whether or not the background-color attribute is supported. If both the backgroundColor and background-color attributes are absent, then the background is `transparent`.

bottom
: The use and definition of this attribute are identical to the "bottom" property in the CSS2 specification. Attribute values can be non-negative "percentage" values, and a variation of the "length" values defined in CSS2. For "length" values, SMIL 2.0 BasicLayout only supports pixel units as defined in CSS2. It allows the author to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2). Conflicts between the region size attributes bottom, left, right, top, width, and height are resolved according to the rules for absolutely positioned, replaced elements in [CSS2]. The default value of this attribute is `auto`.

fit
: This attribute specifies the behavior if the intrinsic height and width of a visual media object differ from the values specified by the height and width attributes in the region element. This attribute does not have a one-to-one mapping onto a CSS2 property, but can be simulated in CSS2. This attribute can have the following values:

  `fill`
  : Scale the object's height and width independently so that the content just touches all edges of the box.

  `hidden`
  : Has the following effect:

    - If the intrinsic height (width) of the media object element is smaller than the height (width) defined in the region element, render the object starting from the top (left) edge and fill up the remaining height (width) with the background color.
    - If the intrinsic height (width) of the media object element is greater than the height (width) defined in the region element, render the object starting from the top (left) edge until the height (width) defined in the region element is reached, and clip the parts of the object below (right of) the height (width).

  `meet`
  : attributes, while none of the content is clipped. The object's left top corner is positioned at the top-left coordinates of the box, and empty

space at the right or bottom is filled up with the background color.

`scroll`
A scrolling mechanism should be invoked when the element's rendered contents exceed its bounds.

`slice`
Scale the visual media object while preserving its aspect ratio so that its height or width are equal to the value specified by the height and width attributes while some of the content may get clipped. Depending on the exact situation, either a horizontal or a vertical slice of the visual media object is displayed. Overflow width is clipped from the right of the media object. Overflow height is clipped from the bottom of the media object.

The default value of fit is `hidden`.

Note that the fit attribute applies to visual media once it has an intrinsic two-dimensional size, such as images and video. It does not apply to visual media that is rendered and adapted to varying circumstances, such as the visual display of HTML, until its two-dimensional spatial dimensions have been determined, such as after an HTML page has been laid out to specific size.

height
The use and definition of this attribute are identical to the "height" property in the CSS2 specification. Attribute values follow the same restrictions and rules as the values of the bottom attribute. The intrinsic height of a region is the same as that of the parent geometry. The default value of this attribute is `auto`.

left
The use and definition of this attribute are identical to the "left" property in the CSS2 specification.
Attribute values follow the same restrictions and rules as the values of the "bottom" attribute.
The default value of this attribute is `auto`.

regionName
This attribute assigns a name to this region element that can be referred to by the region attribute of media object elements. The regionName attribute is not a unique identifier; multiple region elements can share the same regionName attribute value.

right
The use and definition of this attribute are identical to the "right" property in the CSS2 specification.
Attribute values follow the same restrictions and rules as the values of the "bottom" attribute.
The default value of this attribute is `auto`.

showBackground
This attribute controls whether the backgroundColor of a region is shown when no media is being rendered to the region:

- If the value of showBackground is `always`, then the background color will be shown in the region when no media object is rendering into that region. If the region is part of a hierarchical layout, then any ancestor regions must also either be active or have a showBackground value of `always` for the background color to be shown.
- If the value of showBackground is `whenActive`, then the background color will be not be shown in the region when no media object is rendering into that region. If the region is part of a hierarchical layout, then the background color will also be shown when any descendent regions are active.

The default value of showBackground is `always`.

top
The use and definition of this attribute are identical to the "top" property in the CSS2 specification.
Attribute values follow the same restrictions and rules as the values of the bottom attribute.
The default value of this attribute is `auto`.

width
The use and definition of this attribute are identical to the "width" property in the CSS2 specification.
Attribute values follow the same restrictions and rules as the values of the bottom attribute. The intrinsic width of a region is the same as that of the parent geometry.
The default value of this attribute is `auto`.

z-index
The use and definition of this attribute are identical to the "z-index" property in the CSS2 specification, with the following exception:

If two boxes generated by elements A and B have the same stack level, then:

- If the display of an element A starts later than the display of an element B, the box of A is stacked on top of the box of B (temporal order).
- Else, if the display of the elements starts at the same time, and an element A occurs later in the SMIL document text than an element B, the box of A is stacked on top of the box of B (document tree order as defined in CSS2).

A profile integrating the SMIL 2.0 BasicLayout module must provide a means of declaring an XML identifier on region elements.

**The root-layout element**

The root-layout element determines the value of the layout properties of the root element, which in turn determines the size of the window in which the SMIL presentation is rendered.

If more than one root-layout element is parsed within a single layout element, this is an error, and the document should not be displayed. This does not include root-layout elements skipped by the user agent (e.g. because the enclosing layout element was skipped due to an unrecognized type or a test attribute evaluated to false).

The semantics of the root-layout element are as in SMIL 1.0: the attributes of the root-layout element determine the size of the top level presentation window, and the declared sibling regions are arranged within this top level window. If either the height or width of the root-layout element is not specified, the value of the attribute is implementation-dependent.

**Element attributes**

The root-layout element can have the following attributes:

backgroundColor
Defined in backgroundColor under the region element. Note that the default background color is `transparent`, which implies that, by default, the implementation-dependent window background will be shown.

background-color
Defined in background-color under the region element.

height
Sets the height of the root element. Only length values are allowed. For "length" values, SMIL 2.0 BasicLayout only supports pixel units as defined in CSS2. It allows the author to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2).

width

Sets the width of the root element. Only length values are allowed. For "length" values, SMIL 2.0 BasicLayout only supports pixel units as defined in CSS2. It allows the author to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2).

**Element content**

The root-layout element is an empty element.

This element supports the SMIL 1.0 syntax where the root-layout element is an empty sibling of the top level region elements.

**The region attribute**

The region attribute is applied to an element in order to specify which rendering region is assigned to the element. The attribute refers to the abstract rendering region (either visual or acoustic) defined within the layout section of the document. The referenced abstract rendering region is determined by applying the following rules, in order:

1. Find all elements in the layout section with regionName attributes that are assigned the same value as that of the region attribute.
2. Remove the elements from this collection that are removed from the rendered presentation due to the processing of switch elements and test attributes.
3. If any elements remain, the media should be rendered in all of the referred to regions. If the application cannot render the media simultaneously in multiple regions, then the media should be rendered using the lexically first remaining element.
4. If no elements have a regionName attribute that is assigned the same value as that of the region attribute, then select the element in the layout section whose unique identifier is the value of the region attribute.

If this process selects no rendering surface defined in the layout section, the values of the formatting properties of this element are defined by the default layout values, which is described in the section on integration requirements for this module.

The language integrating this module must specify which elements have a region attribute and any inheritance of the attribute.

### 5.3.2 SMIL BasicLayout Module Details

SMIL 2.0 BasicLayout module is consistent with the visual rendering model defined in CSS2, it reuses the formatting properties defined by the CSS2 specification, and newly introduces the fit attribute [CSS2]. The reader is expected to be familiar with the concepts and terms defined in CSS2.

SMIL 2.0 layout regions influence the propagation of user interface events (such as a mouse click, or hyperlink activation) to underlying visible elements. When the location of an event corresponds to the background of a region rather than the media that is displayed in that region, a region background color of `transparent` allows user interface events to pass through to elements lower in the display stacking order. Conversely, regions with non-`transparent` background colors will capture user interface events, not allowing the event to pass through to elements lower in the display stacking order. This behavior is separate from that of a language profile's ability to make use of user interface events captured by region elements.

**Integration Requirements**

A profile integrating the SMIL 2.0 BasicLayout module must define the content models for the layout element if any elements beyond those specified here are to be allowed as children.

A profile integrating the SMIL 2.0 BasicLayout module must provide a means of declaring an XML identifier on region elements if the profile intends on referring to region elements by XML identifier. This value is used as the argument value to the region attribute. This is not required if the profile will only use the regionName method of referring to a region element.

A profile integrating the SMIL 2.0 BasicLayout module must specify which elements have a region attribute and any inheritance of the attribute.

If not otherwise defined by the profile, the default values of the layout attributes listed in the SMIL 2.0 layout modules will apply to presented elements not otherwise specifying layout semantics.

An element that does not refer to a valid region element will display in the default region. If not otherwise specified by the profile, the default region is defined as filling and aligned to the upper-left corner of the presentation window. This default region takes on default values for all other region attributes.

### 5.3.3 Document Type Definition (DTD) for the BasicLayout Module

See the full DTD for the SMIL Layout modules.

## 5.4 Overview of the SMIL 2.0 AudioLayout Module

***This section is informative.***

This section defines the functionality in the SMIL 2.0 AudioLayout module. This level contains an attribute providing audio rendering surface volume control.

### 5.4.1 Integration Requirements for the AudioLayout Module

***This section is normative.***

The functionality in this module builds on top of the functionality in the BasicLayout module, which is a required prerequisite for inclusion of the AudioLayout module.

### 5.4.2 Audio Volume Control

***This section is normative.***

SMIL 2.0 AudioLayout module supports control of aural media volumes via a new property on the region element, soundLevel. Multimedia assigned to

## 5.5 SMIL 2.0 AudioLayout Module Syntax and Semantics

***This section is normative.***

### 5.5.1 Elements and Attributes

This section defines the elements and attributes that make up the SMIL 2.0 AudioLayout module.

**The region element**

The region element defined in the BasicLayout module is extended with the addition of the soundLevel attribute.

**Element attributes**

The region element can have the following aural attributes:

soundLevel
: Specifies the relative volume of the audio portion of a media element assigned to play within the given region. This associates the region element with a sound reproduction unit. Cascaded region's will accumulate their respective sound level settings, as will be explained below. region's that are used for multiple sources apply their sound level setting to all of them. A sound source may be reproduced by different units, e.g., through application of the regionName attribute. In such a "multiple window" case a separate soundLevel may be applied to each instance of the sound source, one per region. Assigned level changes accumulate across nested regions by multiplying percentage values.

  Valid values are non-negative CSS2 percentage values. Percentage values are interpreted relative to the recorded volume of the media. The percentages are interpreted as a ratio of output to input signal level, and is defined in terms of dB:

  $$\text{dB change in signal level} = 20 \log_{10}(\text{percentage-value} / 100)$$

  A setting of '0%' plays the media silently. A value of '100%' will play the media at its recorded volume (0 dB). Similarly, a value of '200%' will play the media nearly twice as loud (6 dB) as its recorded volume (subject to hardware limitations). The default value is '100%'. The absolute sound level of media perceived is further subject to system volume settings, which cannot be controlled with this attribute.

## 5.6 Overview of the SMIL 2.0 MultiWindowLayout Module

***This section is informative.***

This section defines the functionality in the SMIL 2.0 MultiWindowLayout module. This level contains elements and attributes providing for creation and control of multiple top level windows on the rendering device.

### 5.6.1 Integration Requirements for the MultiWindowLayout Module

***This section is normative.***

The functionality in this module builds on top of the functionality in the BasicLayout module, which is a required prerequisite for inclusion of the MultiWindowLayout module.

### 5.6.2 Multiple Top-Level Window Support

***This section is normative.***

In [[SMIL 10]], and the SMIL 2.0 BasicLayout module, each presentation is rendered into a single root window of a specific size/shape. The root window contains all of the regions used to manage the rendering of specific media objects.

This specification supports the concept of multiple top-level windows. Since there is no longer a single root window, we use the term *top level* instead. The assignment of the regions to individual top level windows allows independent placement and resizing of each top-level window, if supported by the including profile and implementation. The initial placement of the top level windows on the display device and any available means of relocating the top level windows is implementation-dependent.

A top level window is declared with the topLayout element in a manner similar to the SMIL 1.0 root-layout window, except that multiple instances of the topLayout element may occur within a single layout element:

```
<layout>
  <topLayout id="WinV" title="Video" width="320" height="240"/>
    <region id="pictures" title="pictures" height="100%" fit="meet"/>
  <topLayout id="WinC" title="Captions" width="320" height="60">
    <region id="captions" title="caption text" top="90%" fit="meet"/>
  </topLayout>
</layout>
```

In this example, two top-level windows are defined ("WinV" and "WinC"), and two regions are defined with one region ("pictures") assigned to WinV and the other ("captions") to WinC. The definitions of the top-level windows and the contained regions use a hierarchical syntax, unlike the older root-layout element.

The top-level windows function as rendering containers only, that is, they do not carry temporal significance. In other words, each window does not define a separate timeline or any other time-container properties. There is still a single master timeline for the SMIL presentation, no matter how many top-level windows have been created. This is important to allow synchronization between media displayed in separate top-level windows.

The display of top level windows can be controlled automatically by the player, or manually by the user of the application. If a window is closed (by the user) while any of the elements displayed in that window are active, there is no effect on the timeline (if any) of those elements. However, a player may choose not to decode content as a performance improvement. The means provided to a user to close top level windows is implementation-dependent.

For SMIL 1.0 compatibility, the root-layout element will continue to support SMIL 1.0 layout semantics. The new topLayout element will support the extension semantics and the improved, nested syntax.

Note also that any one region may belong to at most one top-level (or root-level) window. Regions not declared as children of a topLayout element belong to the root-layout window. If no root-layout element has been declared, the region is assigned to an additional window according to the semantics in the BasicLayout module.

## 5.7 SMIL 2.0 MultiWindowLayout Module Syntax and Semantics

***This section is normative.***

### 5.7.1 Elements and Attributes

This section defines the elements and attributes that make up the SMIL 2.0 MultiWindowLayout module.

**The topLayout element**

The topLayout element determines the size of the a window in which the SMIL presentation is rendered, as well as serving as a top level window in which to place child region elements.

Multiple topLayout elements may appear within a single layout element, each declaring an independent top-level window.

Each instance of a topLayout element determines the size of a separate top-level presentation window, and the descendant regions are arranged within this top-level window and relative to the coordinate system of this window.

This module also provides control over when topLayout windows open and close in a presentation. Note that the precise mapping of topLayout windows on to the host environment is implementation-dependent. It is expected that implementations will "pop up" independent desktop windows if they can, but other means of supporting multiple topLayouts, such as by using frames, are allowed. When automatically opening and closing windows, applications should try to comply with the WAI User Agent Guidelines [UAAG] and allow the user to choose whether to be warned that windows are being opened and closed, and give a method for disabling automatic opening and closing of windows.

**Element attributes**

The topLayout element can have the following attributes:

backgroundColor
: Defined in backgroundColor under the region element. Note that the default background color is `transparent`, which implies that, by default, the implementation-dependent window background will be shown.

close
: Specifies when the top level window should be closed. If the value of close is `onRequest`, then the top level window should not be closed automatically by the player and will only close if the user explicitly closes it via the user interface. If the value of close is `whenNotActive`, then the top level window should close automatically when no media is being displayed in any one of the window's regions. For timed media using the SMIL 2.0 timing and synchronization modules, this means when there is no media within its active duration or freeze period using any region of the topLayout. The default value of close is `onRequest`.

height
: Sets the height of the top-level window. Only length values are allowed.

open
: Specifies when the top level window should be opened. If the value of open is `onStart`, then the top level window should be opened when the presentation begins, and if closed, should not be reopened automatically during the presentation. If the value of open is `whenActive`, then, if not already open, the top level window should be opened when media is displayed in one of the window's regions. For timed media using the SMIL 2.0 timing and synchronization modules, this means when there is any media within its active duration or freeze period using any region of the topLayout. The default value of open is `onStart`.

width
: Sets the width of the top-level window. Only length values are allowed.

*Element content*

The topLayout element may contain any number of region elements, or be empty.

Allowing multiple topLayout elements within a single layout element supports multiple top level windows.

**The layout element**

*Element content*

The layout element defined in the SMIL BasicLayout module is extended with the addition that the topLayout element is added to the content model of the layout element if the type attribute of the layout element has the value "`text/smil-basic-layout`". In this case it can contain the following elements:

region
root-layout
topLayout

### 5.7.2 MultiWindowLayout Module Events

This module includes two events that may be included in the integrating language profile.

**topLayoutOpenEvent**
: Raised when a topLayout window opens. This event is delivered to the associated topLayout element. If a topLayout closes and then reopens when additional media becomes active in any of its regions, this event will be raised again, and will be raised every subsequent time it reopens.

**topLayoutCloseEvent**
: Raised when a topLayout closes for any reason. This event is delivered to the associated topLayout element. If a topLayout reopens when additional media becomes active in any of its regions, this event will be raised again if and when the topLayout closes again, and will be raised

**Integration Requirements for the MultiWindowLayout Module Events**

The language profile must specify the declarative names for binding these events, as well as the bubbling behavior of the events.

### 5.7.3 Document Type Definition (DTD) for the MultiWindowLayout Module

5.7.6 Document Type Definition (DTD) for the MultiWindowLayout Module

See the full DTD for the SMIL Layout modules.

## 5.8 Overview of the SMIL 2.0 HierarchicalLayout Module

***This section is informative.***

This section defines the functionality in the SMIL 2.0 HierarchicalLayout module. This module contains elements and attributes for advanced positioning of media elements on the visual rendering surface and builds upon the SMIL 2.0 BasicLayout module.

The SMIL 2.0 HierarchicalLayout module extends the basic layout model for organizing media elements into regions on the visual rendering surface providing much greater author control and flexibility. These extensions are important for certain classes of multimedia presentations in which author control of object placement is critical.

This module:

- extends the definition of the region element to allow for the specification of hierarchical regions;
- introduces a new layout element, regPoint, for controlling relative placement with an associated region (the *registration element* facility);
- includes additional attributes, regPoint and regAlign, that specify how a media object's presentation can be aligned relative to the set of defined registration points (the *registration alignment* facility); and
- provides additional attributes, top, bottom, left, and right, that specify exact positioning of objects within a region (the so-called *sub-region positioning* facility);
- allows an object placed in a sub- region to set the sub- region's fit, z-index, and backgroundColor attributes.

### 5.8.1 Integration Requirements for the HierarchicalLayout Module

***This section is normative.***

The functionality in this module builds on top of the functionality in the BasicLayout module, which is a required prerequisite for inclusion of the HierarchicalLayout module.

## 5.9 New Features in the SMIL 2.0 HierarchicalLayout Module

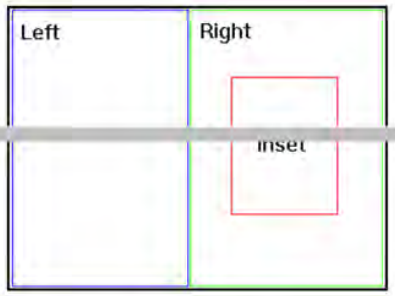
***This section is normative.***

### 5.9.1 Hierarchical Region Layout

A new feature in this module is support for hierarchical layout. This allows for the declaration of regions nested inside other regions, much like regions are laid out inside the top level window declared by the topLayout element. For example, the following declares a top level window of 640 by 480 pixels, regions "left" and "right" which covers the left and right sides of the window respectively, and a hierarchical region "inset" that is centered within "right".

```
<layout>
        <topLayout width="640px" height="480px" />
                <region id="left" top="0%" left="0%"
                        width="50%" height="100%" />
                <region id="right" top="0%" left="50%"
                        width="50%" height="100%">
                        <region id="inset" top="25%" left="25%"
                                width="50%" height="50%" />
                </region>
        </topLayout>
</layout>
```

The resulting layout looks like this:




By default, each hierarchical region shares the z-index (depth) value of its parent. Hierarchical regions may also introduce their own local z-index value. In this case, all hierarchical regions with a common direct parent define local z-indexes within the z-index value of their parent. For example, if a parent region has a z-index value of "4" and two hierarchical children of that parent define z-indexes of "1" and "2", respectively, then each of these are treated as further sub-divisions of the parent's z-index of "4".

If two hierarchical regions with the same z-index attribute value overlap, the existing rules for z-index processing (from the BasicLayout module) are applied. Specifically, the rule concerning time priority is maintained, meaning that in the case of a z-index conflict, the media visible in the overlap will be determined by the region that is rendering the media that has most recently begun in time. If the conflicting media began at the same time, then the rule using the textual order of the media elements in the SMIL document is applied.

For example:

```
<layout>
   <root-layout width="640px" height="480px" />
   <region id="whole" top="0px" left="0px" width="640px"
                              height="480px" z-index="5"/>
   <region id="right" top="0px" left="320px" width="320px"
                              height="480px" z-index="4">
       <region id="inset" top="140px" left="80" width="160px"
```

```
                        height="200px" z-index="6"/>
      <region id="inset2" top="140px" left="80" width="160px"
                        height="200px" z-index="6"/>
      <region id="inset3" top="140px" left="80" width="160px"
                        height="200px" z-index="7"/>
    </region>
</layout>
...
<par>
      <img id="A" region="whole" src="imageA.jpg" dur="10s"/>
      <img id="B" region="inset" src="imageB.jpg" dur="10s"/>
</par>
<par>
      <img id="C" region="inset" src="imageC.jpg" begin="0s" dur="10s"/>
</par>
<par>
      <img id="E" region="inset2" src="imageE.jpg" dur="10s"/>
      <img id="F" region="inset3" src="imageF.jpg" dur="10s"/>
</par>
```

1. In the first "par", image "A" and image "B" begin at the same time. Image A will obscure image "B", even though the z-index of "inset" is greater than that of "whole". This is because the z-index of "right", which is the region containing "inset" is less than that of "whole".
2. In the second "par", images "C" and "D" are rendered into regions occupying the same area of the rendering surface. Image "C" will be shown for one second and then obscured by Image "D", since "D" begins after image "C". Note that lexical order is irrelevant here.
3. In the third "par", the z-index of region "inset" is considered when computing stacking between siblings, and therefore image "F" will be shown, but image "E" will be obscured for the entire 10 seconds that they are both active.

### 5.9.2 Sub-Region Layout

Where hierarchical layout provides a facility for defining a set of regions with a common parent, it does not provide any facility for finer control within a region where a given media object will be placed. The SMIL 2.0 HierarchicalLayout module solves this problem by defining a set of attributes which, when placed on an object using the region attribute, allow that media item to be declared with an explicitly positioned sub-region of a given region. These attributes are collectively referred to as *sub-region positioning attributes*. A sub-region is a child of the region declared in the region attribute on the media element. The sub-region positioning argument values follow the conventions of placeholder elements described in the section on the region element .

For example, suppose a region *"d"* is defined:

```
<layout>
  ...
  <region id="d" ... />
  ...
</layout>
```

The following code describes the placement of an object in a sub-region at a particular offset within a region, using the SMIL 2.0 HierarchicalLayout syntax:

```
<ref id="a" ... region="d" top="5%" left="3" />
```

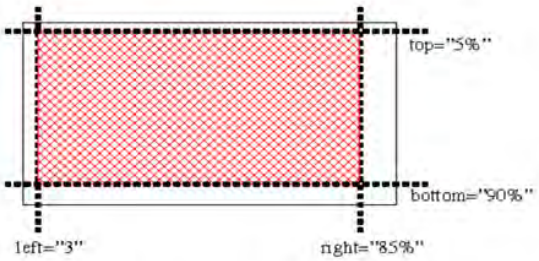
displayed 5% from the top and 3 pixels from the left of region *"d"*, and extends to the right and bottom edges of the region *"d"*.

All other placement operations, such as the fit attribute, now operate on the sub-region. For example, the following document fragment describes a region and a media object reference that make use of sub-region positioning:

```
<layout>
  ...
  <region id="d" ... fit ="fill" />
  ...
</layout>

<body>    ...
  <ref src="..." ... region="d" fit="hidden"
           top="5%" left="3" bottom="10%" right="15%" />
  ...
</body>
```

In this example, the effective boundaries of the sub-region for the placement of this object are defined by declaring the top, bottom, left and right edges of the region to the values shown, and then filling the resulting sub-region with the specified image as directed by the fit attribute. If the size of the media object being displayed is smaller than that of the resulting sub-region, the display will be similar to:




The use of sub-region placement is intended as a light-weight alternative to defining a large number of single-use regions. Often, the dimensions used

object in the sub-region. The other attributes on the media element that would have been applied to a referenced region are applied to the sub-region instead. Note that the default values for the sub-region attributes are all 'auto', resulting that, by default, a sub-region is created having the same size and position as the parent region.

Rules for handling clipping of objects within regions based on the sub-region attributes are provided below.

### 5.9.3 Media Object fit, z-index, and backgroundColor

The SMIL 2.0 HierarchicalLayout module includes the ability to use the fit, z-index, and backgroundColor attributes on objects displayed in a sub-region in order to declare different behavior from that on the region element.

### 5.9.4 Registration Points

A registration element is an element defined within this module that is used to define a point within a region and a default object alignment algorithm about that point. The element can be used in a media object element, where it is associated with a region and an optional override alignment algorithm. The placement values within registration elements can be either percentages or pixels.

The use of registration points allows for consistent relative placement across regions. As such, registration points are defined outside of any single region.

For example, the following code describes two registration points (with id values "midPoint" and "topMargin"), one of which is defined as a relative location and one at a fixed pixel location, using the SMIL 2.0 HierarchicalLayout syntax:

```
<layout>
  <regPoint id="midPoint" top="50%" left="50%" regAlign="center" />
  <regPoint id="topMargin" top="10" left="15" regAlign="topLeft" />
  <region id="a" ... />
  <region id="b" ... />
</layout>
```

In this example, the registration point with the id value "midPoint" has a default alignment algorithm that centers the media object about the defined point, while the registration point with the id value "topMargin" has a default alignment algorithm that places the top-left point of the media object at the registration point.

Various media elements could be displayed in the regions using the alignment points as follows:

```
<ref region="a" src="rtsp://..." dur="2s" regPoint="midPoint" />
<ref region="b" src="http://..." dur="2s"
                regPoint="midPoint" regAlign="bottomRight"/>
<ref region="a" src="http://..." dur="2s" regPoint="topMargin" />
<ref region="b" src="http://..." dur="2s"
                regPoint="topMargin" regAlign="center"/>
```

In the first example, a media object is centered in the middle of region *a*. In the second example, a media object has its bottom right corner centered in the middle of region *b*. Similarly, in the third example, a media object has its top left corner placed at a point 10,15 within region *a*, and in the fourth example, an object is centered around the point 10,15 in region *b*.

Registration points can be used to coordinate the placement of a set of media objects that do not share the same sizes. (For example, a set of images can be aligned to the center of a region.) They can also be used to coordinate the display of images about a particular point in a region, as in:

```
<layout>
   <regPoint id="middle" top="50%" left="50%" regAlign="center" />
   <region id="a" ... />
</layout>
...
<seq>
 <ref region="a" src="rtsp://..." dur="2s" regPoint="middle"
                                  regAlign="bottomRight"/>
 <ref region="a" src="http://..." dur="2s" regPoint="middle"
                                  regAlign="bottomLeft"/>
 <ref region="a" src="http://..." dur="2s" regPoint="middle"
                                  regAlign="topLeft"/>
 <ref region="a" src="http://..." dur="2s" regPoint="middle"
                                  regAlign="topRight"/>
</seq>
```

In this example, four objects are aligned over time to the middle of region. If any media element extends outside the bounds of a region, it will be clipped to the region.

Note that registration points are global within the context of a layout, and are not tied to a particular region, but can be reused across regions. As such, pixel-based offsets should be used with care.

For authoring convenience, SMIL HierarchicalLayout module provides several pre-defined region registration points including `topLeft`, `topMid`, `topRight`, `midLeft`, `center`, `midRight`, `bottomLeft`, `bottomMid`, and `bottomRight`.

```
<ref ..." regPoint="center" regAlign="center" />
```

The default value of regAlign for a region is `topLeft`. If the regAlign attribute is used without a regPoint attribute, the alignment operation is relative to the upper left point of the region containing this object, that is, the behavior is the same as if the regPoint were to be specified as `topLeft`.

Rules for handling clipping of objects within regions based on the regPoint and regAlign attributes are defined below.

## 5.10 SMIL 2.0 HierarchicalLayout Syntax and Semantics

***This section is normative.***

### 5.10.1 Elements and attributes

This section defines the elements and attributes that make up the SMIL 2.0 HierarchicalLayout module.

**The layout element**

This element is defined as in the BasicLayout module with extensions presented here.

In order to support the registration point functionality described above, one new element is added to the content model of the layout element.

**Element attributes**

The SMIL HierarchicalLayout module does not provide any new attributes to the layout element.

**Element content**

**Element content**

If the type attribute of the layout element has the value "`text/smil-basic-layout`", it can contain the following elements:

region
root-layout
topLayout

All element content except regPoint are defined above in the BasicLayout and MultiWindowLayout modules. The regPoint element is described below.

**The region element**

The region element controls the position, size and scaling of media object elements. This module extends the definition of the region element to include the definition of hierarchical regions.

The position of a region, as specified by its top and left attributes, is always relative to the parent geometry, which is defined by the parent element. For the SMIL 2.0 HierarchicalLayout module, all hierarchical region elements must have as their immediate parent a region or topLayout element. The position of the hierarchical region is defined relative to that parent element. The intrinsic size of a region is equal to the size of the parent geometry.

When region sizes, as specified by width and height attributes are declared relative with the "%" notation, the size of the hierarchical region is relative to the size of the parent region. Sizes declared as absolute pixel values are absolute values even when used with a child region.

Note that a (hierarchical) region may be defined in such a way as to extend beyond the limits of its parent. In this case the child region must be clipped to the parent boundaries.

If a z-index attribute is defined on the hierarchical region, it is evaluated as a local index within that of the parent.

**Element attributes**

In the HierarchicalLayout module, the region element has no additional attributes beyond that provided in the other included layout modules. However, the semantics of the z-index attribute are extended to support hierarchical regions.

z-index
: This attribute is defined as in the BasicLayout module with extensions presented here.

  The z-index attribute defines the level of the region within the parent region stacking context. Elements assigned to higher level regions are rendered in front of lower level regions within the same parent region. Child regions are always placed in front of their parent region. This results in a two stage sorting of region visibility: first by parent-child containment, and then by z-index among siblings.

  Just as with simple non-hierarchical regions, the stacking order of hierarchical regions may be affected by temporal activation. A region becomes active either when media begins rendering into it, or when one of its child regions becomes active. If two sibling regions have the same z-index, the region most recently made active is in front of the other region.

**Element content**

SMIL HierarchicalLayout module extends the region element content model to optionally include other region elements.

**The regPoint element**

The regPoint element determines the (x, y) coordinates of a point relative to a region upper-left corner for use in aligning elements in the document's body on regions within a visual rendering surface. A regPoint may be defined using absolute (pixel) or relative (percentage) based values. The regPoint functionality is not defined and may not be used for media without intrinsic size.

For the purposes of regPoint functionality, media and regions are defined to be rectangular, with perpendicular sides, with the sides ordered clockwise top, right, bottom, and left. The top side is the edge closest to the point or edge of the display device considered "up".

The regPoint element may only appear as an immediate child of a layout element.

If the registration point functionality is used on a media object that also uses sub-region positioning, the registration point applies to the subregion.

If the registration point or alignment functionality is used on a media object, the interaction between the regPoint attribute value, the regAlign attribute value, and the fit attribute value of the region in which the media object is displayed is as follows:

`fill` fit **value:**
: This depends on the value of the regAlign attribute. (Note: in all cases, the media must maintain proper alignment over the regPoint):

  `topLeft` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the bottom and right edges of the box.

  `topMid` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the bottom edge of the box, and the nearest (to the regPoint) of the left or right edges of the box.

  `topRight` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the bottom and left edges of the box.

  `midLeft` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the nearest (to the regPoint) of the top or bottom edges of the box, and touches the right edge of the box.

  `center` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the nearest (to the regPoint) of the top or bottom edges of the box, and touches the nearest (to the regPoint) of the left or right edges of the box.

  `midRight` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the nearest (to the regPoint) of the top or bottom edges of the box, and touches the left edge of the box.

  `bottomLeft` regAlign **value:**
  : Scale the object's height and width independently so that the content just touches the top and right edges of the box.

`bottomMid` regAlign **value:**
Scale the object's height and width independently so that the content just touches the top edge of the box, and the nearest (to the regPoint) of the left or right edges of the box.
`bottomRight` regAlign **value:**
Scale the object's height and width independently so that the content just touches the top and left edges of the box.

`hidden` **fit value:**
Align the media over the given regPoint per the regAlign attribute. If the media so positioned extends past the bounds of the region, clip the excess media. If the media so positioned doesn't meet the bounds of the region, fill the remaining space with the background color of the region.
`meet` **fit value:**
This depends on the value of the regAlign attribute. Any part of the region that is not covered by the media is then filled with the region's background color. (Note: in all cases, the media must maintain proper alignment over the regPoint):

`topLeft` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the right or bottom edge of the box while none of the content is clipped.
`topMid` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches at least one of the left, right, or bottom edges while none of the content is clipped.
`topRight` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the left or bottom edge of the box while none of the content is clipped.

Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches at least one of the top, right, or bottom edges while none of the content is clipped.
`center` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches at least one of the top, left, right, or bottom edges while none of the content is clipped.
`midRight` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches at least one of the top, left, or bottom edges while none of the content is clipped.
`bottomLeft` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top or right edge of the box while none of the content is clipped.
`bottomMid` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches at least one of the top, left, or right edges while none of the content is clipped.
`bottomRight` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top or left edge of the box while none of the content is clipped.

`scroll` **fit value:**
Align the media over the given regPoint per the regAlign attribute. If any part of the media extends beyond the bounds of the region, a scrolling mechanism should be invoked that allows viewing of the media that is clipped beyond the bounds of the region.
`slice` **fit value:**
This depends on the value of the regAlign attribute. (Note: in all cases, the media must maintain proper alignment over the regPoint):

`topLeft` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the right or bottom edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped.
`topMid` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the left, right, or bottom edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped. Clipping will occur on up to two sides of the region.
`topRight` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the left or bottom edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped.
`midLeft` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top, right, or bottom edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped. Clipping will occur on up to two sides of the region.
`center`regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top, left, right, or bottom edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped. Clipping will occur on up to three sides of the region.
`midRight` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top, left, or bottom edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped. Clipping will occur on up to two sides of the region.
`bottomLeft` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top or right edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped.
`bottomMid`regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top, left, or right edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped. Clipping will occur on up to two sides of the region.
`bottomRight` regAlign **value:**
Scale the visual media object's height and width while maintaining its aspect ratio so that the content just touches the top or left edge of the box, whichever requires the largest scale factor. Any content that extends beyond the bounds of the region is clipped.

For example, a wide-screen video can be made to play in "letterbox" mode in a region, whose width-to-height ratio is smaller, by using regPoint ="`center`" and regAlign="`center`" and setting the region's fit value to "`meet`". The result is the video will touch the left and right edges of the region and will be centered vertically with the gaps above and below filled in with the region's background color.

**Element attributes**

top
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the registration point relative to the top of the region. The default value of top is `auto`.

bottom
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the registration point relative to the bottom of the region. The default value of bottom is `auto`.

left
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the registration point relative to the left location of the region. The default value of left is `auto`.

right
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the registration point relative to the right location of the region. The default value of right is `auto`.

regAlign
This attribute specifies the default alignment algorithm which is associated with a regPoint relative to the media object. If no value is specified the value of `topLeft` is the default for the regPoint. The following values are allowed:

`topLeft`
Align the top-left corner of the object with the registration point.

`topMid`
Align the top-middle point of the object with the registration point. The top-middle is the point offset width/2 to the right of the object top-left corner.

`topRight`
Align the top-right corner of the object with the registration point.

`midLeft`
Align the middle-left point of the object with the registration point. The middle-left is the point offset height/2 down from the object top-left corner.

`center`
Align the center of the object with the registration point.

`midRight`
Align the middle-right point of the object with the registration point. The middle-right is the point offset height/2 down from the object top-right corner.

`bottomLeft`
Align the bottom-left corner of the object with the registration point.

`bottomMid`
Align the bottom-middle point of the object with the registration point. The bottom-middle is the point offset width/2 right from the object bottom-left corner.

`bottomRight`
Align the bottom-right corner of the object with the registration point.

**Element content**

None.

### 5.10.2 SMIL HierarchicalLayout Positioning and Presentation Attributes

While the SMIL 2.0 BasicLayout module provides only the region attribute on elements to place them on the rendering surface, the HierarchicalLayout module includes attributes to refine the position of media content within a region, and to refine the visual presentation of the media within the region.

This module provides fine control over the background color surrounding media elements by allowing the media element to declare the background color of a region in which the media is being shown.

One set of attributes (the *sub-region positioning* attributes) allows a sub-region to be defined that is a child of a declared region and is wholly contained within the enclosing layout region for that media object; the other set of attributes can be used to define a registration point to be used with that object and an optional layout algorithm that will override the default algorithm associated with the registration point.

If the fit attribute and alignment attributes regPoint and regAlign are relevant to the placement of a particular media object, the interaction is the same as described in the definition of regPoint. If sub-region positioning attributes are used on a media object along with fit or the alignment attributes regPoint and regAlign, these attributes apply to the sub-region. In this case the fit setting on the referenced region element does not apply to the sub-region.

For both sub-region positioning and registration point use, the value of the z-index attribute on the associated region is used. If media objects overlap spatially, existing rules for resolving z-index conflicts are applied.

Note that placement within the region may be defined in such a way as to extend the media object beyond the limits of the region. In this case the media object must be clipped to the region boundaries.

**Media specific background color**

backgroundColor
This attribute allows specifying the background color that will be used when a media element is being shown in a sub-region. The range of legal values are the same as the backgroundColor attribute on the region element. The default value for this attribute is `transparent`.

**Sub-region Positioning Attributes**

top
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the top edge of the sub-region relative to the top of the region. The default value of top is `auto`.

bottom
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the bottom edge of the sub-region relative to the bottom of the region. The default value of bottom is `auto`.

height
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the bottom edge of the sub-region relative to the top side of the region. The default value of height is `auto`.

left
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the left edge of the sub-region relative to the left of the region. The default value of left is `auto`.

right
This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the right edge of the sub-

region relative to the right side of the region. The default value of right is `auto`.

width
: This value uniquely identifies the position or the distance (using CSS2 pixel or non-negative percentage values) of the right edge of the sub-region relative to the left side of the region. The default value of width is `auto`.

Conflicts between the region size attributes bottom, height, left, right, top, and width are resolved according to the rules for placeholder elements described in the section on the region element.

The sub-region positioning attributes will be ignored if they are used on an element without a region attribute that specifies the identifier of region element in the layout section.

**Registration point attributes**

The regPoint attribute is used in conjunction with the regPoint element. If a regPoint attribute is missing or refers to a non-existent regPoint element the value of the regAlign attributes are applied to the top-left point of the region containing the media object.

regPoint
: This value uniquely identifies the registration point to be used for the placement of the object. The value should be an XML identifier of a regPoint

The following values are predefined registration points that do not have to be declared as regPoint elements before they are used:

`topLeft`
: Same as using top="0%" left="0%" on the element without the regPoint attribute.

`topMid`
: Same as using top="0%" left="50%" on the element without the regPoint attribute.

`topRight`
: Same as using top="0%" left="100%" on the element without the regPoint attribute.

`midLeft`
: Same as using top="50%" left="0%" on the element without the regPoint attribute.

`center`
: Same as using top="50%" left="50%" on the element without the regPoint attribute.

`midRight`
: Same as using top="50%" left="100%" on the element without the regPoint attribute.

`bottomLeft`
: Same as using top="100%" left="0%" on the element without the regPoint attribute.

`bottomMid`
: Same as using top="100%" left="50%" on the element without the regPoint attribute.

`bottomRight`
: Same as using top="100%" left="100%" on the element without the regPoint attribute.

Note that the predefined registration points have the same meaning relative to the region as the regAlign attribute values have relative to the media. If no value is given, the default registration point of `topLeft` (top="0%", left="0%") is assumed, that is, the media is aligned with the top-left of the region.

regAlign
: This value uniquely identifies the registration algorithm to be used for the regPoint defined for the object, and overrides any regAlign attribute on the referenced regPoint element. Permissible values are those defined under the regAlign attribute for the regPoint element. If used without an explicit regPoint attribute, the value is relative to the top left point of the region used by the associated media object.

**Region fit override**

The fit attribute is used on an element in conjunction with the regionattribute to control the display of the element on the rendering surface. A value of the fit attribute given on the element will override the value of fit declared in the referenced region element.

fit
: This attribute specifies the behavior if the intrinsic height and width of a visual media object differ from the values specified by the height and width values of the region. It takes the same values as the fit attribute on the region element, and has the same semantics with the exception that the declared fit only applies to the sub-region declared by the media element using it, and not to other elements which may use the same parent region.

**Region z-index override**

The z-index attribute is used on an element in conjunction with the sub-region positioning attributes to control the display of the element on the rendering surface. A value of the z-index attribute given on the element will set the z-index for the sub-region within the context of the parent region stacking order.

z-index
: This attribute specifies the stacking order behavior for the sub-region. It takes the same values as the z-index attribute on the region element, and has the same semantics with the exception that the declared z-index only applies to the sub-region declared by the media element, and not to other elements which may use the same parent region.

# 6. The SMIL 2.0 Linking Modules

***Editors***
Lloyd Rutledge (Lloyd.Rutledge@cwi.nl), (CWI)
Aaron Cohen (aaron.m.cohen@intel.com), (Intel)

## 6.1 Introduction

The SMIL 2.0 Linking Modules define the SMIL 2.0 document attributes and elements for navigational hyperlinking. These are navigations through the SMIL presentation that can be triggered by user interaction or other triggering events, such as temporal events. SMIL 2.0 provides only for in-line link elements. Links are limited to uni-directional single-headed links (i.e. all links have exactly one source and one destination resource).

The SMIL 2.0 Linking Modules are named LinkingAttributes, BasicLinking and ObjectLinking. The LinkingAttributes module includes a set of attributes

used to provide SMIL linking semantics to linking elements. The BasicLinking module includes the SMIL 2.0 linking elements themselves. The [illegible] explicitly includes the attributes from the LinkingAttributes module on its elements.

## 6.2 Relationship with Other XML Linking-related Formats

### 6.2.1 Relationship with XPointer

XPointer [XPTR] allows components of XML documents to be addressed in terms of their placement in the XML structure rather than on their unique identifiers. This allows referencing of any portion of an XML document without having to modify that document. Without XPointer, pointing within a document may require adding unique identifiers to it, or inserting specific elements into the document, such as a named anchor in HTML. XPointers are put within the fragment identifier part of a URI [URI] attribute value. The SMIL 2.0 specification allows but does not require that user agents be able to process XPointers in SMIL 2.0 URI attribute values.

### 6.2.2 Relationship with XLink

Where possible, SMIL linking constructs have the same element, attribute and value names as constructs from XLink [XLINK]. This makes it easier to learn to write linking in code in both formats: authors familiar with XLink can more quickly learn SMIL linking, and vice versa. It also makes it easier for SMIL code to be processed into and recognized as XLink code once XLink is released as a recommendation and when the appropriate transform mechanisms become available. However, the SMIL linking attributes are distinct from the XLink constructs and are part of a separate namespace. Using SMIL's modularization mechanism, these constructs are not in the XLink namespace but in the namespaces defined in the SMIL 2.0 specification.

### 6.2.3 Relationship with XML Base

SMIL profiles may use XML Base [XMLBase]. The SMIL 2.0 Language Profile, for example, includes support for XML Base. When XML Base is incorporated into a profile, XML Base declarations apply to the URI attribute values of SMIL used in that profile's documents. These attributes include the href attribute of the SMIL BasicLinking Module and the src attribute of the SMIL BasicMedia Module.

### 6.2.4 Relationship with XHTML

The elements names, attributes names and attribute values of SMIL linking constructs are, where possible, the same as constructs in XHTML [XHTML11] with corresponding linking behavior. This facilitates learning and writing in both languages and avoids confusion. It may also facilitate the processibility of both languages' linking constructs as XLink once the format is released. The linking constructs in SMIL, however, fall under the namespaces defined in SMIL 2.0, and not under any XHTML-related namespace.

## 6.3 Linking into SMIL 2.0 Documents

The SMIL 2.0 Linking Modules support name fragment identifiers and the '#' connector. The fragment part is an id value that identifies one of the elements within the referenced SMIL document. With this construct, SMIL 2.0 supports locators as currently used in HTML (that is, it uses locators of the form "http://www.example.org/some/path#anchor1"), with the difference that the values are of unique identifiers and not the values of "name" attributes. Of course, this type of link can only target elements that have an attribute of type ID.

Links using fragment identifiers enable authors to encode links to a SMIL 2.0 presentation at the start time of a particular element rather than at the beginning of its presentation. If a link containing a fragment part is followed, the presentation should start as if the user had fast-forwarded the presentation represented by the destination document to the effective begin of the element designated by the fragment. See the discussion of linking to timing constructs in the SMIL 2.0 Timing and Synchronization Modules for more information.

There are special semantics defined for following a link containing a fragment part into a document containing SMIL timing. These semantics are defined in the SMIL 2.0 Timing and Synchronization Modules.

### 6.3.1 Handling of Links in Embedded Documents

Due to its integrating nature, the presentation of a SMIL 2.0 document may involve other (non-SMIL) applications or plug-ins. For example, a SMIL 2.0 user agent may use an HTML plug-in to display an embedded HTML page. Vice versa, an HTML user agent may use a SMIL plug-in to display a SMIL 2.0 document embedded in an HTML page. Note that this is only one of the supported methods of integrating SMIL 2.0 and HTML. Another alternative is to use the merged language approach. See the SMIL 2.0 Modules for further details.

In embedded presentations, links may be defined by documents at different levels and conflicts may arise. In this case, the link defined by the containing document should take precedence over the link defined by the embedded object. Note that since this might require communication between the user agent and the plug-in, SMIL 2.0 implementations may choose not to comply with this recommendation.

If a link is defined in an embedded SMIL 2.0 document, traversal of the link affects only the embedded SMIL 2.0 document.

If a link is defined in a non-SMIL document which is embedded in a SMIL 2.0 document, link traversal can only affect the presentation of the embedded document and not the presentation of the containing SMIL 2.0 document. This restriction may be relaxed in future versions of SMIL.

### 6.3.2 Error Handling

When a link into a SMIL 2.0 document contains an un-resolvable fragment identifier ("dangling link") because it identifies an element that is not actually part of the document, SMIL 2.0 software should ignore the fragment identifier, and start playback from the beginning of the document.

When a link into a SMIL 2.0 document contains a fragment identifier which identifies an element that is the content of a switch element, SMIL 2.0 software should interpret this link as going to the outermost ancestor switch element instead. In other words, the link should be considered as accessing the switch ancestor element that is not itself contained within a switch.

## 6.4 SMIL 2.0 LinkingAttributes Module

The SMIL 2.0 LinkingAttribues module defines several attributes that a language profile can include on linking elements to add SMIL linking semantics to those elements. The elements in the BasicLinking Module explicitly include these attributes. These attributes can be applied to linking elements from other namespaces if allowed by the language profile.

accessKey

sourceLevel
This attribute sets the relative audio volume of media objects in the presentation containing the link when the link is followed. This attribute takes non-negative percentage values. The units of the sourceLevel attribute are consistent with those described for the soundLevel attribute of the SMIL 2.0 Layout Modules, and the audio levels are modified in the same manner. The application of sourceLevel volume control is cumulative with any media specific volume control (such as the soundLevel attribute) on the presentation containing the link. When the display of the destination resource completes, the effect of soundLevel attribute on the originating presentation should be removed. The default value is '100%'.

destinationLevel
This attribute sets the relative audio volume of media objects in the remote resource when the link is followed. This attribute can take non-negative percentage values. The destinationLevel attribute is applied to the natural or intrinsic audio volume of the destination media, and therefore is relative to the volume that the media would be played without application of the destinationLevel attribute. The units of the destinationLevel attribute are consistent with those described for the soundLevel attribute of the SMIL 2.0 Layout Modules, and the audio levels are modified in the same manner. The application of destinationLevel volume control is cumulative with any media specific volume control (such as the soundLevel attribute) specified by the remote resource. The default value is '100%'.

sourcePlaystate
This attribute controls the temporal behavior of the presentation containing the link when the link is traversed. It can have the following values:

- `play`: When the link is traversed, the presentation containing the link continues playing.
- `pause`: When the link is traversed, the presentation containing the link pauses. When the display of the destination resource completes, the originating presentation should resume playing.
- `stop`: When the link is traversed, the presentation containing the link stops. That is, it is reset to the beginning of the presentation. The termination of the destination resource will not cause the originating presentation to continue or restart.

The value of the show attribute can determine the default value of or override the assignment of the sourcePlaystate attribute. In general, the show attribute takes precedence over the sourcePlaystate attribute. The rules for the impact of show on sourcePlaystate are:

- If the show attribute is assigned the value `new`, then the default for the sourcePlaystate attribute is `play`.
- If the show attribute is assigned the value `replace` or the deprecated value `pause`, then the presentation behaves as if the sourcePlaystate attribute had been set to `pause`. Any assignment of the sourcePlaystate attribute is ignored.

Note that the definition of what constitutes a resource completing needs to be defined in the language profile, or may be implementation dependent. Typical definitions would be when the user closes the display window, or when a continuous media object ends.

destinationPlaystate
This attribute controls the temporal behavior of the external resource (typically identified by the href attribute) when the link is followed. It only applies when this resource is a continuous media object. It can have the following values:

- `play`: When the link is traversed, the destination of the link plays.
- `pause`: When the link is traversed, the destination of the link is displayed in a paused state at the point depicted by the value of the href attribute.

The default value is `play`.

show
This attribute specifies how to handle the current state of the presentation at the time in which the link is activated. The following values are allowed:

- `new`: The presentation of the destination resource starts in a new context, not affecting the source resource. If both the presentation containing the link and the remote resource contain audio media, both are played in parallel.
- `pause`: This value is deprecated in favor of setting the show attribute to `new` and the sourcePlaystate attribute to `pause`.
- `replace`: The current presentation is paused at its current state and is replaced by the destination resource. If the player offers a history mechanism, the source presentation resumes from the state in which it was paused when the user returns to it. The default value of sourcePlaystate is `pause` when the show attribute has the value `replace`. If the link destination is within the same document in which this link element lies, then any assignment of this element's sourcePlaystate attribute is ignored, and the link is processed as if the value of sourcePlaystate was `stop`. For more discussion regarding hyperlinking within the current document, see the "Hyperlinks and Timing" section of the SMIL 2.0 Timing and Synchronization Modules.

The default value of show is `replace`.

external
This attribute defines whether the link destination should be opened by the current application or some external application. A value of `true` will open the link in an external application defined on the system to handle this media type. A value of `false` will open the destination in the current application, however, if the current application does not support the media type of the referenced media, then it should attempt to render the media using an external application. Note that the means of associating media types with external applications is system dependent and not defined here. The default value of external is `false`.

actuate
The actuate attribute determines whether or not the link is triggered by some event or automatically traversed when its time span is active. Its default value is `onRequest`, which means something must trigger the link traversal, typically user interaction. A value of `onLoad` can also be assigned. This value indicates that the link is automatically traversed when the linking element becomes active. For linking elements containing SMIL timing, this is when the active duration of the linking element effectively begins, in other words, when the element's *beginEvent* event is fired. This means that a SMIL link can be encoded to be triggered by any event that can trigger the begin of a timed element. See the SMIL 2.0 Timing and Synchronization Modules for more details.

Each of the following attributes has the same syntax as the attributes of the same name in HTML [HTML4] and, where applicable, the same semantics:

alt
This attribute is defined for SMIL 2.0 in the SMIL 2.0 Media Object Modules. The recommendations given there for the alt attribute on media object elements apply to its use in linking elements as well.

accesskey
This attribute assigns a keyboard key whose activation by the user in turn activates this link. It has the same meaning as the attribute of the same name in HTML [HTML4]. Keystroke-activated links in embedded presentations stay effective when embedded -- that is, if the user hits that key during the SMIL presentation, that navigation occurs within the embedded media. The rules for disambiguating links in multiple objects are:

- Keystroke links defined in SMIL dominate over those defined in embedded media.
- When simultaneously active SMIL linking elements have the same "accesskey", then priority goes to the one activated earliest, then to

the one defined first on the SMIL code.
- In profiles in which the SMIL 2.0 Layout Modules are used, link in media objects placed in the foremost region dominate, as defined by "stacking level" in the SMIL 2.0 Layout Modules.
- Media objects for which no region is assigned are assumed to be more forward on the stacking level than all media objects assigned regions. This allows for the possibility of audio objects that have keystroke input, such as hyperlinked talking books for the sight-impaired.

tabindex
: This attribute provides the same functionality as the tabindex attribute in HTML [HTML4]. It specifies the position of the element in the tabbing order for the current document. The tabbing order defines the order in which elements will receive focus when navigated by the user via the keyboard. At any particular point in time, only elements with an active timeline are taken into account for the tabbing order. Inactive elements should be ignored for the tabbing order.
When a media object element has a tabindex attribute, then its ordered tab index is inserted in the SMIL tab index at the location specified by the media object's tabindex attribute value. This assumes the media object itself has tab indices, such as embedded HTML with tabindex attributes. This enables all link starting points in a SMIL presentation to have a place on the ordered list to be tab-keyed through, including those in embedded presentations.

target
: This attribute defines either the existing display environment in which the link should be opened (e.g., a SMIL region, an HTML frame or another named window), or triggers the creation of a new display environment with the given name. Its value is the identifier of the display environment. If no currently active display environment has this identifier, a new display environment is opened and assigned the identifier of the target. When a presentation uses different types of display environments (e.g. SMIL regions and HTML frames), the namespace for identifiers is shared between these different types of display environments. For example, one cannot use a target attribute with the value "foo" twice in a document, and have it point once to an HTML frame, and then to a SMIL region. If the element has both a show attribute and a target attribute, the show attribute is ignored.

**Examples**

These examples are encoded in the SMIL 2.0 Language Profile.

*Example 1*

This examples shows the use of the target and accesskey attributes. The upper half of the display shows an image. If the user clicks on the image, a SMIL presentation is played in the lower half of the display. The same thing happens if the user hits the 'a' key.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
 <head>
  <layout>
   <region id="source"      height="50%"/>
   <region id="destination" top   ="50%"/>
  </layout>
 </head>
 <body>
  <a href="embeddedSMIL.smil" target="destination" accesskey="a">
   <img region="source" src="source.jpg" dur="indefinite"/>
  </a>
 </body>
</smil>
```

*Example 2*

This example shows the use of the tabindex attribute on media object elements. The HTML file "caption1.html" has 3 links, so the first 3 tabs focus on those links in turn. The file caption2.html has 4 links, so tabs 4-7 focus on them in turn. Tabs 8 and 9 focus the two links inside v1.mpg. Tab 10 focuses on the whole presentation of graph.imf. If any of the first 9 tabbed foci is activated, then a link inside one of the embedded presentations caption1.html, caption2.rtx or v1.mpg is triggered, affecting only that presentation. If the 10th tabbed focus is activated, then the SMIL presentation itself is affected, loading http://www.example.org/presentation into the same presentation space.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
 ...
 <seq>
  <video src="http://www.example.org/graph.imf"/>
  <par>
      <a tabindex="4" href="http://www.example.org/presentation">
          <video src="http://www.example.org/graph.imf" ... />
      </a>
      <video tabindex="3" src="http://www.example.org/v1.mpg" ... />
      <text tabindex="1" src="http://www.example.org/caption1.html" ... />
      <text tabindex="2" src="http://www.example.org/caption2.html" ... />
  </par>
 </seq>
```

## 6.5 SMIL 2.0 BasicLinking Module

The link elements allows the description of navigational links between objects. SMIL 2.0 linking provides only uni-directional, single-headed, in-line link elements.

### 6.5.1 The a Element

The functionality of the a element is very similar to the functionality of the a element in HTML [HTML4]. For synchronization purposes, the a element is transparent. That is, it does not influence the synchronization of its child elements. a elements may not be nested. An a element must have an href attribute.

An a element can specify several triggers for its traversal simultaneously. For example, the element's content visual media can be selected by the user or the key specified by the accesskey attribute can be typed to trigger a traversal. In cases where multiple triggers are specified, any of them can activate the link's traversal. That is, a logical OR is applied to the list of triggering conditions to determine if traversal occurs.

Traversal occurs if one of the conditions for traversal is met during the time that the a element is active. An a element is sensitive if the media or elements that it contains are active or frozen. See the SMIL 2.0 Timing and Synchronization Modules for further details. For timing purposes an a element is considered to be discrete media, that is, the intrinsic duration is 0. Note that an a element is not a time container and does not constrain the timing of its child elements.

**Attributes**

href
: This attribute has the same syntax and semantics as the href attribute of HTML [HTML4]. It contains the URI of the link's destination. The href attribute is required for a elements.

href attribute is required for a elements.

The a element also includes the attributes defined in the SMIL 2.0 LinkingAttributes Module:

- sourceLevel
- destinationLevel
- sourcePlaystate
- destinationPlaystate
- alt
- show
- accesskey
- tabindex
- target
- external
- actuate

**Element Content**

The content of the a element must be defined by the language profile. In general, it is expected that a elements may contain the media and timing elements present in the language profile as children.

**Other Integration Requirements**

Language profiles that apply SMIL 2.0 timing to the a element must specify the default and allowed values of the fill attribute on the a element. Languages applying SMIL 2.0 timing to the a element wishing to remain compatible with SMIL 1.0, such as the SMIL 2.0 language profile, must default the value of the fill attribute on the a element to `auto`, and should consider fixing the value to `auto`. In all other cases, for compatibility, it is recommended to use a default value of `auto`.

If not otherwise specified by the profile, the value of the fill attribute on the a element is fixed to `auto`.

**Examples**

These examples are encoded in the SMIL 2.0 Language Profile.

*Example 1*

The link starts up the new presentation replacing the presentation that was playing.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/somewhereelse.smi">
     <video src="rtsp://www.example.org/graph.imf" region="1_window"/>
  </a>
```

*Example 2*

The link starts up the new presentation in addition to the presentation that was playing.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/somewhereelse.smi" show="new">
     <video src="rtsp://www.example.org/graph.imf" region="1_window"/>
  </a>
```

This could allow a SMIL 2.0 player to spawn off an HTML user agent:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/somewebpage.html" show="new">
     <video src="rtsp://www.example.org/graph.imf" region="1_window"/>
  </a>
```

*Example 3*

The link starts up the new presentation and pauses the presentation that was playing.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/somewhereelse.smi" show="new" sourcePlaystate="pause">
     <video src="rtsp://www.example.org/graph.imf" region="1_window"/>
  </a>
```

*Example 4*

The following example contains a link from an element in one presentation A to the middle of another presentation B. This would play presentation B starting from the effective begin of the element with id "next".

```
Presentation A:

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/presentationB#next">
    <video src="rtsp://www.example.org/graph.imf"/>
  </a>


Presentation B (http://www.example.org/presentation):

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <seq>
    <video src="rtsp://www.example.org/graph.imf"/>
    <par>
      <video src="rtsp://www.example.org/timbl.rm" region="1_window"/>
      <video id="next" src="rtsp://www.example.org/v1.rm" region="r_window"/>
            ^^^^^^^^
      <text src="rtsp://www.example.org/caption1.html" region="1_2_title"/>
    </par>
  </seq>
```

## 6.5.2 The area Element

The functionality of the a element is restricted in that it only allows associating a link with a complete media object. The HTML area element [HTML4] has demonstrated that it is useful to associate links with spatial portions of an object's visual display.

The semantics of the area element in SMIL 2.0 is the same as it is for HTML in that it can specify that a spatial portion of a visual object can be selected to trigger the appearance of the link's destination. The coords attribute specifies this spatial portion. In contrast, if an a element is applied to a visual object, then it specifies that any visual portion of that object can be selected to trigger the link traversal.

The area element also extends the syntax and semantics of the HTML area element by providing for linking from non-spatial portions of the media object's display. When used in profiles that include SMIL 2.0 Timing and Synchronization Modules, the area element allows breaking up an object into temporal subparts, using attributes such as the begin and end attributes. The values of the begin and end attributes are relative to the beginning of the containing media object. The area element can allow make a subpart of the media object the destination of a link, using these timing attributes and the id attribute.

The anchor element of SMIL 1.0 [SMIL10] is deprecated in favor of area. For purposes of this specification of SMIL 2.0, the anchor element should be treated as a synonym for area

**Attributes**

The area element can have the attributes listed below, with the same syntax as in HTML [HTML4] and, where applicable, the same semantics:

href
: Defined in the BasicLinking module.

alt
: Defined in the LinkingAttributes module.

tabindex
: Defined in the LinkingAttributes module.

accesskey
: Defined in the LinkingAttributes module.

target
: [illegible]

nohref
: When set, this Boolean attribute specifies that the region has no associated link, even if other area elements for the media object define links for it.

shape
: This attribute specifies the shape of an anchor on the screen, and is used with the coords attribute. The shape attribute of SMIL has the same behavior as in HTML [HTML4]. The object that the shape attribute is applied to is the media of the containing element after the media has been scaled for presentation but before it has been clipped.

coords
: Along with the shape attribute, this attribute specifies the position and shape of the anchor on the screen. The number and order of values depends on the shape being defined. Where SMIL and HTML share visual display behavior, the coords attribute of SMIL has the same behavior as in HTML [HTML4]. How the coords attribute of SMIL applies to SMIL visual display behavior is as follows:

  1. The upper-left corner origin used by the coords attribute is in the image display space, not the region it is displayed in. One example of when the image and region upper-left corner differ is when left and top attributes are used in the media object elements and applied to the region.
  2. As in HTML, pixel values of the coords attribute refer to the pixels of the display space - that is, typically, of the user's screen. These can differ from the pixel values of the source image. One example of when they can differ in SMIL is when the fit attribute of the region element used is not "hidden".
  3. In SMIL, it is possible for a portion of the image to be not visible. For example, this can happen when the fit attribute of the region element is hidden, and the image is bigger in pixels than the region. In such cases, the rules for placing active areas on the image apply to the screen space the image would take if no cropping occurred. This is particular important to note for percentage values: these do not apply to the cropped image but to the space the image would occupy if it weren't cropped. Areas that, with this rule, are placed beyond display boundaries are not active.

  If multiple area element children of the same media object element define simultaneously active overlapping areas with their coords attributes, then, as in HTML [HTML4], the area element that is encoded earliest in the document takes precedence.

The following attributes of the area element are unique to SMIL and not found in HTML. They are defined above in the section on LinkingAttributes module attributes:

- sourcePlaystate
- destinationPlaystate
- show
- [illegible]
- destinationLevel
- external
- actuate

**Element Content**

The area element is empty.

**Examples**

These examples are encoded in the SMIL 2.0 Language Profile.

*1) Decomposing a video into temporal segments*

In the following example, the temporal structure of an interview in a newscast (camera shot on interviewer asking a question followed by shot on interviewed person answering) is exposed by fragmentation:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <video src="video" title="Interview" >
        <area id="firstQ" begin="0s" dur="20s" title="first question" />
        <area id="firstA" begin="firstQ.end" dur="50s" title="first answer" />
    </video>
  </body>
</smil>
```

*2) Associating links with spatial segments* In the following example, the screen space taken up by a video clip is split into two sections. A different link is associated with each of these sections.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <video src="video" title="Interview" >
      <area shape="rect" coords="5,5,50,50"
            title="Journalist" href="http://www.example.org/>
      <area shape="rect" coords="60,5,100,50"
            title="Subject" href="http://www.example.org/>
    </video>
  </body>
</smil>
```

*3) Associating links with temporal segments*

In the following example, the duration of a video clip is split into two sub-intervals. A different link is associated with each of these sub-intervals.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <video src="video" title="Interview" >
        <area begin="0s" dur="20s" title="first question"
              href="http://www.example.org/>
        <area begin="20s" dur="50s" title="first answer"
              href="http://www.example.org/>
    </video>
  </body>
</smil>
```

*4) Associating links with spatial subparts*

In the following example, two areas are assigned in the screen space taken up by a video clip. A different link is associated with each of these areas.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <video src="http://www.example.org/CoolStuff">
    <area href="http://www.example.org/AudioVideo" coords="0%,0%,50%,50%"/>
    <area href="http://www.example.org/Style"      coords="50%,50%,100%,100%"/>
  </video>
```

*5) Associating links with temporal subparts*

In the following example, the duration of a video clip is split into two subintervals. A different link is associated with each of these subintervals.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <video src="http://www.example.org/CoolStuff">
    <area href="http://www.example.org/AudioVideo" begin="0s" end="5s"/>
    <area href="http://www.example.org/Style"      begin="5s" end="10s"/>
  </video>
```

*6) Jumping to a subpart of an object*

The following example contains a link from an element in one presentation A to the middle of a video object contained in another presentation B. This would play presentation B starting from second 5 in the video. That is, the presentation would start as if the user had fast-forwarded the whole presentation to the point at which the designated fragment in the "CoolStuff" video begins.

```
Presentation A:

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/mm/presentationB#tim">
    <video id="graph" src="rtsp://www.example.org/graph.imf" region="1_window"/>
  </a>


Presentation B:

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <video src="http://www.example.org/CoolStuff">
    <area id="joe" begin="0s" end="5s"/>
    <area id="tim" begin="5s" end="10s"/>
  </video>
```

*7) Combining different uses of links*

The following example shows how the different uses of associated links can be used in combination.

```
Presentation A:

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <a href="http://www.example.org/mm/presentationB#tim">
    <video id="graph" src="rtsp://www.example.org/graph.imf" region="1_window"/>
  </a>


Presentation B:

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <video src="http://www.example.org/CoolStuff">
    <area id="joe" begin="0s" end="5s" coords="0%,0%,50%,50%"
          href="http://www.example.org/"/>
    <area id="tim" begin="5s" end="10s" coords="0%,0%,50%,50%"
          href="http://www.example.org/Tim"/>
  </video>
```

*8) The coords attribute and re-sized images*

The following example shows the image file "example.jpg", which has the dimensions of 100x100 pixels. The active area for "example1.smil" is the entire display space, which is the cropped upper-left quarter of the original image. The active area for "example2.smil" cannot be triggered because the image area corresponding to it was cropped.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <layout>
      <region id=region right="50" bottom="50"/>
    </layout>
```

```
  </head>
  <body>
    <img src="example.jpg" region="region">
      <area shape="rect" coords="0%,0%,50%,50%" href="example1.smil"/>
      <area shape="rect" coords="50%,50%,100%,100%" href="example2.smil"/>
    </img>
  </body>
</smil>
```

## 6.6 SMIL 2.0 ObjectLinking Module

The contents of this section represent capabilities that can be optionally included in the document profile. These features may or may not be included in a language profile, but they should not be optional features within a profile. This module requires support of the BasicLinking Module.

### 6.6.1 The fragment Attribute

A profile may choose to include the fragment attribute as part of the area element. It provides for a host document to externally include a link in a contained media object that will be processed at the level of the host document.

fragment

This attribute refers to a portion of the embedded media object that is to act as the starting point of this link in the SMIL presentation. If the user clicks on, or otherwise activates, this portion of the embedded media display, the SMIL user agent recognizes this as the current link being activated. This overrides any linking that may happen within the embedded display of the media object.

The value of the fragment attribute must be recognizable by the process managing the media object as an activate-able portion of the object. If the referenced media object is an HTML file, then the value of the fragment attribute is a named anchor within the HTML file. If the referenced media object is an XML file, then the value of the fragment attribute is a fragment identifier (the part that comes after a '#' in a URI [URI]).

Take for example the following SMIL code. It establishes a portion of the display as a formatted text menu. Clicking on an item in this menu triggers a link to elsewhere within the presentation.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
  <ref src="menu.html" region="menubar">
    <area fragment="menuitem1" href="#selection1"/>
  </ref>
```

In the rendered HTML display, there is a portion of displayed text that is marked-up as an area with the name "menuitem1". If the user clicks on this during the SMIL presentation, a SMIL-activated link is triggered, navigating to the portion of the SMIL document with the ID "selection1". If the HTML area named "menuitem1" has an href attribute itself, then this hyperlink is overridden - only the SMIL hyperlink is processed. HTML area with href attributes and no associated SMIL fragment attributes are not overridden. This HTML area activates links within the embedded HTML presentation when clicked upon.

Use of the fragment attribute can override linking in the embedded media. If the attribute refers to a portion of the embedded media that is a link within that media, activating that link will trigger navigation in the SMIL presentation only, and not in the embedded presentation. For example, suppose a fragment attribute refers to a named anchor in an embedded HTML document. This named area has an href attribute, making it the starting point of a potential navigation within the HTML presentation itself. When embedded in the SMIL presentation, activation of this part of the HTML display triggers the SMIL link and not the HTML link. Links in embedded media that are not overridden in this manner, on the other hand, continue to trigger navigation within the embedded display when activated. All functionality defined for the SMIL link will override any equivalent functionality defined for the link in the embedded media. With the above example, the alt attribute of the SMIL area element would override the alt tag of the embedded HTML anchor.

The referencing performed by the fragment attribute only applies to one level of depth of embedded media. It only applies to directly embedded media; it does not apply to media embedded in turn within media embedded in a SMIL presentation. For example, consider a SMIL presentation that embeds a second SMIL presentation within it. The media object element of the first that embeds the second has within it an area element with a fragment attribute. The value of this attribute applies only to the embedded SMIL document itself. It does not apply to any media embedded within this second SMIL presentation.

**Examples**

These examples are encoded in the SMIL 2.0 Language Profile.

*Associating links with syntactic subparts*

Below is an example with an integrated HTML file that displays a menu of

```
link one
link two
```

The user can click on one of the menu items, and the matching HTML file is displayed. That is, if user clicks on "link one", the "Link1.html" file is displayed in the "LinkText" region. Note that the links defined inside the embedded HTML presentation, those to "overridden1.html" and "overridden2.html" are not active when embedded here because they are overridden by the fragments.

The "menu.html" file contains the code:

```
<html>
...
  <A NAME="link1" HREF="overridden1.html">link one</A><BR/>
  <A NAME="link2" HREF="overridden2.html">link two</A>
```

The SMIL 2.0 file is:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <layout>
      <region id="HTML"     width="100" height="100"/>
      <region id="LinkText" width="100" top   ="100"/>
    </layout>
  </head>
  <body>
    <par>
      <text region="HTML" src="menu.html" dur="indefinite">
        <area fragment="link1" href="#LinkOne"/>
        <area fragment="link2" href="#LinkTwo"/>
      </text>
      <excl dur="indefinite" >
        <text id="LinkOne" region="LinkText" src="Link1.html" dur="indefinite"/>
```

```
        <text id="linkTwo" region="linkText" src="link2.html" dur="indefinite"/>
      </excl>
    </par>
  </body>
</smil>
```

# 7. The SMIL 2.0 Media Object Modules


Editor
Rob Lanphier (robla@real.com), RealNetworks


## 7.1 Introduction

This section defines the SMIL media object modules, which are composed of a BasicMedia module and five modules with additional functionality that build on top of the BasicMedia module: the MediaClipping, MediaClipMarkers, MediaParam, MediaAccessibility, and MediaDescription modules. These modules contain elements and attributes used to describe media objects. Additionally, a BrushMedia element is provided which can be used as a media object. Since these elements and attributes are defined in a series of modules, designers of other markup languages can reuse the SMIL media module when they need to include media objects into their language.

Changes with respect to the media object elements in SMIL 1.0 provide additional functionality that was brought up as requirements of the Working Group, and those differences are explained in Appendix A and Appendix B.

## 7.2 Definitions

**Continuous Media**
: Audio file, video file or other media for which there is a measurable and well-understood duration. For example, a five second audio clip is continuous media, because it has a well-understood duration of five seconds. Opposite of "discrete media".

**Discrete Media**
: Image file, text file or other media which has no obvious duration. For example, a JPEG image is generally considered discrete media, because there's nothing in the file indicating how long the JPEG should be displayed. Opposite of "continuous media".

**Intrinsic Duration**
: The duration of a referenced item without any explicit timing markup. Continuous media has an intrinsic duration defined by the media, and discrete media has no intrinsic duration. (In SMIL, discrete media is assigned an intrinsic duration of zero).

## 7.3 SMIL BasicMedia Module

This module defines the baseline functionality of a SMIL player. It is very close in functionality to the media object specification in SMIL 1.0.

### 7.3.1 Media Object Elements - ref, animation, audio, img, text, textstream and video

The media object elements allow the inclusion of media objects into a SMIL presentation. Media objects are included by reference (using a URI). The following media elements are defined in this section:

ref
: Generic media reference

animation
: Animated vector graphics or other animated format

audio
: Audio clip

img
: Still image, such as PNG or JPEG

text
: Text reference

textstream
: Streaming text

video
: Video clip

All of these media elements are semantically identical. When playing back a media object, the player must not derive the exact type of the media object from the name of the media object element. Instead, it must rely solely on other sources about the type, such as type information contained in the type attribute, or the type information communicated by a server or the operating system.

Authors, however, should make sure that the group into which of the media object falls (animation, audio, img, video, text or textstream) is reflected in the element name. This is in order to increase the readability of the SMIL document. When in doubt about the group of a media object, authors should use the generic "ref" element.

The animation element defined here should not be confused with the elements defined in the SMIL 2.0 Animation Module. The animation element defined in this module is used to include an animation (such as a vector graphics animation) by reference. This is in contrast to the elements defined in the Animation module, which provide an in-line syntax for the animation of attributes and properties of other elements.

Anchors and links can be attached to visual media objects, i.e. media objects rendered on a visual abstract rendering surface.

*Attributes Definitions*

implementing the SMIL BasicMedia module, media object elements can have the following attributes:

src
: The value of the src attribute is the [URI] of the media element, used for locating and fetching the associated media. Note that this attribute is not required. A media object with no src attribute has an intrinsic duration of zero, and participates in timing just as any other media element. No media will be fetched by the SMIL implementation for a media element without a src attribute.

type
: Content type of the media object referenced by the src attribute. The usage of this attribute depends on the protocol of the src attribute.

**RTSP [RTSP]**
The type attribute is used for purposes of content selection and when the type of the referenced media is not otherwise available. It may be overridden by the contents of the RTSP DESCRIBE response or by the static RTP payload number.

**HTTP [HTTP], FTP [FTP], and local file playback [URL] (mainly URL spec, but describes the "file:" URL scheme])**
The type attribute value takes precedence over other possible sources of the media type (for instance, the "Content-type" field in an HTTP exchange, or the file extension).

When the content represented by a URL is available in many data formats, implementations MAY use the type value to influence which of the multiple formats is used. For instance, on a server implementing HTTP content negotiation, the client may use the type attribute to order the preferences in the negotiation.

For protocols not enumerated in this specification, implementations should use the following rules: When the media is encapsulated in a media file and delivered intact to the SMIL user agent via a protocol designed for delivery as a complete file, the type attribute value should take precedence over other possible sources of the media type. For protocols which deliver the media in a media-aware fashion, such as those delivering media in a manner using or dependent upon the specific type of media, the application of the type attribute is not defined by this specification.

*Element Content*

Languages utilizing the SMIL BasicMedia module must define the complete set of elements which may act as children of media object elements. There are currently no required children of a media object defined in the BasicMedia Module, but languages utilizing the BasicMedia module may impose requirements beyond this specification.

### 7.3.2 Integration Requirements

If the including profile supports the XMLBase functionality [XMLBase] , the values of the src and longdesc attributes on the media object elements must be interpreted in the context of the relevant XMLBase URI prefix.

## 7.4 SMIL MediaParam Module

This section defines the elements and attributes that make up the SMIL MediaParam Module definition. Languages implementing elements and attributes found in the MediaParam module must implement all elements and attributes defined below, as well as BasicMedia.

### 7.4.1 Media object initialization: the **param** element

param elements specify a set of values that may be required by a media object at run-time. Any number of param elements may appear in the content of a media object element, in any order, but must be placed at the start of the content of the enclosing media object element.

The syntax of names and values is assumed to be understood by the object's implementation. This document does not specify how user agents should retrieve name/value pairs nor how they should interpret parameter names that appear twice.

*Attribute definitions*

name
(CDATA) This attribute defines the name of a run-time parameter, assumed to be known by the inserted object. Whether the property name is case-sensitive depends on the specific object implementation.

value
(CDATA) This attribute specifies the value of a run-time parameter specified by name. Property values have no meaning to SMIL; their meaning is determined by the object in question.

valuetype
["data"|"ref"|"object"] This attribute specifies the type of the value attribute. Possible values:

- data: This is default value for the attribute. It means that the value specified by value will be evaluated and passed to the object's implementation as a string.
- ref: The value specified by value is a URI [URI] that designates a resource where run-time values are stored. This allows support tools to identify URIs given as parameters. The URI must be passed to the object **as is**, i.e., unresolved.
- object: The value specified by value is an identifier that refers to a media object declaration in the same document. The identifier must be the value of the id attribute set for the declared media object element.

type
This attribute specifies the content type of the resource designated by the value attribute **only** in the case where valuetype is set to "ref". This attribute thus specifies for the user agent, the type of values that will be found at the URI designated by value. See 6.7 Content Type in [HTML4] for more information.

*Example*

To illustrate the use of param: suppose that we have a facial animation plug-in that is able to accept different moods and accessories associated with characters. These could be defined in the following way:

```
<ref src="http://www.example.com/herbert.face">
  <param name="mood" value="surly" valuetype="data"/>
  <param name="accessories" value="baseball-cap,nose-ring" valuetype="data"/>
</ref>
```

### 7.4.2 Element Attributes for All Media Objects

In addition to the element attributes defined in BasicMedia, media object elements can have the attributes and attribute extensions defined below. The inclusion or exclusion of these elements is left as an option in the language profile.

erase
Controls the behavior of the media object after the effects of any timing are complete. For example, when SMIL Timing is applied to a media element, erase controls the display of the media when the active duration of the element and when the freeze period defined by the fill attribute is complete (see SMIL Timing and Synchronization). Possible values for erase are `never` and `whenDone`.

erase="`whenDone`" is the default value. When this is specified (or implied) the media removal occurs at the end of any applied timing.

erase="`never`" is defined to keep the last state of the media displayed until the display area is reused (or if the display area is already being

used by another media object). Any profile that integrates this element must define what is meant by "display area" and further define the interaction. Intrinsic hyperlinks (e.g., Flash, HTML) and explicit hyperlinks (e.g., area, a) stay active as long as the hyperlink is displayed. If timing is reapplied to an element, the effect of the erase=`never` is cleared. For example, when an element is restarted according to the SMIL Timing and Synchronization module, the element is cleared immediately before it restarts.

Example:

```
<par>
  <seq>
        <par>
            <img src="image1.jpg" region="foo1" fill="freeze" erase="never" .../>
            <audio src="audio1.au"/>
        </par>
        <par>
            <img src="image2.jpg" region="foo2" fill="freeze" erase="never" .../>
            <audio src="audio2.au"/>
        </par>
         ...
        <par>
            <img src="imageN.jpg" region="fooN" fill="freeze" erase="never" .../>
            <audio src="audioN.au"/>
        </par>
  </seq>
</par>
```

In this example, each image is successively displayed and remains displayed until the end of the presentation.

mediaRepeat

Used to strip the intrinsic repeat value of the underlying media object. The interpretation of this attribute is specific to the media type of the media object, and is only applicable to those media types for which there is a definition of a repeat value found in the media type format specification. Media type viewers used in SMIL implementations will need to expose an interface for controlling the repeat value of the media for this attribute to be applied. For all media types where there is an expectation of interoperability between SMIL implementations, there should be a formal specification of the exact repeat value to which the mediaRepeat attribute applies.

Values:

`strip`
: Strip the intrinsic repeat value of the media object.

`preserve` **(default)**
: Leave the intrinsic repeat value of the media object intact.

As an example of how this would be used, many animated GIFs intrinsically repeat indefinitely. The application of mediaRepeat= "`strip`" allows an author to remove the intrinsic repeat behavior of an animated GIF on a per-reference basis, causing the animation to display only once, regardless of the repeat value embedded in the GIF.

When mediaRepeat is used in conjunction with SMIL Timing Module attributes, this attribute is applied first, so that the repeat behavior can [illegible] repeatCount and repeatDur.

sensitivity

Used to provide author control over the sensitivity of media to user interface selection events, such as the SMIL 2.0 activateEvent, and hyperlink activation. If the media is sensitive at the event location, it captures the event, and will not pass the event through to underlying media objects. If not, it allows the event to be passed through to any media objects lower in the display hierarchy.

Values:

`opaque`
: The media is sensitive to user interface selection events over the entire area of the media. This is the default.

`transparent`
: The media is not sensitive to user interface selection events over the entire area of the media. Any user interface selection events will be "passed through" to any underlying media.

*`percentage-value`*
: The media sensitivity to user interface selection events is dependent upon the opacity of the media at the location of the event (the alpha channel value). If rendered media supports an alpha channel and the opacity of the media is less than the given percentage value at the event location, the behavior will be `transparent` as specified above. Otherwise the behavior will be as `opaque`. Valid values are non-negative CSS2 percentage values.

### 7.4.3 Integration Requirements

Any profile that integrates the erase attribute must define what is meant by "display area" and further define the interaction. See the definition of erase for more details.

The supported uses of the type and valuetype attributes on the param element must be specified by the integrating profile. If a profile does not specify this, the type and valuetype attributes will be ignored in that profile.

## 7.5 SMIL MediaClipping Module

This section defines the attributes that make up the SMIL MediaClipping Module definition. Languages implementing the attributes found in the MediaClipping module must implement the attributes defined below, as well as BasicMedia.

### 7.5.1 MediaClipping Attributes

clipBegin (clip-begin)

The clipBegin attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object. This offset is measured in normal media playback time from the beginning of the media.
Values in the clipBegin attribute have the following syntax:

```
Clip-value-MediaClipping ::= [ Metric "=" ] ( Clock-val | Smpte-val )
Metric                   ::= Smpte-type | "npt"
Smpte-type               ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val                ::= Hours ":" Minutes ":" Seconds
                             [ ":" Frames [ "." Subframes ]]
Hours                    ::= Digit+
                             /* see XML 1.0 for a definition of 'Digit' */
Minutes                  ::= Digit Digit; range from 00 to 59
Seconds                  ::= Digit Digit; range from 00 to 59
```

```
Frames          ::= Digit Digit; smpte range = 00-29, smpte-30-drop range = 00-29, smpte-25 range = 00-24
Subframes       ::= Digit Digit; smpte range = 00-01, smpte-30-drop range = 00-01, smpte-25 range = 00-01
```

*Note: additional BNF for level 1 extensions defined later*

The value of this attribute consists of a metric specifier, followed by a time value whose syntax and semantics depend on the metric specifier. The following formats are allowed:

**SMPTE Timestamp**
SMPTE time codes [SMPTE] can be used for frame-level access accuracy. The metric specifier can have the following values:

**smpte**
**smpte-30-drop**
These values indicate the use of the "SMPTE 30 drop" format (approximately 29.97 frames per second), as defined in the SMPTE specification (also referred to as "NTSC drop frame"). The "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute.
**smpte-25**
The "frames" field in the time specification can assume the values 0 through 24. This corresponds to the PAL standard as noted in [SMPTE]

The time value has the format hours:minutes:seconds:frames.subframes. If the subframe value is zero, it may be omitted. Subframes are measured in one-hundredths of a frame.
Examples:
```
clipBegin="smpte=10:12:33:20"
```

**Normal Play Time**
Normal Play Time expresses time in terms of SMIL clock values. The metric specifier is "npt", and the syntax of the time value is identical to the syntax of SMIL clock values.
Examples:
```
clipBegin="npt=123.45s"
clipBegin="npt=12:05:35.3"
```
**Marker**
Not defined in this module. See clipBegin Media Marker attribute extension in the MediaClipMarkers module.

If no metric specifier is given, then a default of "npt=" is presumed.

When used in conjunction with the timing attributes from the SMIL Timing Module, this attribute is applied before any SMIL Timing Module attributes.

clipBegin may also be expressed as clip-begin for compatibility with SMIL 1.0. Software supporting the SMIL 2.0 Language Profile must be able to handle both clipBegin and clip-begin, whereas software supporting only the SMIL MediaClipping module only needs to support clipBegin. If an element contains both a clipBegin and a clip-begin attribute, then clipBegin takes precedence over clip-begin.

Example:

```
<audio src="radio.wav" clip-begin="5s" clipBegin="10s" />
```

The clip begins at second 10 of the audio, and not at second 5, since the clip-begin attribute is ignored. A strict SMIL 1.0 implementation will start the clip at second 5 of the audio, since the clipBegin attribute will not be recognized by that implementation. See Changes to SMIL 1.0 Media Object Attributes for more discussion on this topic.

clipEnd (clip-end)
The clipEnd attribute specifies the end of a sub-clip of a continuous media object as offset from the start of the media object. This offset is measured in normal media playback time from the beginning of the media. It uses the same attribute value syntax as the clipBegin attribute. If the value of the clipEnd attribute exceeds the duration of the media object, the value is ignored, and the clip end is set equal to the effective end of the media object. clipEnd may also be expressed as clip-end for compatibility with SMIL 1.0. Software supporting the SMIL 2.0 Language Profile must be able to handle both clipEnd and clip-end, whereas software supporting only the SMIL media object module only needs to support clipEnd. If an element contains both a clipEnd and a clip-end attribute, then clipEnd takes precedence over clip-end. When used in conjunction with the timing attributes from the SMIL Timing Module, this attribute is applied before any SMIL Timing Module attributes.

See Changes to SMIL 1.0 Media Object Attributes for more discussion on this topic.

## 7.6 SMIL MediaClipMarkers Module

This section defines the attribute extensions that make up the SMIL MediaClipMarkers Module definition. Languages implementing elements and attributes found in the MediaClipMarkers module must implement all elements and attributes defined below, as well as BasicMedia and MediaClipping.

### 7.6.1 MediaClipMarkers Attribute Extensions

clipBegin Media Marker attribute extension
Used to define a clip using named time points in a media object, rather than using clock values or SMPTE values. The metric specifier is "marker", and the marker value is a URI (see [URI] ). The URI is relative to the src attribute, rather than to the document root or the XML base of the SMIL document.

```
Clip-value-MediaClipMarkers ::= Clip-value-MediaClipping |
                                "marker" "=" URI-reference
  /* "URI-reference" is defined in  [URI]  */
```

Example: Assume that a recorded radio transmission consists of a sequence of songs, which are separated by announcements by a disk jockey. The audio format supports marked time points, and the begin of each song or announcement with number X is marked as songX or djX respectively. To extract the first song using the "marker" metric, the following audio media element can be used:

```
<audio clipBegin="marker=#song1" clipEnd="marker=#dj1" />
```

clipEnd Media Marker attribute extension
clipEnd media markers use the same attribute value syntax as the **clipBegin media marker extension** media marker attribute extension. For

the complete description, see **clipBegin media marker extension**.

## 7.7 SMIL BrushMedia Module

This section defines the elements and attributes that make up the SMIL BrushMedia Module definition. Languages implementing elements and attributes found in the BrushMedia module must implement all elements and attributes defined below.

### 7.7.1 The **brush** element

The **brush** element is a lightweight media object element which allows an author to paint a solid color or other pattern in place of a media object. Thus, all attributes associated with media objects may also be applied to **brush**. Since all information about the media object is specified in the attributes of the element itself, the src attribute is ignored, and thus is not required.

*Attribute definitions*

color
: The use and definition of this attribute are identical to the "background-color" property in the CSS2 specification.

### 7.7.2 Integration Requirements

Profiles including the BrushMedia module must provide semantics for using a color attribute value of `inherit` on the **brush** element. Because `inherit` doesn't make sense in all contexts, a profile may choose to prohibit the use of this value. The value of `inherit` is prohibited on the color attribute of the **brush** element for profiles that do not otherwise define these semantics.

## 7.8 SMIL MediaAccessibility Module

This section defines the elements and attributes that make up the SMIL MediaAccessibility Module definition. Languages implementing elements and attributes found in the MediaAccessibility module must implement all elements and attributes defined below, as well as MediaDescription.

### 7.8.1 MediaAccessibility Attributes

*Attribute definitions*

alt
: For user agents that cannot display a particular media object, this attribute specifies alternate text. alt may be displayed in addition to the media, or instead of media when the user has configured the user agent to not display the given media type.

  It is strongly recommended that all media object elements have an "alt" attribute with a brief, meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

  The value of this attribute is a CDATA text string.

longdesc
: This attribute specifies a link ( [URI] ) to a long description of a media object. This description should supplement the short description provided using the alt attribute or the abstract attribute. When the media object has associated hyperlinked content, this attribute should provide information about the hyperlinked content.

readIndex
: This attribute specifies the position of the current element in the order in which longdesc, title and alt text are read aloud by assistive devices (such as screen readers) for the current document. User agents should ignore leading zeros. The default value is 0.

  Elements that contain alt, title or longdesc attributes are read by the assistive technology according to the following rules:

  - Those elements that assign a positive value to the readindex attribute are read out first. Navigation proceeds from the element with the lowest readindex value to the element with the highest value. Values need not be sequential nor must they begin with any particular value. Elements that have identical readindex values should be read out in the order they appear in the character stream of the document.
  - Those elements that assign it a value of "0" are read out in the order they appear in the character stream of the document.
  - Elements in a switch statement that have test-attributes which evaluate to "false" are not read out.

  *Example*

```
<par>
  <video id="carvideo" src="car.rm" region="videoregion" title="Car video"
         alt="Illustration of relativistic time dilation and length
              contraction."
         longdesc="carvideodesc.html" readIndex="3"/>
  <audio id="caraudio" src="caraudio.rm" region="videoregion"
         title="Car presentation voiceover" begin="bar.begin"/>
  <animation id="cardiagram" src="car.svg" region="animregion"
         title="Diagram of the car" readIndex="2"/>
  <img id="scvad" src="scv.png" region="videoregion"
         title="Advertisement for Sugar Coated Vegetables"
         readIndex="1"/>
```

  In this example, an assistive device that is presenting titles should present the "scvad" element title first (having the lowest readIndex value of "1"), followed by the "cardiagram" title, followed by the "carvideo" element title, and finally present the "caraudio" element title (having an implicit readIndex value of "0").

## 7.9 SMIL MediaDescription Module

This section defines the elements and attributes that make up the SMIL MediaDescription Module definition. Languages implementing elements and attributes found in the MediaDescription module must implement all elements and attributes defined below.

### 7.9.1 MediaDescription Attributes

*Attribute definitions*

abstract
A brief description of the content contained in the element. Unlike alt, this attribute is generally not displayed as alternate content to the media object. It is typically used as a description when table of contents information is generated from a SMIL presentation, and typically contains more information than would be advisable to put in an alt attribute.

This attribute is deprecated in favor of using appropriate SMIL metadata markup in RDF. For example, this attribute maps well to the "description" attribute as defined by the Dublin Core Metadata Initiative [DC] .

author
The name of the author of the content contained in the element.

The value of this attribute is a CDATA text string.

copyright
The copyright notice of the content contained in the element.

The value of this attribute is a CDATA text string.

title
The title attribute as defined in the SMIL Structure module. It is strongly recommended that all media object elements have a title attribute with a brief, meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

xml:lang
Used to identify the natural or formal language for the element. For a complete description, see section 2.12 Language Identification of [XML10].

xml:lang differs from the system-language test attribute in one important respect. xml:lang provides information about the content's language independent of what implementations do with the information, whereas system-language is a test attribute with specific associated behavior (see system-language in SMIL Content Control Module for details)

## 7.10 Appendices

### 7.10.1 Appendix A: Changes to SMIL 1.0 Media Object Attributes

**clipBegin, clipEnd, clip-begin, clip-end**

With regards to the clipBegin/clip-begin and clipEnd/clip-end elements, SMIL 2.0 defines the following changes to the syntax defined in SMIL 1.0:

- Addition of the attribute names clipBegin and clipEnd as an equivalent alternative to the SMIL 1.0 clip-begin and clip-end attributes. The attribute names with hyphens are deprecated.
- If the attribute consists only of a clock value without further specification, it is assumed to be specified in normal play time, i.e. to have the metric "npt".
- A new metric called "marker" can be used to define a clip using marked time points in a media object, rather than using clock values or SMPTE values.

**Handling of new clipBegin/clipEnd syntax in SMIL 1.0 software**

Using attribute names with hyphens such as clip-begin and clip-end is problematic when using a scripting language and the DOM to manipulate these attributes. Therefore, this specification adds the attribute names clipBegin and clipEnd as an equivalent alternative to the SMIL 1.0 clip-begin and clip-end attributes. The attribute names with hyphens are deprecated.

Authors can use two approaches for writing SMIL 2.0 presentations that use the new clipping syntax and functionality ("marker", default metric) defined in this specification, but can still can be handled by SMIL 1.0 software. First, authors can use non-hyphenated versions of the new attributes that use the new functionality, and add SMIL 1.0 conformant clipping attributes later in the text.

Example:

```
<audio src="radio.wav" clipBegin="marker=song1" clipEnd="marker=moderator1"
       clip-begin="npt=0s" clip-end="npt=3:50" />
```

SMIL 1.0 players implementing the recommended extensibility rules of SMIL 1.0 [SMIL10] will ignore the clip attributes using the new functionality, since they are not part of SMIL 1.0. SMIL 2.0 players, in contrast, will ignore the clip attributes using SMIL 1.0 syntax, since they occur later in the text.

The second approach is to use the following steps:

1. Add a "system-required" test attribute to media object elements using the new functionality. The value of the "system-required" attribute would correspond to a namespace prefix whose namespace URI ( [URI] ) points to a SMIL specification which integrates the new functionality.
2. Add an alternative version of the media object element that conforms to SMIL 1.0
3. Include these two elements in a "switch" element

Example:

```
<smil xmlns:smil2="http://www.w3.org/2001/SMIL20/Language">
...
<switch>
  <audio src="radio.wav" clipBegin="marker=song1" clipEnd="marker=moderator1"
   system-required="smil2" />
  <audio src="radio.wav" clip-begin="npt=0s" clip-end="npt=3:50" />
</switch>
```

**New Accessibility Attributes**

readIndex
Allows explicit ordering for controlling assistive technology.

**New Advanced Media Attributes**

mediaRepeat
The mediaRepeat attribute was added to provide better timing control over media with intrinsic repeat behavior (such as animated GIFs).

erase
Provides a way for visual media to remain visible throughout the duration of a presentation rather by overriding the default erase behavior.

### 7.10.2 Appendix B: Changes to SMIL 1.0 Media Object Elements

**New child elements for media objects**

SMIL 1.0 only allowed anchor as a child element of a media element. In addition to anchor (now defined in the Linking module), the param is now allowed as children of a SMIL media object. Additionally, other new children may also be defined by the host language.

**The param element**

A new param element provides a generalized mechanism to attach media-specific attributes to media objects.

**The brush element**

A new brush element allows the specification of solid color media objects with no associated media.

# 8. The SMIL 2.0 Metainformation Module

**Editors:**
Thierry Michel (tmichel@w3.org), W3C

## 8.1 Introduction

This section defines the SMIL 2.0 Metainformation Module composed of a single module. This module contains elements and attributes that allow description of SMIL documents.
Since these elements and attributes are defined in a module, designers of other markup languages can choose whether or not to include this functionality in their languages.

The World Wide Web was originally built for human consumption, and although everything on it is machine-readable, this data is not machine-understandable. It is very hard to automate anything on the Web, and because of the volume of information the Web contains, it is not possible to manage it manually. Metadata is "data about data" (for example, a library catalog is metadata, since it describes publications) or specifically in the context of this specification "data describing Web resources".
The solution proposed here is to use metadata information to describe SMIL documents published on the Web.

The earlier SMIL 1.0 specification allowed authors to describe documents with a very basic vocabulary using the meta element.
The SMIL 2.0 Metainformation module defined in this specification fully supports the use of this meta element from SMIL 1.0 but it also introduces new capabilities for describing metadata using the Resource Description Framework Model and Syntax [RDFsyntax], a powerful meta information language for providing information about resources.

## 8.2 Overview of the SMIL 2.0 Metainformation module

### 8.2.1 Compatibility with SMIL 1.0

To insure backward compatibility with SMIL 1.0, the meta element as specified in the SMIL 1.0 [SMIL10] Recommendation can be used to define properties of a document (e.g., author/creator, expiration date, a list of key words, etc.) and assign values to those properties.

### 8.2.2 Extensions to SMIL 1.0

SMIL 2.0 extends SMIL 1.0 meta information functionalities with the new metadata element to host RDF statements as RDF provides a more general treatment of metadata.
RDF is a declarative language and provides a standard way for using XML to represent metadata in the form of statements about properties and relationships of items on the Web. Such items, known as resources, can be almost anything, provided it has a Web address. This means that you can associate metadata information with a SMIL document, but also a graphic, an audio file, a movie clip, and so on.

RDF is the appropriate language for metadata. The specifications for RDF can be found at:

- Resource Description Framework (RDF) Model and Syntax [RDFsyntax]
- Resource Description Framework (RDF) Schema [RDFschema]

Metadata information within an SMIL 2.0 document should be expressed in the appropriate RDF namespaces [XML-NS] and should be placed within the metadata element. (See example below.)

### 8.2.3 Multiple description schemes

RDF appears to be the ideal approach for supporting descriptors from multiple description schemes simultaneously.

Here are some suggestions for content creators regarding metadata:

- Content creators should refer to W3C metadata Recommendations [RDFsyntax] and [RDFschema] when deciding which metadata RDF schema to use in their documents.
- Content creators should refer to the Dublin Core Metadata Initiative [DC] , which is a set of generally applicable core metadata properties (e.g., Title, Creator, Subject, Description, Copyrights, etc.).
- Additionally, a specific SMIL metadata RDF Schema could contain a set of additional metadata properties that are common across most uses of multimedia.

*Note:* Individual industries or individual content creators are free to define their own metadata RDF Schema, but everyone is encouraged to follow existing metadata standards and use standard metadata schemas wherever possible to promote interchange and interoperability. If a particular standard metadata schema does not meet your needs, then it is usually better to define an additional metadata schema in RDF that is used in combination with the given standard metadata schema than to totally avoid the standard schema.

## 8.3 SMIL 2.0 Metainformation Module Syntax and Semantics

This section defines the elements and attributes that make up the functionality in the SMIL Metainformation module.

### 8.3.1 The **meta** element

*Element definition*

The **meta** element is an empty element.
Each **meta** element specifies a single property/value pair in the name and content attributes, respectively.

*Attribute definitions*

**content** = CDATA
: This attribute specifies a property's value. This specification does not list legal values for this attribute.
The content attribute is required for **meta** elements.

**name** = CDATA
: This attribute identifies a property name. The name attribute is required for **meta** elements. The list of properties for the name attribute is open-ended. This specification defines the following properties:

- **base:** The value of this property determines the base URI for all relative URIs used in the document.
  *Note*: SMIL 2.0 deprecates the base property in favor of the more general XML base url mechanisms currently being completed [XMLBase]. The language profile is responsible for specifying which of these technologies are specifically supported.
- **pics-label or PICS-Label:** The value of this property specifies a valid rating label for the document as defined by PICS [PICS] .
- **title:** The value of this property specifies the title of the presentation.

### 8.3.2 The **metadata** element

*Element definition*

The **metadata** element contains information that is also related to meta information of the document. It acts as the root element of the RDF tree. The **metadata** element can contain the following child elements:

**RDF** element and its sub-elements (refer to W3C metadata Recommendations [RDFsyntax]).

## 8.4 An Example

**This section is informative.**

Here is an example of how metadata can be included in an SMIL document. The example uses the Dublin Core version 1.0 RDF schema [DC] and an hypothetic SMIL metadata RDF Schema:

```
<?xml version="1.0" ?>
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">

 <head>

  <meta id="meta-smil1.0-b" name="Date" content="1999-10-12" />
  <meta id="meta-smil1.0-c" name="Rights" content="Copyright 1999 John Smith" />
  <meta id="meta-smil1.0-d" http-equiv="Expires" content=" 31 Dec 2001 12:00:00 GMT"/>


   <metadata id="meta-rdf">
    <rdf:RDF
        xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs = "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
        xmlns:dc = "http://purl.org/metadata/dublin_core#"
        xmlns:smilmetadata = "http://www.example.org/AudioVideo/.../smil-ns#" >

<!-- Metadata about the SMIL presentation -->
    <rdf:Description about="http://www.example.com/meta.smi"
        dc:Title="An Introduction to the Resource Description Framework"
        dc:Description="The Resource Description Framework (RDF) enables the encoding, exchange and reuse of structured metadata"
        dc:Publisher="W3C"
        dc:Date="1999-10-12"
        dc:Rights="Copyright 1999 John Smith"
        dc:Format="text/smil" >
        <dc:Creator>
           <rdf:Seq ID="CreatorsAlphabeticalBySurname">
              <rdf:li>Mary Andrew</rdf:li>
              <rdf:li>Jacky Crystal</rdf:li>
           </rdf:Seq>
        </dc:Creator>
        <smilmetadata:ListOfVideoUsed>
           <rdf:Seq ID="VideoAlphabeticalByFormatname">
              <rdf:li Resource="http://www.example.com/videos/meta-1999.mpg"/>
              <rdf:li Resource="http://www.example.com/videos/meta2-1999.mpg"/>
           </rdf:Seq>
        </smilmetadata:ListOfVideoUsed>
        <smilmetadata:Access LevelAccessibilityGuidelines="AAA"/>
    </rdf:Description>

<!-- Metadata about the video -->
    <rdf:Description about="http://www.example.com/videos/meta-1999.mpg"
        dc:Title="RDF part one"
        dc:Creator="John Smith"
        dc:Subject="Metadata,RDF"
        dc:Description="RDF basic functionalities"
        dc:Publisher="W3C Press Service"
        dc:Format="video/mpg"
        dc:Language="en"
        dc:Date="1999-10-12"
        smilmetadata:Duration="60 secs"
        smilmetadata:VideoCodec="MPEG2" >
        <smilmetadata:ContainsSequences>
           <rdf:Seq ID="ChronologicalSequences">
              <rdf:li Resource="http://www.example.com/videos/meta-1999.mpg#scene2"/>
```

```
            </rdf:Seq>
          </smilmetadata:ContainsSequences>
    </rdf:Description>

<!-- Metadata about a scene of the video -->
    <rdf:Description about="#scene1"
          dc:Title="RDF intro"
          dc:Description="Introduction to RDF functionalities"
          dc:Language="en"
          smilmetadata:Duration="30 secs"
          smilmetadata:Presenter="David Jones" >
          <smilmetadata:ContainsShots>
             <rdf:Seq ID="ChronologicalShots">
               <rdf:li>Panorama-shot</rdf:li>
               <rdf:li>Closeup-shot</rdf:li>
             </rdf:Seq>
          </smilmetadata:ContainsShots>
    </rdf:Description>
  </rdf:RDF>
 </metadata>

<!-- SMIL presentation -->
 <layout>
    <region id="a" top="5" />
 </layout>
 </head>
 <body>
 <seq>
   <video region="a" src="/videos/meta-1999.mpg" >
     <area id="scene1" begin="0s" end ="30s"/>
     <area id="scene2" begin="30s" end ="60s"/>
   </video>
   <video region="a" src="/videos/meta2-1999.mpg"/>
 </seq>
 </body>
</smil>
```

# 9. The SMIL 2.0 Structure Module

***Editors***

Warner ten Kate (warner.ten.kate@philips.com), (Philips Electronics)

Aaron Cohen (aaron.m.cohen@intel.com), (Intel)

## 9.1 Introduction

***This section is informative.***

This Section defines the SMIL structure module. The Structure module provides the base elements for structuring SMIL content. These elements act as the root in the content model of all SMIL Host Language conformant language profiles. The Structure module is a mandatory module for SMIL Host Language conformant language profiles.

The SMIL Structure module is composed of the **smil**, **head**, and **body** elements, and is compatible with SMIL 1.0 [SMIL10]. The corresponding SMIL 1.0 elements form a subset of the Structure module, both in syntax and semantics, as their attributes and content model is also exposed by the Structure module. Thus, the Structure module is backwards compatible with SMIL 1.0.

## 9.2 The SMIL 2.0 Structure Module Syntax and Semantics

***This section is normative.***

### 9.2.1 Elements and attributes

This section defines the elements and attributes that make up the SMIL 2.0 Structure module.

**The smil element**

The **smil** element acts as the root element for all SMIL Host Language conformant language profiles.

*Element attributes*

The **smil** element can have the following attributes:

id

class
: The class attribute assigns a class name or a set of class names to an element. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters.

xml:lang
: The xml:lang attribute specifies the language of an element, and is specified in XML 1.0 [XML10]. xml:lang differs from systemLanguage test attribute in one important respect. xml:lang provides information about content's language independent of what implementations do with the information, whereas systemLanguage is a test attribute with specific associated behavior (see systemLanguage in SMIL 2.0 BasicContentControl Module for details).

title
: The title attribute offers advisory information about the element for which it is set. Values of the title attribute may be rendered by user agents in a variety of ways. For instance, visual browsers frequently display the title as a "tool tip" (a short message that appears when the pointing device pauses over an object).

xmlns
: The xmlns attribute declares an XML namespace, and is defined in "Namespaces in XML" [XML-NS].

*Element content*

The smil element can contain the following elements:

head
body

**The head element**

The head element contains information that is not related to the temporal behavior of the presentation. Three types of information may be contained by head. These are meta information, layout information, and author-defined content control.

*Element attributes*

The head element can have the following attributes:

id
: Defined in id under the smil element.

class
: Defined in class under the smil element.

xml:lang
: Defined in xml:lang under the smil element.

title
: Defined in title under the smil element.

*Element content*

The head element contains elements depending on the other modules and specific syntax included in the language profile integrating this module.

**The body element**

The body element contains information that is related to the temporal and linking behavior of the document. It acts as the root element of the timing tree.

The body element has the timing semantics of a time container equal to that of the seq element [BasicTimeContainers module]. Note, that in other language profiles, where a body element from another (Structure) Module is in use, that body element may have different timing semantics. For example, in the XHTML+SMIL language profile (still in progress and not yet ready for Last Call), the body element comes from XTML, and acts as a par time container.

*Element attributes*

The body element can have the following attributes:

id
: Defined in id under the smil element.

class
: Defined in class under the smil element.

xml:lang
: Defined in xml:lang under the smil element.

title
: Defined in title under the smil element.

The timing attributes defined in the various SMIL 2.0 timing modules are part of the body element so far as the corresponding timing modules, such as BasicInlineTiming, are part of the language profile. When a timing module is included in a language profile, the features of that module should be supported on the body element just as they are supported on the other elements in the profile. For example, the syncMaster attribute should be supported on the body element if the SyncMaster module is included in the integrating profile.

*Element content*

The body element contains elements depending on the other modules and specific syntax included in the language profile integrating this module.

## 9.3 Integrating the SMIL Structure Module

***This section is normative.***

When this module is included in a language profile, the id, class, and title attributes defined in this module must be included on all elements from all modules used in the profile, including those from other module families and of non-SMIL origin. The integrating profile should also consider adding the xml:lang attribute to the applicable elements.

The SMIL Structure module is the starting module when building any SMIL Host Language conformant language profile. The Structure module may not be used for building other, non-SMIL Host Language conformant language profiles. This implies that the SMIL Structure module must at least be accompanied with the other modules mandatory for SMIL Host language conformance, and the elements in the structure module must include at least the minimum content models required for SMIL Host language conformance.

When modules from outside the SMIL 2.0 namespace are integrated in the language profile, it must be specified how the elements from those non-SMIL modules fit into the content model of the used SMIL modules (and vice versa). For example, with respect to the SMIL Structure module, the Profiling Entities in the DTD need to be overridden. This creates a so-called *hybrid document type* [XMOD]. In case of a so-called *compound document type*, the rules of XML namespaces must be satisfied [XML-NS].

# 10. The SMIL 2.0 Timing and Synchronization Module

**Editors:**
Patrick Schmitz (cogit@ludicrum.org), (Microsoft)
Jeff Ayars (jeffa@real.com), (RealNetworks)
Bridie Saccocio (bridie@real.com), (RealNetworks)
Muriel Jourdan (Muriel.Jourdan@inrialpes.fr), (INRIA)

## 10.1 Introduction

*This section is informative*

SMIL 1.0 solved fundamental media synchronization problems and defined a powerful way of choreographing multimedia content. SMIL 2.0 extends the timing and synchronization support, adding capabilities to the timing model and associated syntax. Some SMIL 1.0 syntax has been changed or deprecated. This section of the document specifies the Timing and Synchronization module.

There are two intended audiences for this module: implementers of SMIL 2.0 document viewers or authoring tools, and authors of other XML languages who wish to integrate timing and synchronization support. A language with which this module is integrated is referred to as a *host language*. A document containing SMIL Timing and Synchronization elements and attributes is referred to as a *host document*.

As this module is used in different profiles (i.e. host languages), the associated syntax requirements may vary. Differences in syntax should be minimized as much as is practical.

SMIL 2.0 Timing and Synchronization support is broken down into 15 modules, allowing broad flexibility for language designers integrating this functionality. These modules are described in Appendix E: SMIL Timing and Synchronization modules.

## 10.2 Overview of SMIL timing

*This section is informative*

SMIL Timing defines elements and attributes to coordinate and synchronize the presentation of *media* over time. The term *media* covers a broad range, including *discrete* media types such as still images, text, and vector graphics, as well as *continuous* media types that are intrinsically time-based, such as video, audio and animation.

Three synchronization elements support common timing use-cases:

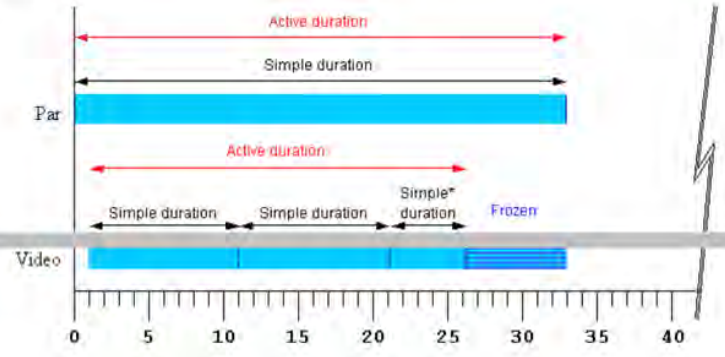
- The <seq> element plays the child elements one after another in a *sequence*.
- The <excl> element plays one child at a time, but does not impose any order.
- The <par> element plays child elements as a group (allowing "parallel" playback).

These elements are referred to as *time containers*. They group their contained children together into coordinated timelines.

SMIL Timing also provides attributes that can be used to specify an element's timing behavior. Elements have a begin, and a *simple duration*. The begin can be specified in various ways - for example, an element can begin at a given time, or based upon when another element begins, or when some event (such as a mouse click) happens. The *simple duration* defines the basic presentation duration of an element. Elements can be defined to repeat the simple duration, a number of times or for an amount of time. The simple duration and any effects of repeat are combined to define the *active duration*. When an element's active duration has ended, the element can either be removed from the presentation or *frozen* (held in its final state), e.g. to fill any gaps in the presentation.

An element *becomes active* when it begins its active duration, and *becomes inactive* when it ends its active duration. Within the active duration, the element is *active*, and outside the active duration, the element is *inactive*.

Figure 1 illustrates the basic support of a repeating element within a simple <par> time container. The corresponding syntax is included with the diagram.

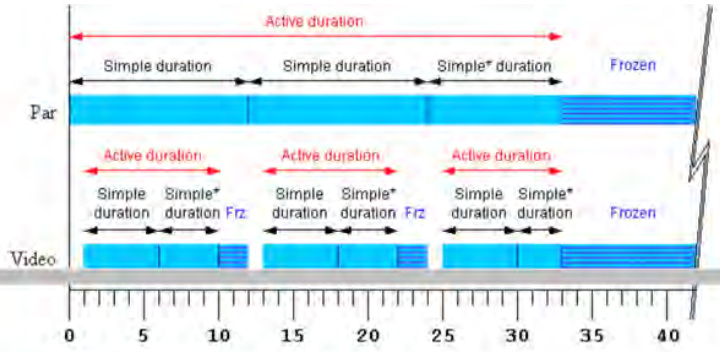



```
<par begin="0s" dur="33s">
   <video begin="1s" dur="10s" repeatCount="2.5" fill="freeze" .../>
</par>
```

**Figure 1 - Strip diagram of basic timing support. The starred "Simple*" duration indicates that the simple duration is partial (i.e. it is cut off early).**

The attributes that control these aspects of timing can be applied not only to media elements, but to the time containers as well. This allows, for example, an entire sequence to be repeated, and to be coordinated as a unit with other media and time containers. While authors can specify a particular simple duration for a time container, it is often easier to leave the duration unspecified, in which case the simple duration is defined by the contained child elements. When an element does not specify a simple duration, the time model defines an *implicit* simple duration for the element. For example, the implicit simple duration of a sequence is based upon the sum of the active durations of all the children.

Each time container also imposes certain *defaults* and *constraints* upon the contained children. For example in a <seq>, elements begin *by default* right after the previous element ends, and in all time containers, the active duration of child elements is *constrained* not to extend past the end of the time container's simple duration. Figure 2 illustrates the effects of a repeating <par> time container as it constrains a <video> child element.




```
<par begin="0s" dur="12s" repeatDur="33s" fill="freeze" >
   <video begin="1s" dur="5s" repeatCount="1.8" fill="freeze" .../>
</par>
```

**Figure 2 - Strip diagram of time container constraints upon child elements. The starred "Simple*" durations indicate that the simple duration is partial (i.e. it is cut off early).**

The SMIL *Timing Model* defines how the time container elements and timing attributes are interpreted to construct a *time graph*. The *time graph* is a model of the presentation schedule and synchronization relationships. The time graph is a dynamic structure, changing to reflect the effect of user events, media delivery, and DOM control of the presentation. At any given instant, the time graph models the document at that instant, and the semantics described in this module. However, as user events or other factors cause changes to elements, the semantic rules are re-evaluated to yield an updated time graph.

When a begin or end value refers to an event, or to the begin or active end of another element, it may not be possible to calculate the time value. For example, if an element is defined to begin on some event, the begin time will not be known until the event happens. Begin and end values like this are described as *unresolved*. When such a time becomes known (i.e. when it can be calculated as a presentation time), the time is said to be *resolved*. A resolved time is said to be *definite* if it is not the value "indefinite". See also the discussion of Unifying scheduled and interactive timing.

In an ideal environment, the presentation would perform precisely as specified. However, various real-world limitations (such as network delays) can influence the actual playback of media. How the presentation application adapts and manages the presentation in response to media playback problems is termed *runtime synchronization behavior*. SMIL includes attributes that allow the author to control the runtime synchronization behavior for a presentation.

## 10.3 Language definition

*This section is informative*

The timing model is defined by building up from the simplest to the most complex concepts: first the basic timing and simple duration controls, followed by the attributes that control repeating and constraining the active duration. Finally, the elements that define time containers are presented.

### 10.3.1 Attributes

This section defines the set of timing attributes that are common to all of the SMIL synchronization elements.

Unless otherwise specified below, if there is any error in the argument value syntax for an attribute, the attribute will be ignored (as though it were not specified).

**The begin and dur attributes: basic timing support**

*This section is informative*

The basic timing for an element is described using the begin and dur attributes. Authors can specify the begin time of an element in a variety of ways, ranging from simple clock times to the time that an event (e.g. a mouse click) happens. The simple duration of an element is specified as a simple time value. The begin attribute syntax is described below. The normative syntax rules for each attribute value variant are described in Timing attribute value grammars; an attribute value syntax summary is provided here as an aid to the reader.

*This section is normative*

begin : **smil-1.0-syncbase-value | begin-value-list**
Defines when the element becomes active.
The attribute value is either a SMIL 1.0 syncbase declaration, or a semi-colon separated list of values.

**smil-1.0-syncbase-value : "id(" Id-value ")" ( "(" ( "begin" | "end" | Clock-value ) ")" )?**
*Deprecated*. Describes a syncbase and an offset from that syncbase. The element begin is defined relative to the begin or active end of another element.
**begin-value-list : begin-value (";" begin-value-list )?**
A semi-colon separated list of begin values. The interpretation of a list of begin times is detailed in the section Evaluation of begin and end time lists.
**begin-value : ( offset-value | syncbase-value | event-value | repeat-value | accesskey-value | media-marker-value | wallclock-sync-value | "indefinite" )**
Describes the element begin.
**offset-value : ( "+" | "-" )? Clock-value**
Describes the element begin as an offset from an implicit syncbase. The definition of the implicit syncbase depends upon the element's parent time container. The offset is measured in parent simple time.
**syncbase-value : ( Id-value "." ( "begin" | "end" ) ) ( ( "+" | "-" ) Clock-value )?**
Describes a syncbase and an offset from that syncbase. The element begin is defined relative to the begin or active end of another element.
**event-value : ( Id-value "." )? ( event-ref ) ( ( "+" | "-" ) Clock-value )?**

Describes an event and an optional offset that determine the element begin. The element begin is defined relative to the time that the event is raised. Events may be any event defined for the host language in accordance with [DOM2Events]. These may include user-interface events, event-triggers transmitted via a network, etc. Details of event-based timing are described in the section below on Unifying Event-based and Scheduled Timing.

**repeat-value : ( Id-value "." )? "repeat(" integer ")" ( ( "+" | "-" ) Clock-value )?**
Describes a qualified repeat event. The element begin is defined relative to the time that the repeat event is raised with the specified iteration value.

**accesskey-value : "accesskey(" character ")"( ( "+" | "-" ) Clock-value )?**
Describes an accesskey that determines the element begin. The element begin is defined relative to the time that the accesskey character is input by the user.

**media-marker-value : Id-value ".marker(" marker-name ")"**
Describes the element begin as a named marker time defined by a media element.

**wallclock-sync-value : "wallclock(" wallclock-value ")"**
Describes the element begin as a real-world clock time. The wallclock time syntax is based upon syntax defined in [ISO8601].

**"indefinite"**
The begin of the element will be determined by a "beginElement()" method call or a hyperlink targeted to the element. The SMIL Timing and Synchronization DOM methods are described in the Reserveed DOM methods section. Hyperlink-based timing is described in the Hyperlinks and timing section.

*Begin value semantics*

*This section is normative*

- Children of a seq can only specify a non-negative offset value for begin (see The seq element).
- If no begin is specified, the default timing is dependent upon the time container.
- If there is a syntax error in any individual value in the list of begin or end values (i.e. the value does not conform to the defined syntax for any of
- A time value may conform to the defined syntax but still be invalid (e.g. if an unknown element is referenced by ID in a syncbase value). If there is such an evaluation error in an individual value in the list of begin or end values, the individual value will be will be treated as though "indefinite" were specified, and the rest of the list will not be processed normally. If no legal value is specified for a begin or end attribute, the element assumes an "indefinite" begin or end time (respectively).
- The deprecated smil-1.0-syncbase-values are semantically equivalent to the following smil 2.0 begin-value types:
  - `id(Id-value)(begin)` is equivalent to `Id-value.begin`
  - `id(Id-value)(end)` is equivalent to `Id-value.end`
  - `id(Id-value)(Clock-value)` is equivalent to `Id-value.begin+ Clock-value`

*This section is informative*

Children of a par begin by default when the par begins (equivalent to `begin="0s"`). Children of a seq begin by default when the previous child ends its active duration (equivalent to `begin="0s"`); the first child begins by default when the parent seq begins. Children of an excl default to a begin value of "`indefinite`".

The begin value can specify a list of times. This can be used to specify multiple "ways" or "rules" to begin an element, e.g. if any one of several events is raised. A list of times can also define multiple begin times, allowing the element to play more than once (this behavior can be controlled, e.g. to only allow the earliest begin to actually be used - see also The restart attribute).

In general, the earliest time in the list determines the begin time of the element. There are additional constraints upon the evaluation of the begin time list, detailed in Evaluation of begin and end time lists.

Note that while it is legal to include "indefinite" in a list of values for begin, "indefinite" is only really useful as a single value. Combining it with other values does not impact begin timing, as DOM begin methods can be called with or without specifying "indefinite" for begin.

When a begin time is specified as a syncbase variant, a marker value or a wallclock value, the defined time must be converted by the implementation to a time that is relative to the parent time container (i.e. to the equivalent of an offset value). This is known as *timespace conversion*, and is detailed in the section Converting between local and global times.

*Handling negative offsets for begin*

*This section is informative*

The use of negative offsets to define begin times merely defines the synchronization relationship of the element. It does not in any way override the time container constraints upon the element, and it cannot override the constraints of presentation time.

*This section is normative*

- For children of <par> and <excl> time containers, the computed offset relative to the parent begin time may be negative.
- A begin time may be specified with a negative offset relative to an event or to a syncbase that is not initially resolved. When the syncbase or eventbase time is resolved, the computed time may be in the past.

The computed begin time defines the *scheduled synchronization relationship* of the element, even if it is not possible to begin the element at the computed time. The time model uses the computed begin time, and not the observed time of the element begin.

*This section is informative*

If an element has a begin time that resolves to a time before the parent time container begins, the parent time container constraint still applies. For example:

```
<par>
   <video id="vid" begin="-5s" dur="10s" src="movie.mpg" />
   <audio begin="vid.begin+2s" dur="8s" src="sound.au" />
</par>
```

The video element cannot begin before the par begins. The begin is simply defined to occur *"in the past"* when the par begins. The viewer will observe that the video begins 5 seconds into the media, and ends after 5 seconds. Note that the audio element begins relative to the video begin, and that the computed begin time is used, and not the observed begin time as constrained by the parent. Thus the audio begins 3 seconds into the media, and also lasts 5 seconds.

The behavior can be thought of as a clipBegin value applied to the element, that only applies to the first iteration of repeating elements. In the example above, if either element were defined to repeat, the second and later iterations of the media would play from the beginning of the media (see also The repeatCount, repeatDur, and repeat attributes: repeating elements).

*This section is normative*

- When a begin time is resolved to be in the past (i.e., before the current presentation time), the element begins immediately, but acts as though it had begun at the specified time (playing from an offset into the media).

The behavior can be thought of as a clipBegin value applied to the element, that only applies to the first iteration of repeating elements.

The element will actually begin at the time computed according to the following algorithm:

```
Let o be the offset value of a given begin value,
d be the associated simple duration,
AD be the associated active duration.
Let rAt be the time when the begin time becomes resolved.
Let rTo be the resolved sync-base or event-base time without the offset
Let rD be rTo - rAt.  If rD < 0 then rD is set to 0.

If AD is indefinite, it compares greater than any value of o or ABS(o).
REM( x, y ) is defined as x - (y * floor( x/y )).
If y is indefinite or unresolved, REM( x, y ) is just x.

Let mb = REM( ABS(o), d ) - rD

If ABS(o) >= AD then the element does not begin.
Else if mb >= 0 then the media begins at mb.
Else the media begins at mb + d.
```

If the element repeats, the iteration value of the `repeat` event has the calculated value based upon the above computed begin time, and not the observed number of repeats.

*This section is informative*

Thus for example:

```
<smil ...>
...
<ref begin="foo.activateEvent-8s" dur="3s" repeatCount="10" .../>
...
</smil>
```

The element begins when the user activates (for example, clicks on) the element "foo". Its calculated begin time is actually 8 seconds earlier, and so it begins to play at 2 seconds into the 3 second simple duration, on the third repeat iteration. One second later, the fourth iteration of the element will begin, and the associated `repeat` event will have the iteration value set to 3 (since it is zero based). The element will end 22 seconds after the activation. The `beginEvent` event is raised when the element begins, but has a time stamp value that corresponds to the defined begin time, 8 seconds earlier. Any time dependents are activated relative to the computed begin time, and not the observed begin time.

Note: If script authors wish to distinguish between the computed repeat iterations and observed repeat iterations, they can count actual `repeat` events in the associated event handler.

*Negative begin delays*

A begin time specifies a synchronization relationship between the element and the parent time container. Syncbase variants, eventbase, marker and wallclock timing are implicitly converted to an offset on the parent time container, just as an offset value specifies this directly. For children of a seq, the result is always a positive offset from the begin of the seq time container. However, for children of par and excl time containers the computed offset relative to the parent begin time may be negative.

Note that an element cannot actually begin until the parent time container begins. An element with a negative time delay behaves as if it had begun earlier. The presentation effect for the element (e.g. the display of visual media) is equivalent to that for a clipBegin value (with the same magnitude) for the first -- and only the first -- iteration of a repeated element. If no repeat behavior is specified, the element presentation effect of a negative begin offset is equivalent to a clipBegin specification with the same magnitude as the offset value. Nevertheless, the *timing* side effects are *not* equivalent to a clipBegin value as described. Time dependents of the begin value will behave as though the element had begun earlier.

*Dur value semantics*

The length of the simple duration is specified using the dur attribute. The dur attribute syntax is described below.

*This section is normative*

dur
: Specifies the simple duration.
The attribute value can be any of the following:

  **Clock-value**
  : Specifies the length of the simple duration, measured in element active 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.

  **"indefinite"**
  : Specifies the simple duration as indefinite.

If there is any error in the argument value syntax for dur, the attribute will be ignored (as though it were not specified).

If the "`media`" attribute value is used on an element that does not define media (e.g. on the SMIL 2.0 time container elements par, seq and excl), the attribute will be ignored (as though it were not specified). Contained media such as the children of a par are not considered media directly associated with the element.

If the element does not have a (valid) dur attribute, the simple duration for the element is defined to be the implicit duration of the element. The implicit duration depends upon the type of an element. The primary distinction is between different types of media elements and time containers. If the media

element has no timed children, it is described as a *simple media element*.

- For simple media elements that specify *continuous* media (i.e. media with an inherent notion of time), the implicit duration is the intrinsic duration of the media itself - e.g. video and audio files have a defined duration. Note that clipBegin and clipEnd attributes on a media element can override the intrinsic media duration, and will define the implicit duration. See also the Media Object module.
- For simple media elements that specify *discrete* media (some times referred to as "static" media), the implicit duration is defined to be 0.
- For par, seq and excl time containers, and media elements that are also time containers, the implicit simple duration is a function of the type of the time container and of itsendsync attribute. For details see the section Time container durations.

If the author specifies a value for dur that is *shorter* than the implicit duration for an element, the implicit duration will be cut short by the specified simple duration.

If the author specifies a simple duration that is *longer* than the implicit duration for an element, the implicit duration of the element is extended to the specified simple duration:

- For a discrete media element, the media will be shown for the specified simple duration.
- For a continuous media element, the ending state of the media (e.g. the last frame of video) will be shown from the end of the intrinsic media duration to the end of the specified simple duration. This only applies to visual media - aural media will simply stop playing (i.e. be silent).
- For a seq time container, the last child is frozen until the end of the simple duration of the seq if and only if its fill behavior is "freeze" or "hold" (otherwise the child just ends without freezing).
- Children of a par or excl are frozen until the end of the simple duration of the par or excl if and only if the children's fill behavior is "freeze" or "hold" (otherwise the children just ends without freezing).

Note that when the simple duration is "indefinite", some simple use cases can yield surprising results. See the related example #4 in Appendix B.

Examples

The following example shows simple offset begin timing. The <audio> element begins 5 seconds after the <par> time container begins, and ends 4 seconds later.

```
<par>
  <audio src="song1.au" begin="5s" dur="4s" />
</par>
```

The following example shows syncbase begin timing. The <img> element begins 2 seconds after the <audio> element begins.

```
<par>
  <audio id="song1" src="song1.au" />
  <img src="img1.jpg" begin="song1.begin+2s" />
</par>
```

Elements can also be specified to begin in response to an event. In this example, the image element begins (appears) when the user clicks on element "show". The image will end (disappear) 3 and a half seconds later.

```
<smil ...>
...
<text id="show" ... />
<img begin="show.activateEvent" dur="3.5s" ... />
...
</smil ...>
```

**The end attribute: controlling active duration**

*This section is informative*

SMIL 2.0 provides an additional control over the active duration. The end attribute allows the author to constrain the active duration by specifying an end value using a simple offset, a time base, an event-base, a syncbase, or DOM methods calls. The rules for combining the attributes to compute the active duration are presented in the section, Computing the active duration.

The normative syntax rules for each attribute value variant are described in the section Timing attribute value grammars; a syntax summary is provided here as an aid to the reader.

*This section is normative*

end : **smil-1.0-syncbase-value | end-value-list**
Defines an end value for the element that can constrain the active duration.
The attribute value is either a SMIL 1.0 syncbase declaration, a semi-colon separated list of values.

**smil-1.0-syncbase-value : "id(" Id-value ")" ( "(" ( "begin" | "end" | Clock-value ) ")" )?**
*Deprecated.* Describes a syncbase and an offset from that syncbase. The end value is defined relative to the begin or active end of another element.

**end-value-list : end-value (";" end-value-list )?**
A semi-colon separated list of end values. The interpretation of a list of end times is detailed in the section Evaluation of begin and end time lists.

**end-value : ( offset-value | syncbase-value | event-value | repeat-value | accesskey-value | media-marker-value | wallclock-sync-value | "indefinite" )**
Describes the end value of the element.

**offset-value : ( "+" | "-" )? Clock-value**
Describes the end value as an offset from an implicit syncbase. The definition of the implicit syncbase depends upon the element's parent time container. The offset is measured in parent simple time.

**syncbase-value : ( Id-value "." ( "begin" | "end" ) ) ( ( "+" | "-" ) Clock-value )?**
Describes a syncbase and an offset from that syncbase. The end value is defined relative to the begin or active end of another element.

**event-value : ( Id-value "." )? ( event-ref ) ( ( "+" | "-" ) Clock-value )?**
Describes an event and an optional offset that determine the end value. The end value is defined relative to the time that the event is raised. Events may be any event defined for the host language in accordance with [DOM2Events]. These may include user-interface events, event-triggers transmitted via a network, etc. Details of event-based timing are described in the section below on Unifying Event-based and Scheduled Timing.

**repeat-value : ( Id-value "." )? "repeat(" integer ")" ( ( "+" | "-" ) Clock-value )?**
Describes a qualified repeat event. The end value is defined relative to the time that the repeat event is raised with the specified

iteration value.

**accesskey-value : "accesskey(" character ")"( ( "+" | "-" ) Clock-value )?**
Describes an accesskey that determines the end value. The end value is defined as the time that the accesskey character is input by the user.

**media-marker-value : Id-value ".marker(" marker-name ")"**
Describes the end value as a named marker time defined by a media element.

**wallclock-sync-value : "wallclock(" wallclock-value ")"**
Describes the end value as a real-world clock time. The wallclock time is based upon syntax defined in [ISO8601].

**"indefinite"**
The end value of the element will be determined by an `endElement()` method call.
The SMIL Timing and Synchronization DOM methods are described in the Reserved DOM methods section.

If an end attribute is specified but none of dur, repeatCount and repeatDur are specified, the simple duration is defined to be indefinite, and the end value constrains this to define the active duration. The behavior of the simple duration in this case is defined in Dur value semantics, as though dur had been specified as "indefinite".

If the end value becomes resolved while the element is still active, and the resolved time is in the past, the element should end the active duration immediately. Time dependents defined relative to the end of this element should be resolved using the computed active end (which may be in the past), and not the observed active end.

The deprecated smil-1.0-syncbase-values are semantically equivalent to the following smil 2.0 end-value types:

- `id(Id-value)(begin)` is equivalent to `Id-value.begin`
- `id(Id-value)(end)` is equivalent to `Id-value.end`
- `id(Id-value)(Clock-value)` is equivalent to `Id-value.begin+Clock-value`

*This section is informative*

The end value can specify a list of times. This can be used to specify multiple "ways" or "rules" to end an element, e.g. if any one of several events is raised. A list of times can also define multiple end times that can correspond to multiple begin times, allowing the element to play more than once (this behavior can be controlled - see also The restart attribute).

In the following example, the dur attribute is not specified, and so the simple duration is defined to be the implicit media duration. In this case (and this case only) the value of end will extend the active duration if it specifies a duration greater than the implicit duration. The video will be shown for 8 seconds, and then the last frame will be shown for 2 seconds.

```
<video end="10s" src="8-SecondVideo.mpg" .../>
```

If an author wishes to specify the implicit duration as well as an end constraint, the dur attribute can be specified as "`media`". In the following example, the element will end at the earlier of the intrinsic media duration, or a mouse click:

```
<html ...>
...
<video dur="media" end="click" src="movie.mpg" .../>
...
</html>
```

These cases arise from the use of negative offsets in the sync-base and event-base forms, and authors should be aware of the complexities this can introduce. See also Handling negative offsets for end.

In the following example, the active duration will end at the earlier of 10 seconds, or the end of the "foo" element. This is particularly useful if "foo" is defined to begin or end relative to an event.

```
<audio src="foo.au" dur="2s" repeatDur="10s"
       end="foo.end" .../>
```

In the following example, the active duration will end at 10 seconds, and will cut short the simple duration defined to be 20 seconds. The effect is that only the first half of the element is actually played. For a simple media element, the author could just specify this using the dur attribute. However in other cases, it is sometimes important to specify the simple duration independent of the active duration.

```
<par>
   <audio src="music.au" dur="20s" end="10s" ... />
</par>
```

In the following example, the element begins when the user activates (e.g., clicks on) the "gobtn" element. The active duration will end 30 seconds after the parent time container begins.

```
<smil ...>
...
<par>
<audio src="music.au" begin="gobtn.activateEvent" repeatDur="indefinite"
          end="30s" ... />
     <img src="foo.jpg" dur="40s" .../>
</par>
...
</smil>
```

Note that if the user has not clicked on the target element before 30 seconds elapse, the element will never begin. In this case, the element has no active duration and no active end.

The defaults for the event syntax make it easy to define simple interactive behavior. The following example stops the image when the user clicks on the element.

```
<html ...>
...
<img src="image.jpg" end="click" />
...
</html>
```

Using end with an event value enables authors to end an element based on either an interactive event or a maximum active duration. This is sometimes known as *lazy interaction*.

In this example, a presentation describes factory processes. Each step is a video, and set to repeat 3 times to make the point clear. Each element can also be ended by clicking on the video, or on some element "next" that indicates to the user that the next step should be shown.

```
<smil ...>
```

```
...
<seq>
  <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
  <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
  <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
  <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
  <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
</seq>
...
</smil>
```

In this case, the active end of each element is defined to be the earlier of 15 (5s dur * 3 repeats) seconds after it begins, or a click on "next". This lets the viewer sit back and watch, or advance the presentation at a faster pace.

*Handling negative offsets for end*

*This section is normative*

- An end time may be specified with a negative offset relative to an event or to a syncbase that is not initially resolved.
- When the syncbase or eventbase time is resolved, the computed time may be in the past.
- The computed time defines the *scheduled synchronization relationship* of the element, even if it is not possible to end the element at the computed time.
- When an end time is defined to be in the past, the element ends immediately. The defined end time is the computed time, and not the observed or performed time of the element end.

**The min and max attributes: more control over the active duration**

*This section is informative*

The min/max attributes provide the author with a way to control the lower and upper bound of the element active duration.

*This section is normative*

min
: 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 element active time.
  Value must be greater than or equal to 0.

  **"media"**
  : Specifies the minimum value of the active duration as the intrinsic media duration. This is only valid for elements that define media.

If there is any error in the argument value syntax for min, the attribute will be ignored (as though it were not specified).

The default value for min is "0". This does not constrain the active duration at all.

max
: 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 element active time.
  Value must be greater than 0.

  **"media"**
  : Specifies the maximum value of the active duration as the intrinsic media duration. This is only valid for elements that define media.

  **"indefinite"**
  : The maximum value of the duration is indefinite, and so is not constrained.

If there is any error in the argument value syntax for max, the attribute will be ignored (as though it were not specified).

The default value for max is "indefinite". This does not constrain the active duration at all.

If the "media" argument value is specified for either min or max on an element that does not define media (e.g. on the SMIL 2.0 time container elements par, seq and excl), the respective attribute will be ignored (as though it were not specified). Contained media such as the children of a par are not considered media directly associated with the element.

If both min and max attributes are specified then the max value must be greater than or equal to the min value. If this requirement is not fulfilled then both attributes are ignored.

The rule to apply to compute the active duration of an element with min or max specified is the following: Each time the active duration of an element is computed (i.e. for each interval of the element if it begins more than once), this computation is made without taking into account the min and max attributes (by applying the algorithm described in Computing the active duration). The result of this step is checked against the min and max bounds. If the result is within the bounds, this first computed value is correct. Otherwise two situations may occur:

- if the first computed duration is greater than the max value, the active duration of the element is defined to be equal to the max value (see the first example below).
- if the first computed duration is less than the min value, the active duration of the element becomes equal to the min value and the behavior of the element is as follows :
  - if the repeating duration (or the simple duration if the element doesn't repeat) of the element is greater than min then the element is played normally for the (min constrained) active duration. (see the second and third examples below).
  - otherwise the element is played normally for its repeating duration (or simple duration if the element does not repeat) and then is frozen or not shown depending on the value of the fill attribute (see the fourth and fifth examples below).

*This section is informative*

The following examples illustrate some simple use cases for min and max attributes:

Example 1. In the following example, the video will only play for 10 seconds.

```
<smil ...>
...
<par >
   <video id="video_of_15s" max="10s".../>
</par>
...
</smil>
```

Example 2. In the following example, if an activate event happens before 10 seconds, this activation (e.g. click) does not interrupt the video immediately, but the video plays until 10 seconds and then stops. If a click event happens after 10 seconds, the video plays (repeating) until the click happens. Note, the endEvent is only raised if a click occurs after 10 seconds, not at the simple end of each repeat.

```
<smil ...>
...
<par >
   <video id="video_of_15s" repeatDur="indefinite" end="activateEvent" min="10s".../>
</par>
...
</smil>
```

Example 3. In the following example, if an activate event happens on element "foo" at 5 seconds, this event does not end the time container immediately, but rather at 12 seconds. The simple duration is defined to be "indefinite" (because an end attribute is specified with no dur attribute), and so the time container plays normally until it ends at 12 seconds.

```
<smil ...>
...
<par end="foo.activateEvent" min="12s" >
   <video id="video_of_15s" .../>
   <video id="video_of_10s" .../>
</par>
...
</smil>
```

Example 4. In the following example, if a click event happens on the first video at 5 seconds, then the simple duration of the time container is computed as 5 seconds. Respecting the fill attribute in the time between the end of the simple duration and the end of the active duration, the two videos are frozen between 5 seconds and 12 seconds.

```
<html ...>
...
<par endsync="first" min="12s" fill="freeze" >
   <video id="video_of_15s" end="click" .../>
   <video id="video_of_10s" .../>
</par>
...
```

Example 5. In the following example, the time container simple duration is defined to be 5 seconds, and the min constraint defines the active duration to be 12 seconds. Since the default value of fill in this case is "remove", nothing is shown for the time container between 5 seconds and 12 seconds.

```
<par dur="5s" min="12s" >
   <video id="video_of_15s"/>
   <video id="video_of_10s" />
</par>
```

*The min attribute and negative begin times*

If an element is defined to begin before its parent (e.g. with a simple negative offset value), the min duration is measured from the calculated begin time not the observed begin (see example 1 below). This means that the min value may have no observed effect (as in example 2 below).

Example 1. In the following example, the image will be displayed from the beginning of the time container for 2 seconds.

```
<par>
   <img id="img" begin="-5s" min="7s" dur="5s" .../>
</par>
```

Example 2. In the following example, the image will not be displayed at all.

```
<par>
   <img id="img" begin="-5s" min="4s" dur="2s" .../>
</par>
```

See also the sections The min attribute and restart and Time container constraints on child durations.

**Timing attribute value grammars**

*This section is normative*

The syntax specifications are defined using EBNF notation as defined in [XML10]

In the syntax specifications that follow, allowed white space is indicated as "S", defined as follows (taken from the [XML10] definition for 'S'):

*Begin values*

*This section is normative*

A begin-value-list is a semi-colon separated list of timing specifiers:

```
begin-value-list ::= begin-value (S? ";" S? begin-value-list )?
begin-value      ::= (offset-value | syncbase-value
                    | event-value | repeat-value | accesskey-value
                    | media-marker-value | wallclock-sync-value
                    | "indefinite" )
```

*End values*

*This section is normative*

An end-value-list is a semi-colon separated list of timing specifiers:

```
end-value-list ::= end-value (S? ";" S? end-value-list )?
end-value      ::= (offset-value | syncbase-value
                  | event-value | repeat-value | accesskey-value
                  | media-marker-value | wallclock-sync-value
                  | "indefinite" )
```

*Parsing timing specifiers*

Several of the timing specification values have a similar syntax. To parse an individual item in a value-list, the following approach defines the correct interpretation. In addition, Id-values and Event-symbols are XML NMTOKEN values and as such are allowed to contain the full stop '.' and hyphen-minus '-' characters. The reverse solidus character '\' must be used to escape these characters within Id-values and Event-symbols, otherwise these characters will be interpreted as the full stop separator and hyphen-minus sign, respectively. Once these rules are interpreted, but before Id-values in syncbase values, event values, or media-marker values are further handled, all leading and embedded escape characters should be removed.

1. Strip any leading, trailing, or intervening white space characters.
2. If the value begins with a number or numeric sign indicator (i.e. "+" or "-"), this value should be parsed as an offset value.
3. Else if the value begins with the unescaped token "wallclock", it should be parsed as a wallclock-sync-value.
4. Else if the value is the unescaped token "indefinite", it should be parsed as the value "indefinite".
5. Else: Build a token substring up to but not including any sign indicator (i.e. strip off any offset, parse that separately, and add it to the result of this step). In the following, any '.' characters preceded by a reverse solidus '\' escape character should not be treated as a separator, but as a normal token character.
    1. If the token contains no '.' separator character, then the value should be parsed as an event-value with an unspecified (i.e. default) eventbase-element.
    2. Else if the token ends with the unescaped string ".begin" or ".end", then the value should be parsed as a syncbase-value.
    3. Else if the token contains the unescaped string ".marker(", then the value should be parsed as a media-marker-value.
    4. Else, the value should be parsed as an event-value (with a specified eventbase-element).

This approach allows implementations to treat the tokens **`wallclock`** and **`indefinite`** as reserved element IDs, and **`begin`**, **`end`** and **`marker`** as reserved event names, while retaining an escape mechanism so that elements and events with those names may be referenced.

*Clock values*

Clock values have the following syntax:

```
Clock-value         ::= ( Full-clock-value | Partial-clock-value | Timecount-value )
Full-clock-value    ::= Hours ":" Minutes ":" Seconds ("." Fraction)?
Partial-clock-value ::= Minutes ":" Seconds ("." Fraction)?
Timecount-value     ::= Timecount ("." Fraction)? (Metric)?
Metric              ::= "h" | "min" | "s" | "ms"
Hours               ::= DIGIT+; any positive number
Minutes             ::= 2DIGIT; range from 00 to 59
Seconds             ::= 2DIGIT; range from 00 to 59
Fraction                ::= DIGIT+
Timecount           ::= DIGIT+
2DIGIT                  ::= DIGIT DIGIT
DIGIT                       ::= [0-9]
```

For Timecount values, the default metric suffix is "s" (for seconds). No embedded white space is allowed in clock values, although leading and trailing white space characters will be ignored.

The following are examples of legal clock values:

- Full clock values:
  - `02:30:03` = 2 hours, 30 minutes and 3 seconds
  - `50:00:10.25` = 50 hours, 10 seconds and 250 milliseconds
- Partial clock value:
  - `02:33` = 2 minutes and 33 seconds
  - `00:10.5` = 10.5 seconds = 10 seconds and 500 milliseconds
- Timecount values:
  - `3.2h` = 3.2 hours = 3 hours and 12 minutes
  - `45min` = 45 minutes
  - `30s` = 30 seconds
  - `5ms` = 5 milliseconds
  - `12.467` = 12 seconds and 467 milliseconds

Fractional values are just (base 10) floating point definitions of seconds. The number of digits allowed is unlimited (although actual precision may vary among implementations).
For example:

```
00.5s = 500 milliseconds
00:00.005 = 5 milliseconds
```

*Offset values*

Offset values are used to specify when an element should begin or end relative to its syncbase.

*This section is normative*

An offset value has the following syntax:

```
offset-value   ::= (( S? "+" | "-" S? )? ( Clock-value )
```

- An offset value allows an optional sign on a clock value, and is used to indicate a positive or negative offset.
- The offset is measured in parent simple time.

The implicit syncbase for an offset value is dependent upon the time container:

- For children of a <par> or an <excl>, the offset is relative to the begin of the parent <par> or <excl>.
- For children of a <seq>, the offset is relative to the active end of the previous child. If there is no previous child, the offset is relative to the begin of the parent <seq>. See also The seq time container.

*SMIL 1.0 begin and end values*

*Deprecated.*

```
smil-1-syncbase-value  ::= "id(" Id-value ")"
                           ( "(" ( "begin" | "end" | Clock-value) ")" )?
```

*ID-Reference values*

*This section is normative*

ID reference values are references to the value of an "id" attribute of another element in the document.

```
Id-value                ::= Id-ref-value
Id-ref-value             ::= IDREF | Escaped-Id-ref-value
Escaped-Id-ref-value     ::= Escape-Char NMTOKEN
Escape-Char              ::= "\"
```

- The IDREF is a legal XML identifier.
- If the Id-ref-value is not an IDREF, then it is treated as an Escaped-ID-ref-value.
- The Escaped-Id-ref-value allows the use of a SMIL 2.0 reserved symbol as an IDREF value for an attribute by prefixing the reserved symbol with a backslash character. In this case the value should be treated as an IDREF and not as the reserved symbol, and the leading backslash is significant in the parsing as detailed in the section on parsing XML identifiers in begin and end values.
- Single characters can also be escaped by using the backslash character, and the character, minus the backslash, should be treated as a literal part of the NMTOKEN.

If the element referenced by the IDREF is ignored as described in the Content Control modules (e.g. if it specifies test attributes that evaluate false), the associated time value (i.e., the syncbase value or the eventbase value that specifies the Id-value) will be considered invalid.

*This section is informative*

The semantics of ignored elements may change in a future version of SMIL. One possible semantic is that the associated sync arc arguments will not be invalid, but will instead always be "unresolved". When this behavior needs to be simulated in this version of SMIL Timing and Synchronization, an author can include the value "indefinite" in the list of values for the begin or end attribute.

*Syncbase values*

A syncbase value starts with a Syncbase-element term defining the value of an "id" attribute of another element referred to as the *syncbase element*.

*This section is normative*

A syncbase value has the following syntax:

```
Syncbase-value   ::= ( Syncbase-element "." Time-symbol )
                     ( S? ("+"|"-") S? Clock-value )?
Syncbase-element ::= Id-value
Time-symbol      ::= "begin" | "end"
```

- The syncbase element must be another timed element contained in the host document.
- The syncbase element may not be a descendent of the current element.
- If the syncbase element specification refers to an illegal element, the time-value description will be treated as though "indefinite" were specified.

The syncbase element is qualified with one of the following *time symbols:*

**begin**
Specifies the begin time of the syncbase element.
**end**
Specifies the Active End of the syncbase element.

- The time symbol can be followed by an offset value. The offset value specifies an offset from the time (i.e. the begin or active end) specified by the syncbase and time symbol.
- The offset is measured in parent simple time.

- No embedded white space is allowed between a syncbase element and a time-symbol.
- White space will be ignored before and after a "+" or "-" for a clock value.
- Leading and trailing white space characters (i.e. before and after the entire syncbase value) will be ignored.

Examples

```
begin="x.end-5s"        : Begin 5 seconds before "x" ends
begin=" x.begin "       : Begin when "x" begins
end="x.begin + 1min"    : End 1 minute after "x" begins
```

*Event values*

*This section is informative*

An Event value starts with an Eventbase-element term that specifies the *event-base element*. The event-base element is the element on which the

event is observed. Given DOM event bubbling, the event-base element may be either the element that raised the event, or it may be an ancestor element on which the bubbled event can be observed. Refer to DOM-Level2-Events [DOM2Events] for details.

*This section is normative*

An event value has the following syntax:

```
Event-value       ::= ( Eventbase-element "." )? Event-symbol
                      ( S? ("+"|"-") S? Clock-value )?
Eventbase-element ::= ID
```

The eventbase-element must be another element contained in the host document.

If the Eventbase-element term is missing, the event-base element defaults to the element on which the eventbase timing is specified (the current element).

The event value must specify an Event-symbol. This term is an XML NMTOKEN that specifies the name of the event that is raised on the Event-base element. The host language designer must specify which events can be specified.

- Host language specifications must include a description of legal event names (with "none" as a valid description), and/or allow any name to be used.
- If an integrating language specifies no supported events, the event-base time value is effectively unsupported for that language.
- A host language may choose not to include support for offsets with event values. The language must specify if this support is omitted.
- If the host language allows dynamically created events (as supported by DOM-Level2-Events [DOM2Events]), all possible Event-symbol names cannot be specified and so unrecognized names may not be considered errors.
- Unless explicitly specified by a host language, it is not considered an error to specify an event that cannot be raised on the Event-base element (such as activateEvent or click for audio or other non-visual elements). Since the event will never be raised on the specified element, the event-base value will never be resolved.

The last term specifies an optional offset-value that is an offset from the time of the event.

- The offset is measured in parent simple time. If this term is omitted, the offset is 0.
- No embedded white space is allowed between an eventbase element and an event-symbol.
- White space will be ignored before and after a "+" or "-" for a clock value.
- Leading and trailing white space characters (i.e. before and after the entire eventbase value) will be ignored.

*This section is informative*

If the eventbase element has no associated layout (e.g. a time container in a SMIL document), then some UI events may not be defined (e.g. mouse events). A host language designer may override the definition of the default eventbase element. As an example of this, the SMIL Animation elements (animate, animateMotion, etc.) specify that the default eventbase element is the *target element* of the animation. See also [[SMIL ANIMATION]].

This module defines several events that may be included in the supported set for a host language, including `beginEvent` and `endEvent`. These should not be confused with the syncbase time values. See the section on Events and event model.

The semantics of event-based timing are detailed in Unifying Scheduling and Interactive Timing. Constraints on event sensitivity are detailed in Event sensitivity.

Examples:

```
begin="  x.load  "        : Begin when "load" is observed on "x"
begin="x.focus+3s"        : Begin 3 seconds after a "focus" event on "x"
begin="x.endEvent+1.5s"   : Begin 1 and a half seconds after an "endEvent" event on "x"
begin="x.repeat"          : Begin each time a repeat event is observed on "x"
```

The following example describes a qualified repeat eventbase value:

```
<html ...>
...
<video id="foo" repeatCount="10" end="endVideo.click" ... />
<img id="endVideo" begin="foo.repeat(2)" .../>
...
</html>
```

The "endVideo" image will appear when the video "foo" repeats the second time. This example allows the user to stop the video after it has played though at least twice.

*Repeat values*

Repeat values are a variant on event values that support a qualified repeat event. The `repeat` event defined in Events and event model allows an additional suffix to qualify the event based upon an iteration value.

A repeat value has the following syntax:

```
Repeat-value      ::= ( Eventbase-element "." )? "repeat(" iteration ")"
                      ( S? ("+"|"-") S? Clock-value )?
iteration         ::= DIGIT+
```

If this qualified form is used, the eventbase value will only be resolved when a repeat is observed that has a iteration value that matches the specified iteration.

The qualified repeat event syntax allows an author to respond only to an individual repeat of an element.

*Accesskey values*

Accesskey values allow an author to tie a begin or end time to a particular key press, independent of focus issues. It is modeled on the HTML accesskey support. Unlike with HTML, user agents should not require that a modifier key (such as "ALT") be required to activate an access key.

An access key value has the following syntax:

```
Accesskey-value  ::= "accesskey(" character ")"
                     ( S? ("+"|"-") S? Clock-value )?
```

The character is a single character from [ISO10646].

The time value is defined as the time that the access key character is input by the user.

*Media marker values*

Certain types of media can have associated *marker* values that associate a name with a particular point (i.e. a time) in the media. The media marker value provides a means of defining a begin or end time in terms of these marker values. Note that if the referenced id is not associated with a media element that supports markers, or if the specified marker name is not defined by the media element, the associated time may never be resolved.

*This section is normative*

```
Media-Marker-value ::= Id-value ".marker(" S? marker-name S? ")"
```

- The marker symbol is a string that must conform to the definition of marker names for the media associated with the Id-value.

*Wallclock-sync values*

*This section is informative*

Wallclock-sync values have the following syntax. The values allowed are based upon several of the "profiles" described in [DATETIME], which is based upon [ISO8601].

*This section is normative*

```
wallclock-sync-value  ::= "wallclock(" S? (DateTime | WallTime | Date) S? ")"
DateTime        ::= Date "T" WallTime
Date            ::= Years "-" Months "-" Days
WallTime        ::= (HHMM-Time | HHMMSS-Time)(TZD)?
HHMM-Time       ::= Hours24 ":" Minutes
HHMMSS-Time     ::= Hours24 ":" Minutes ":" Seconds ("." Fraction)?
Years           ::= 4DIGIT;
Months          ::= 2DIGIT; range from 01 to 12
Days            ::= 2DIGIT; range from 01 to 31
Hours24         ::= 2DIGIT; range from 00 to 23
4DIGIT          ::= DIGIT DIGIT DIGIT DIGIT
TZD             ::= "Z" | (("+" | "-") Hours24 ":" Minutes )
```

- Exactly the components shown here must be present, with exactly this punctuation.
- Note that the "T" appears literally in the string, to indicate the beginning of the time element, as specified in [ISO8601].

*This section is informative*

```
Complete date plus hours and minutes:

   YYYY-MM-DDThh:mmTZD (e.g. 1997-07-16T19:20+01:00)

Complete date plus hours, minutes and seconds:

   YYYY-MM-DDThh:mm:ssTZD (e.g. 1997-07-16T19:20:30+01:00)

Complete date plus hours, minutes, seconds and a decimal fraction of a second

   YYYY-MM-DDThh:mm:ss.sTZD (e.g. 1997-07-16T19:20:30.45+01:00)
```

Note that the Minutes, Seconds, Fraction, 2DIGIT and DIGIT syntax is as defined for Clock-values. Note that white space is not allowed within the date and time specification.

*This section is normative*

There are three ways of handling time zone offsets:

1. Times are expressed in UTC (Coordinated Universal Time), with a special UTC designator ("Z").
2. Times are expressed in local time, together with a time zone offset in hours and minutes. A time zone offset of "+hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes ahead of UTC. A time zone offset of "-hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes behind UTC.

The presentation engine must be able to convert wallclock-values to a time within the document.

- When the document begins, the current wallclock time must be noted - this is the *document wallclock begin*.
- Wallclock values are then converted to a document time by subtracting the document wallclock begin, and then converting the time to the element's parent time space as for any syncbase value, as though the syncbase were the document body.
- Date wallclock values are treated as a DateTime value of the given date at time 00:00:00.00, that is, midnight on the given date, in the time zone given, or the local time zone if none is specified.
- WallTime values are treated as a DateTime value on the date of the *document wallclock begin* at the given time. Specified time zones must be respected, and the time converted into the local time zone before applying the *document wallclock begin*.

*This section is informative*

Note that the resulting begin or end time may be before the begin, or after end of the parent time container. This is not an error, but the time container constraints still apply. In any case, the semantics of the begin and end attribute govern the interpretation of the wallclock value.

EXAMPLES

The following examples all specify a begin at midnight on January 1st 2000, UTC

```
begin="wallclock(2000-01-01Z)"
begin="wallclock( 2000-01-01T00:00Z )"
begin="wallclock( 2000-01-01T00:00:00Z )"
begin="wallclock( 2000-01-01T00:00:00.0Z )"
begin="wallclock( 2000-01-01T00:00:00.0-00:00 )"
```

The following example specifies a begin at 3:30 in the afternoon on July 28th 1990, in the Pacific US time zone:

```
begin="wallclock( 1990-07-28T15:30-08:00 )"
```

The following example specifies a begin at 8 in the morning wherever the document is presented:

```
begin="wallclock( 08:00 )"
```

### The endsync attribute

*This section is normative*

The endsync attribute controls the implicit duration of time containers, as a function of the children. The endsync attribute is only valid for par and excl time container elements, and media elements with timed children (e.g. animate or area elements). Integrating languages may allow the endsync attribute on any element with time container semantics. The endsync attribute is particularly useful with children that have "unknown" duration, e.g. an MPEGmovie, that must be played through to determine the duration, or elements with event-based end timing.

endsync = " `first` | `last` | `all` | `media` | *`Id-value`* | *`smil1.0-Id-value`*"

Legal values for the attribute are:

`first`
: The par, excl, or media element's implicit duration ends with the earliest active end of all the child elements. This does not refer to the lexical first child, or to the first child to start, but rather refers to the first child to end its (first) active duration.

`last`
: The par, excl, or media element's implicit duration ends with the last active end of the child elements. This does not refer to the lexical last child, or to the last child to start, but rather refers to the last active end of all children that have a resolved, definite begin time. If the time container has no children with a resolved begin time, the time container ends immediately. If child elements have multiple begin times, or otherwise restart, the child elements must complete *all* instances of active durations for resolved begin times.
This is the default value for par and excl elements.

`all`
: The par, excl, or media element's implicit duration ends when all of the child elements have ended their respective active durations. Elements with indefinite or unresolved begin times *will* keep the simple duration of the time container from ending.
When all elements have completed the active duration one or more times, the parent time container can end.

`media`
: The time container element's implicit duration ends when the intrinsic media duration of the element ends. This must be defined by a host language. If the time container element does not define an intrinsic media duration, the host language must define the simple duration for the element.
This is the default value for media time container elements.

*`Id-value`*
: The par, excl, or media element time container's implicit duration ends when the specified child ends its (first) active duration. The id must correspond to one of the immediate timed children of the time container.

*`smil1.0-Id-value`*
: This syntax is deprecated.

- Elements may have an unresolved or indefinite begin time when the parent begins. If an element's unresolved begin time becomes resolved (and definite) before the parent time container ends the simple duration, the element must be considered by the `endsync="last"` semantics. This can chain, so that only one element is running at one point, but before it ends its active duration another interactive element is resolved. It may even yield "dead time" (where nothing is playing), if the resolved begin is *after* the other elements active end.
- If the endsync semantics consider any child that has an unresolved active duration, then the implicit duration of the time container is also unresolved.
- For the *`Id-value`* arg-value variant, the referenced child may have an unresolved begin time. If this causes the active end time to be unresolved as well, the implicit duration of the time container is also unresolved.
- If the endsync semantics consider any child that has a (resolved) indefinite active duration, then the implicit duration of the time container is also indefinite.
- Media element time containers define an intrinsic duration equal to the duration of the referenced media.If the referenced media is not continuous, the duration is 0 (`endsync="media"` will not generally be useful on discrete media).
- If the `media` argument value is used for an element that does not declare media, the attribute is ignored (as though endsync had not been specified).
- If the *`Id-value`* arg-value variant is not an immediate child of the time container, it is as if endsync is not specified.
- For the purpose of parsing the endsync argument value, `first`, `last`, `all`, and `media` are reserved words and must be escaped with a backslash in order to be used as `Id-value's`.

Semantics of endsync and dur and end:

- If an element specifies both endsync and dur, the endsync attribute is ignored. The element's simple duration is defined by the value of dur.
- If an element specifies both endsync and end, but none of dur, repeatDur or repeatCount, the endsync attribute is ignored. In this case the element behaves as if only end were specified, therefore the element's implicit duration is indefinite and will be constrained by the end value.

Semantics of endsync and restart:

- In the case of an element that restarts (e.g. because of multiple begin times), the element is considered to have ended its active duration when one active duration instance has completed. It is not a requirement that all instances associated with multiple begin and end times complete, to satisfy the semantics of endsync. This means that if the element is playing a second or later instance of an active duration, it may be cut short by a parent, once the other children satisfy the endsync semantics.

Semantics of endsync and paused elements:

- Note that child elements of an excl that are currently paused (by the excl semantics) have not ended their active duration. Similarly, any element paused via the DOM `pause()` method has not completed its active duration. Paused elements (that have not already completed the active duration

at least once) must be considered in the evaluation ofendsync. For example, if a time container with endsync=`last` has paused child elements, the simple duration of the time container will not end until the paused children resume or otherwise end.

*This section is informative*

Semantics of endsync and unresolved child times:

- endsync="`first`" means that the element must wait for any child element to actually end its active duration. It does not matter whether the first element to end was scheduled or interactive.
- endsync="`last`" means that the element must wait for all child elements that have a resolved begin, to end the respective active durations. Elements with indefinite or unresolved begin times will *not* keep the simple duration of the time container from ending. If there are no children with a resolved begin time, the time container will end immediately.
  Elements with a resolved begin time but indefinite or unresolved end times *will* keep the simple duration of the time container from ending.
- endsync="`all`" means that the element must wait for the end of every child element's active duration.
- endsync=[*`Id-value`*] means that the element must wait for the referenced element to actually end its active duration.

*This section is normative*

The following pseudo-code describes the endsync algorithm:

```
//
// boolean timeContainerHasEnded()
//
// method on time containers called to evaluate whether
// time container has ended, according to the rules of endsync.
// Note: Only supported on par and excl
//
// A variant on this could be called when a child end is updated to
// create a scheduled (predicted) end time for the container.
//
// Note that we never check the end time of children - it doesn't matter.
//
// Assumes:
//     child list is stable during evaluation
//     isActive state of children is up to date for current time.
//      [In practice, this means that the children must all be
//
//        If the time container is done, and repeats, the children
//        may be resampled at the modified time.]
//
//   Uses interfaces:
//   on TimedNode:
//     isActive()             tests if node is currently active
//     hasStarted()           tests if node has (ever) begun
//     begin and end          begin and end TimeValues of node
//
//   on TimeValue         (a list of times for begin or end)
//   is Resolved(t)          true if there is a resolved time
//                                  at or after time t
//

boolean timeContainerHasEnded()
{

TimeInstant now = getCurrentTime(); // normalized for time container

boolean assumedResult;

// For first or ID, we assume a false result unless we find a child that has ended
// For last and all, we assume a true result unless we find a disqualifying child

if( ( endsyncRule == first ) || ( endsyncRule == ID ) )
   assumedResult = false;
else
   assumedResult = true;

// Our interpretation of endsync == all:
//          we're done when all children have begun, and none is active
//

// loop on each child in collection of timed children,
//  and consider it in terms of the endsyncRule

 foreach ( child c in timed-children-collection )
{
   switch( endsyncRule ) {
      case first:
         // as soon as we find an ended child, return true.
         if( c.hasStarted() & !c.isActive() )
            return true;
         // else, keep looking (assumedResult is false)
         break;

      case ID:
         // if we find the matching child, just return result
                return( c.hasStarted() & !c.isActive() );
         // else, keep looking (we'll assume the ID is valid)
         break;

      case last:
         // we just test for disqualifying children
         // If the child is active, we're definitely not done.
         // If the child has not yet begun but has a resolved begin,
         // then we're not done.
         if( c.isActive()
             || c.begin.isResolved(now) )
            return false;
         // else, keep checking (the assumed result is true)
         break;

      case all:
         // we just test for disqualifying children
        // all_means_last_done_after_all_begin

         // If the child is active, we're definitely not done.
         // If the child has not yet begun then we're not done.
         // Note that if it has already begun,
         // then we still have to wait for any more resolved begins
         if( c.isActive() || !c.hasStarted()
             || c.begin.isResolved(now) )
            return false;
         // else, keep checking (the assumed result is true)
         break;

   } // close switch
```

```
} // close foreach loop

return assumedResult;

} // close timeContainerHasEnded()
```

### The repeatCount, repeatDur, and repeat attributes: repeating elements

*This section is informative*

SMIL 1.0 introduced the repeat attribute, which is used to repeat a media element or an entire time container. SMIL 2.0 introduces two new controls for repeat functionality that supercede the SMIL 1.0 repeat attribute. The new attributes, repeatCount and repeatDur, provide a semantic that more closely matches typical use-cases, and the new attributes provide more control over the duration of the repeating behavior.

Repeating an element causes the simple duration to be "played" several times in sequence. This will effectively copy or *loop* the contents of the element media (or an entire timeline in the case of a time container). The author can specify either *how many times* to repeat, using repeatCount, or *how long* to repeat, using repeatDur. Each repeat *iteration* is one instance of "playing" the simple duration.

*This section is normative*

repeatCount
: Specifies the number of iterations of the simple duration. It can have the following attribute values:

  **numeric value**
  : This is a (base 10) "floating point" numeric value that specifies the number of iterations. It can include partial iterations expressed as fraction values. A fractional value describes a portion of the simple duration. Values must be greater than 0.

  **"indefinite"**
  : The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

repeatDur
: Specifies the total duration for repeat. It can have the following attribute values:

  **Clock-value**
  : Specifies the duration in element active time to repeat the simple duration.

  **"indefinite"**
  : The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

- The SMIL 1.0 repeat attribute is deprecated in SMIL 2.0 (it must be supported in SMIL document user agents for backwards compatibility).
- See the Computing the Active Duration section for how repeat attributes interact with other timing attributes.

*Examples*

In the following example, the implicit duration of the audio is constrained by repeatCount. Only the first half of the clip will play; the active duration will be 1.5 seconds.

```
<audio src="3second_sound.au" repeatCount="0.5" />
```

In this example, the 3 second (implicit) simple duration will be played three times through and then is constrained by the dur attribute on the parent par; the active duration will be 9 seconds.

```
<par dur="9s">
   <audio src="3second_sound.au" repeatCount="100" />
</par>
```

In the following example, the 2.5 second simple duration will be repeated twice; the active duration will be 5 seconds.

```
<audio src="background.au" dur="2.5s" repeatCount="2" />
```

In the following example, the 3 second (implicit) simple duration will be repeated two full times and then the first half is repeated once more; the active duration will be 7.5 seconds.

```
<audio src="3second_sound.au" repeatCount="2.5" />
```

In the following example, the audio will repeat for a total of 7 seconds. It will play fully two times, followed by a fractional part of 2 seconds. This is equivalent to a repeatCount of 2.8.

```
<audio src="music.mp3" dur="2.5s" repeatDur="7s" />
```

Note that if the simple duration is indefinite, repeat behavior is not defined (but repeatDur still contributes to the active duration). In the following example the simple duration is 0 and indefinite respectively, and so the repeatCount is ignored. Nevertheless, this is not considered an error. The active duration is equal to the simple duration: for the first element, the active duration is 0, and for the second element, the active duration is indefinite.

```
<img src="foo.jpg" repeatCount="2" />
<img src="bar.png" dur="indefinite" repeatCount="2" />
```

In the following example, the simple duration is 0 for the image and indefinite for the text element, and so repeat behavior is not meaningful. The active duration is 0 for the first element, however for the second element, the active duration is determined by the repeatDur value, and so is 10 seconds. The effect is that the text is shown for 10 seconds.

```
<img src="foo.jpg" repeatDur="10s" />
<text src="intro.html" dur="indefinite" repeatDur="10s" />
```

In the following example, if the audio media is longer than the 5 second repeatDur, then the active duration will effectively cut short the simple duration.

```
<audio src="8second_sound.au" repeatDur="5s" />
```

The repeatCount and repeatDur attributes can also be used to repeat an entire timeline (i.e. a time container simple duration), as in the following images 3 times. The active duration is thus 39 seconds long.

```
<seq begin="5s" repeatCount="3" >
   <img src="img1.jpg" dur="5s" />
   <img src="img2.jpg" dur="4s" />
   <img src="img3.jpg" dur="4s" />
</seq>
```

*The min attribute and restart:*

The min attribute does not prevent an element from restarting before the minimum active duration is reached. If in the following example, the "user.activateEvent" occurs once at 2 seconds, then again at 5 seconds, the "image" element will begin at 2 seconds, play for 3 seconds, and then be restarted at 5 seconds. The restarted interval (beginning at 5 seconds) will display the image until 12 seconds.

```
<smil ...>
...
<par>
   <img id="image" begin="user.activateEvent" min="7s" dur="5s"
        restart="always" fill="freeze"...>
</par>
...
</smil>
```

*SMIL 1.0 repeat (deprecated)*

The SMIL 1.0 repeat attribute behaves in a manner similar to repeatCount, but it defines the functionality in terms of a sequence that contains the specified number of *copies* of the element without the repeat attribute. This definition has caused some confusion among authors and implementers. See also the SMIL 1.0 specification [SMIL10].

In particular, there has been confusion concerning the behavior of the SMIL 1.0 end attribute when used in conjunction with the repeat attribute. SMIL 2.0 complies with the common practice of having the end attribute define the element's simple duration when the deprecated repeat attribute is used. Only SMIL document user agents must support this semantic for the end attribute. Only a single SMIL 1.0 "end" value (i.e. an offset-value or a smil-1.0-syncbase-value, but none of the new SMIL 2.0 timing) is permitted when used with the deprecated repeat attribute. If repeat is used with repeatCount or repeatDur on an element, or if repeat is used with an illegal end value, the repeat value is ignored.

*This section is normative*

repeat
: *This attribute has been deprecated in SMIL 2.0 in favor of the new repeatCount and repeatDur attributes.*
This causes the element to play repeatedly for the specified number of times. It is equivalent to a seq element with the stated number of copies of the element without the "repeat" attribute as children. All other attributes of the element, including any begin delay, are included in the copies.
Legal values are `integer iterations, greater than 0,` and `"indefinite"`.

**The fill attribute: extending an element**

*This section is informative*

When an element's active duration ends, it may be *frozen* at the final state, or it may no longer be presented (i.e., its effect is removed from the presentation). *Freezing* an element extends it, using the final state defined in the last instance of the simple duration. This can be used to fill gaps in a presentation, or to extend an element as context in the presentation (e.g. with additive animation - see [SMIL-ANIMATION]).

*This section is normative*

The fill attribute allows an author to specify that an element should be extended beyond the active duration by *freezing* the final state of the element. The fill attribute is also used to determine the behavior when the active duration is less than the duration specified in the min attribute. For this reason, rather than referring to the end of the active duration, this description refers to the "last instance of the simple duration".

- For discrete media, the media is simply displayed as it would be during the simple duration.
- For visual continuous media, the "frame" that corresponds to the end of the last instance of the simple duration is shown.
- For algorithmic media like animation, the value defined for the end of the last instance of the simple duration should be used.
- For time containers, freezing extends the state of all children that are active or frozen at the end of the last instance of the time container simple duration. The children are frozen as they appear at the end of the last instance of the time container simple duration. If a child element ends its active duration coincident to the end of the last instance of its parent time container simple duration, the child element fill value determines whether the child will be frozen after the end of the parent time container's last simple duration.
- A host language integrating Timing must specify the semantics of freezing elements.

The syntax of the fill attribute is the same as in SMIL 1.0, with two extensions. In addition, the fill attribute may now be applied to any timed element, including time containers.

fill = `( remove | freeze | hold | transition | auto | default )`
: This attribute can have the following values:

`remove`
: Specifies that the element will not extend past the end of the last instance of the simple duration.

`freeze`
: Specifies that the element will extend past the end of the last instance of the simple duration by "freezing" the element state at that point. The parent time container of the element determines how long the element is frozen (as described immediately below).

`hold`
: Setting this to `"hold"` has the same effect as setting to `"freeze"`, except that the element is always frozen to extend to the *end of the simple duration of the parent time container* of theelement (independent of the type of time container). For profiles that support a layered layout model (e.g., SMIL 2.0 Language Profile), held elements (elements with fill=`"hold"`) will refresh their display area when a layer is added on top then later removed.

`transition`
: Setting this to `"transition"` has the same effect as setting to `"freeze"`, except that the element is removed at the end of the transition. This value is only allowed on elements with media directly associated with them. If specified on any other element (e.g. a time container element in the SMIL language profile), the attribute is ignored. See the SMIL Transitions module.

`auto`
: The fill behavior for this element depends on whether the element specifies any of the attributes that define the simple or active duration:

  - If none of the attributes dur, end, repeatCount or repeatDur are specified on the element, then the element will have a fill behavior identical to that if it were specified as `"freeze"`.
  - Otherwise, the element will have a fill behavior identical to that if it were specified as `"remove"`.

`default`
: The fill behavior for the element is determined by the value of the fillDefault attribute.

This is the default value.
If the application of fillDefault to an element would result in the element having a value of fill that is not allowed on that element, the element will instead have a fill value of "auto".

Note that given the default values for fill and fillDefault attributes, if the fill attribute is not specified for an element, and if the fillDefault attribute is not specified for any ascendant of the element, the behavior uses `"auto"` semantics.

An element with `"freeze" behavior` is extended according to the parent time container:

- In a par, the element is frozen to extend to the end of the simple duration of the par. In this case, fill="freeze" is equivalent to fill="hold".
- In a seq, the element is frozen to extend to the begin of the next element in the seq, or until the end of the simple duration of the seq. This will fill any gap in the presentation (although it may have no effect if the next element begins immediately).
- In an excl, the element is frozen to extend to the begin of the next element to be activated in the excl, or until an element in the excl is resumed, or until the end of the simple duration of the excl. This will fill any gap in the presentation (although it may have no effect if the next element interrupts the current element). Note that if an element is paused, the active duration has not ended, and so the fill attribute does not (yet) apply. See also the section The excl element.

When applied to media, fill only has a presentation effect on visual media. Non-visual media (audio) will simply be silent (although they are still frozen from a timing perspective).

*The fillDefault attribute*

*This section is normative*

fillDefault = `( remove | freeze | hold | transition | auto | inherit )`
Defines the default value for the fill behavior for an element and all descendents.
The values "remove", "freeze", "hold", "transition" and "auto" specify that the element fill behavior is the respective value.

`inherit`
Specifies that the value of this attribute (and of the fill behavior) are inherited from the fillDefault value of the parent element. If there is no parent element, the value is "auto".
This is the default value.

*The Event sensitivity and fill*

The effects of the fill attribute apply only to the timing semantics. If an element is still visible while frozen, it behaves normally with respect to other semantics such as user event processing. In particular, elements such as a and area are still sensitive to user activation (e.g. clicks) when frozen. See also the SMIL 1.0 specification [SMIL10].

*This section is informative*

The fill attribute can be used to maintain the value of an media element after the active duration of the element ends:

```
<par endsync="last" >
   <video src="intro.mpg" begin="5s" dur="30s" fill="freeze" />
   <audio src="intro.au"  begin="2s" dur="40s"/>
</par>
```

The video element ends 35 seconds after the parent time container began, but the video frame at 30 seconds into the media remains displayed until the audio element ends. The attribute "freezes" the last value of the element for the remainder of the time container's simple duration.

This functionality is also useful to keep prior elements on the screen while the next item of a seq time container prepares to display as in this example:

```
<seq>
   <video id="v1" fill="freeze" src.../>
   <video id="v2" begin="2s" src.../>
</seq>
```

The first video is displayed and then the last frame is frozen for 2 seconds, until the next element begins. Note that if it takes additional time to download or buffer video "v2" for playback, the first video "v1" will remain frozen until video "v2" actually begins.

**The restart attribute**

*This section is informative*

Note that there are several ways that an element may be restarted. The behavior (i.e. to restart or not) in all cases is controlled by the restart attribute. The different restart cases are:

- An element with begin specified as an event-value can be restarted when the named event fires multiple times.
- An element with begin specified as a syncbase value, where the syncbase element can restart. When an element restarts, other elements defined to begin relative to the begin or active end of the restarting element may also restart (subject to the value of restart on these elements).
- An element can be restarted when the DOM "beginElement()" method is called repeatedly.

As with any begin time, if an element is scheduled to restart after the end of the parent time container simple duration, the element will not restart.

*This section is normative*

restart = `"always | whenNotActive | never | default"`

`always`
The element can be restarted at any time.
`whenNotActive`
The element can only be restarted when it is not active (i.e. it *can* be restarted after the active end). Attempts to restart the element 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.
`default`
The restart behavior for the element is determined by the value of the restartDefault attribute.
This is the default value.

- When an element restarts, the primary semantic is that it behaves as though this were the first time the element had begun, independent of any earlier behavior. Any effect of an element playing earlier is no longer applied (including any fill behavior), and only the new current interval of the element is reflected in the presentation. It should be obvious that this definition applies only to the behavior of the element content, and not to the evaluation of the begin and end times lists described in Evaluation of begin and end time lists.
- When an active element restarts, the element first ends the active duration, propagates this to time dependents and raises an endEvent in the normal manner (see also Evaluation of begin and end time lists). Restart semantics are evaluated *after* the active duration for an element is computed, and so ending the active duration due to a restart is not subject to the semantics of min. See also Computing the active duration.
- The synchronization relationship between an element and its parent time container is re-established when the element restarts. A new synchronization relationship may be defined. See also Controlling runtime synchronization behavior.
- Note that if the parent time container (or any ascendant time container) repeats or restarts, any state associated with restart="never" will be reset, and the element can begin again normally. See also Resetting element state.

The restartDefault attribute can be used to control the default behavior of the restart attribute. This is described below in Controlling the default behavior of restart.

For details on when and how the restart attribute is evaluated, see Evaluation of begin and end time lists.

*Using restart for toggle activation*

*This section is informative*

A common use-case requires that the same UI event is used begin an element and to end the active duration of the element. This is sometimes described as "toggle" activation, because the UI event toggles the element "on" and "off". The restart attribute can be used to author this, as follows:

```
<smil ...>
...
<img id="foo" begin="bar.activateEvent" end="bar.activateEvent"
             restart="whenNotActive" ... />
</smil ...>
```

If "foo" were defined with the default restart behavior "always", a second activateEvent on the "bar" element would simply restart the element. However, since the second activateEvent cannot restart the element when restart is set to "whenNotActive", the element ignores the "begin" specification of the activateEvent event. The element can then use the activateEvent event to end the active duration and stop the element.

Note that in SMIL Language documents, a SMIL element cannot be visible before it begins so having a begin="activateEvent" means it won't ever begin. In languages with timeAction support, this may not be the case. For example, the following is reasonable:

```
<html ...>
...
<span begin="click" end="click" timeAction="class:highlight" restart="whenNotActive">
  Click here to highlight. Click again to remove highlight.
</span>
...
</html>
```

This is based upon the event sensitivity semantics described in Event sensitivity and Unifying Scheduling and Interactive Timing.

*Controlling the default behavior of restart*

*This section is normative*

The following attribute is provided to specify the default behavior for restart:

restartDefault = "always | whenNotActive | never | inherit"
: Defines the default value for the restart behavior for an element.
The values "always", "whenNotActive" and "never" specify that the element restart behavior is the respective value.

inherit
: Specifies that the value of this attribute (and of the restart behavior) are inherited from the restartDefault value of the parent element. If there is no parent element, the value is "always".
This is the default value.

Given the default values of this attribute ("inherit") and of the restart attribute ("default"), a document that does not specify these attributes will have restart="always" behavior for all timed elements.

*Resetting element state*

*This section is normative*

When a time container repeats or restarts, all descendent children are "reset" with respect to certain state:

1. Any instance times associated with past event-values, repeat-values, accesskey-values or added via DOM method calls are removed from the dependent begin and end instance times lists. In effect, all events and DOM methods calls in the past are cleared. This does not apply to an instance time that defines the begin of the current interval.
2. Any syncbase times are reevaluated (i.e. the translation between timespaces must be recalculated - see Converting between local and global times).
3. A resolved syncbase time is removed from the dependent instance time list when a common ascendant of the syncbase *and* the dependent element restarts or repeats
4. Any state associated with the interpretation of the restart semantics is reset.

*This section is informative*

Thus, for example if an element specifies restart="never", the element can begin again after a reset. The restart="never" setting is only defined for the extent of the parent time container simple duration.

*This section is normative*

When an element restarts, rules 1 and 2 are also applied to the element itself, although rule 4 (controlling restart behavior) is not applied.

When an element restarts, rules 1 and 2 are also applied to the element itself, although rule 4 (controlling restart behavior) is not applied.

Note that when any time container ends its simple duration (including when it repeats), all timed children that are still active are ended. See also Time container constraints on child durations.

When an excl time container restarts or repeats, in addition to ending any active children, the pause queue for the excl is cleared.

### The syncBehavior, syncTolerance, and syncMaster attributes: controlling runtime synchronization

*This section is informative*

New support in SMIL 2.0 introduces finer grained control over the runtime synchronization behavior of a document. The syncBehavior attribute allows an author to describe for each element whether it must remain in a hard sync relationship to the parent time container, or whether it can be allowed slip with respect to the time container. Thus, if network congestion delays or interrupts the delivery of media for an element, the syncBehavior attribute controls whether the media element can slip while the rest of the document continues to play, or whether the time container must also wait until the media delivery catches up.

The syncBehavior attribute can also be applied to time containers. This controls the sync relationship of the entire timeline defined by the time container. In this example, the audio and video elements are defined with hard or "locked" sync to maintain lip sync, but the "speech" par time container is allowed to slip:

```
<par>
  <animation src="..." />
  ...
  <par id="speech" syncBehavior="canSlip" >
    <video src="speech.mpg" syncBehavior="locked" />
    <audio src="speech.au"  syncBehavior="locked" />
  </par>
  ...
</par>
```

If either the video or audio must pause due to delivery problems, the entire "speech" par will pause, to keep the entire timeline in sync. However, the rest of the document, including the animation element will continue to play normally. Using the syncBehavior attribute on elements and time containers, the author can effectively describe the "scope" of runtime sync behavior, defining some portions of the document to play in hard sync without requiring that the entire document use hard synchronization.

This functionality also applies when an element first begins, and the media must begin to play. If the media is not yet ready (e.g. if an image file has not yet downloaded), the syncBehavior attribute controls whether the time container must wait until the element media is ready, or whether the element begin can slip until the media is downloaded.

An additional extension allows the author to specify that a particular element should define or control the synchronization for a time container. This is similar to the default behavior of many user agents that "slave" video and other elements to audio, to accommodate the audio hardware inaccuracies and the sensitivity of listeners to interruptions in the audio playback. The syncMaster attribute allows an author to explicitly define that an element defines the playback "clock" for the time container, and all other elements should be held in sync relative to the syncMaster element.

In practice, linear media often need to be the syncMaster, where non-linear media can more easily be adjusted to maintain hard sync. However, a user agent cannot always determine which media behaves in a linear fashion and which media behaves in a non-linear fashion. In addition, when there are multiple linear elements active at a given point in time, the user agent cannot always make the "right" decision to resolve sync conflicts. The syncMaster attribute allows the author to specify the element that has linear media, or that is "most important" and should not be compromised by the syncBehavior of other elements.

*This section is normative*

syncBehavior = ( `canSlip` | `locked` | `independent` | `default` )
: Defines the runtime synchronization behavior for an element.
Legal values are:

`canSlip`
: Allows the associated element to slip with respect to the parent time container.
When this value is used, any syncTolerance attribute is ignored.

`locked`
: Forces the associated element to maintain sync with respect to the parent time container. This can be eased with the use of the syncTolerance attribute.

`independent`
: Declares an independent timeline that is scheduled with the timegraph, but will ignore any seek operations on the parent.

`default`
: The runtime synchronization behavior for the element is determined by the value of the syncBehaviorDefault attribute.
This is the default value.

The argument value `independent` is equivalent to setting syncBehavior="`canSlip`" and syncMaster="`true`" so that the element is scheduled within the timegraph, but is unaffected by any other runtime synchronization issues. Setting syncBehavior="`canSlip`" and syncMaster="`true` declares the element as being the synchronization master clock and that the element may slip against its parent time line

syncTolerance = ( `Clock-value` | `default` )
: This attribute on timed elements and time containers defines the synchronization tolerance for the associated element. The attribute has an effect only if the element's runtime synchronization behavior is "`locked`". This allows a locked sync relationship to ignore a given amount of slew without forcing resynchronization.

`Clock-value`
: Specifies the synchronization tolerance as a value. Clock values are measured in element simple time.

`default`
: The synchronization tolerance for the element is determined by the value of the syncToleranceDefault attribute.
This is the default value.

syncMaster
: Boolean attribute on media elements and time containers that forces other elements in the time container to synchronize their playback to this element.
The default value is `false`.
The associated property is read-only, and cannot be set by script.

- The syncBehavior can affect the effective begin and effective end of an element, but the use of the syncBehavior attribute does not introduce any other semantics with respect to duration.
- When the syncBehavior attribute is combined with interactive begin timing or restarting an element, the syncBehavior only applies once the sync relationship of the element isresolved (e.g. when the specified event is raised). If at that point the media is not ready and syncBehavior is specified as `"locked"`, then the parent time container must wait until the media is ready. Once an element with an interactive begin time has begun playing, the syncBehavior semantics described above apply as though the element were defined with scheduled timing.
- The syncBehavior attribute is subordinate to any sync relationships defined by time containers, sync arcs, event arcs, restart behavior, etc. The syncBehavior attribute has no bearing on the formation of the time graph, only the enforcement of it.
- The syncMaster attribute interacts with the syncBehaviorattribute. An element with syncMaster set to true will define sync for the "scope" of the time container's synchronizationbehavior. That is, if the syncMaster element's parent time container has syncBehavior=`"locked"`, the syncMaster will also define sync for the ancestor timeContainer. The syncMaster will define sync for everything within the closest ancestor time container that is defined with syncBehavior=`"canSlip"`.
- The syncMaster attribute only applies when an element is active and not paused. If more than one element within the syncBehavior scope has the syncMaster attribute set to `"true"`, and the elements are both active and not paused at any moment in time, the sync master element isthe first of these elements encountered in a post order traversal of the document tree as defined by DOM [DOM2]. In this conflict case, the other elements effectively ignore the syncMaster attribute.
- When an element is paused, the semantics of the syncMaster attribute are effectively ignored. Nevertheless, any accumulated synchronization offset associated with syncMaster semantics (that is an offset accumulated while the sync master element was not paused) is not changed when the sync master element pauses. The offset in this case will be the offset for the closest ancestor time container that is defined with syncBehavior=`"canSlip"`. `See also` The accumulated synchronization offset.

*This section is informative*

Note that the semantics of syncBehavior do not describe or require a particular approach to maintaining sync; the approach will be implementation dependent. Possible means of resolving a sync conflict may include:

- Pausing the parent time container (i.e. first ancestor time container with `canSlip` behavior) until the element that slipped can "catch up".
- Pausing the element that is playing too fast until the parent (document) time container catches up.
- Seeking (i.e. resetting the current position of) the element that slipped, jumping it ahead so that it "catches up" with the parent time container. This would only apply to non-linear media types.

Additional control is provided over the hard sync model using the syncTolerance attribute. This specifies the amount of slip that can be ignored for an element. Small variance in media playback (e.g. due to hardware inaccuracies) can often be ignored, and allow the overall performance to appear smoother.

When any element is paused (including the cases described above for runtime sync behavior), the computed end time for the element may change or even become resolved, and the time model must reflect this. This is detailed in Paused elements and the active duration.

*Controlling the default behavior*

Two attributes are defined to specify the default behavior for runtime synchronization:

syncBehaviorDefault = ( `canSlip` | `locked` | `independent` | `inherit` )
Defines the default value for the runtime synchronization behavior for an element.
The values "canSlip", "locked" and "independent" specify that the element's runtime synchronization behavior is the respective value.

`inherit`
Specifies that the value of this attribute (and the value of the element's runtime synchronization behavior) are inherited from the syncBehaviorDefault value of the parent element. If there is no parent element, the value is implementation dependent.
This is the default value.

syncToleranceDefault = ( `Clock-value` | `inherit` )
Defines the default value for the runtime synchronization tolerance value for an element.
Clock values specify that the element's runtime synchronization tolerance value is the respective value.

`inherit`
Specifies that the value of this attribute (and the value of the element's runtime synchronization tolerance value) are inherited from the syncToleranceDefault value of the parent element. If there is no parent element, the value is implementation dependent but should be no greater than two seconds.
This is the default value.

*The accumulated synchronization offset*

If an element slips synchronization relative to its parent, the amount of this slip at any point is described as the *accumulated synchronization offset*. This offset is used to account for pause semantics as well as performance or delivery related slip. This value is used to adjust the conversion between element and parent times, as described in Converting between local and global times. The offset is computed as follows:

Let $\mathbf{t_c(t_{ps})}$ be the computed element active time for an element at the parent simple time $\mathbf{t_{ps}}$, according to the defined synchronization relationship for the element.

Let $\mathbf{t_o(t_{ps})}$ be the observed element active time for an element at the parent simple time $\mathbf{t_{ps}}$.

The accumulated synchronization offset **O** is:

$$\mathbf{O = t_o(t_{ps}) - t_c(t_{ps})}$$

This offset is measured in parent simple time.

Thus an accumulated synchronization offset of 1 second corresponds to the element playing 1 second "later" than it was scheduled. An offset of -0.5 seconds corresponds to the element playing a half second "ahead" of where it should be.

**Attributes for timing integration: timeContainer and timeAction**

The modularization of SMIL 2.0 functionality allows language designers to integrate SMIL Timing and Synchronization support into any XML language. In addition to just scheduling media elements as in SMIL language documents, timing can be applied to the elements of the host language. For example, the addition of timing to HTML (i.e. XHTML) elements will control the presentation of the HTML document over time, and to synchronize text and presentation with continuous media such as audio and video.

Two attributes are introduced to support these integration cases. The timeContainer attribute allows the author to specify that any XML language element has time container behavior. E.g., an HTML `<ol>` ordered list element can be defined to behave as a sequence time container. The timeAction attribute allows the author to specify what it means to apply timing to a given element.

*The timeContainer attribute*

XML language elements can be declared to have time container semantics by adding the timeContainer attribute. The syntax is:

timeContainer = ( par | seq | excl | none )

`par`
: Defines a parallel time container.

`seq`
: Defines a sequence time container.

`excl`
: Defines an exclusive time container.

`none`
: Defines the current element to *not* have time container behavior (i.e. to behave as a simple time leaf).
**This is the default.**

Constraints upon the use of the timeContainer attribute are:

- Language designers may restrict the set of elements that can be time containers, but any element can in principle be a time container.
- An element with the timeContainer attribute behaves the same as a media time container. If the element has no intrinsic media duration, then the element can behave the same as the respective time container element. Language designers must specify which elements have intrinsic media duration, and how this is defined.
- The timeContainer attribute *may not* be applied to any of the SMIL time container elements par, seq or excl.
- The timeContainer attribute *may* be applied to SMIL media elements (with timed children) to control the behavior of a media time container.

**The timeAction attribute**

The timeAction attribute provides control over the effect of timing upon an attribute. A host language must specify which values are allowed for each host language may specify additional timeAction values. The syntax is:

timeAction = ( intrinsic | display | visibility | style | class | none )

`intrinsic`
: Specifies that timing controls the intrinsic behavior of the element.
**This is the default.**

`display`
: Specifies that timing controls the display of the element, as defined by CSS. The timing of the element can affect the presentation layout. For languages that incorporate CSS, the CSS "`display`" property should be controlled over time.

`visibility`
: Specifies that timing controls the visibility of the element, as defined by CSS. The timing of the element should not affect the presentation layout. For languages that incorporate CSS, the CSS "`visibility`" property should be controlled over time.

`style`
: Specifies that timing controls the application of style defined by an inline "`style`" attribute.

`class:`*`classname`*
: Specifies that timing controls the inclusion of the specified *class-name* in the set of classes associated with the element (i.e. the XML `class` attribute value list).

`none`
: Specifies that timing has no effect upon the presentation of the element.

The `intrinsic` behavior is defined by a host language. For example in the SMIL language, the `intrinsic` behavior of media elements is to schedule and control the visibility of the media. For some elements or some languages, the `intrinsic` behavior may default to one of the other behaviors.

Additional timeAction semantics and constraints:

- SMIL media elements define the `intrinsic` behavior as the scheduling and playback of the media. When timeAction is set to any other value (besides `intrinsic`), the `intrinsic` scheduling behavior will be controlled *in addition to* the specified timeAction.
- SMIL time container elements (par, seq, and excl) define the `intrinsic` behavior as simply performing the scheduling semantics. When the `intrinsic` behavior is specified, only the scheduling semantics are controlled. Timing does not otherwise control the presentation styling of the element. When timeAction is set to any other value (besides `intrinsic`), the `intrinsic` scheduling behavior will be controlled *in addition to* the specified timeAction.
- It is recommended that phrasal, presentation, and style-like elements (e.g., XHTML's b, em, strong, ...etc.) have an `intrinsic` behavior that applies the associated semantic (e.g. to *embolden* or *emphasize* the content of the element). When timeAction is set to any other value (besides `intrinsic`), the `intrinsic` behavior will be controlled *in addition to* the specified timeAction.
- It is recommended that "content" elements (e.g., XHTML's p, div, span) have an intrinsic behavior equivalent to `visibility`. When timeAction is set to any other value (besides `intrinsic`), the `intrinsic` behavior should *not* be controlled, but rather only the specified timeAction should be applied.
- If a language supports CSS styling, the presentation effect of `display` and `visibility` should use the CSS override style, rather than setting the original value for the associated properties. This model for presentation values is described in the SMIL Animation module.
- If a language supports CSS styling, the `visibility` property should be set to "`hidden`" when the element is not active or frozen. If the original value of the `visibility` property was not "`hidden`", the original value should be used when the element is active or frozen. If the original value of the `visibility` property was "`hidden`", the property should be set to "`visible`" when the element is active or frozen.
- If a language does not support CSS styling, the presentation semantics of `display` and `visibility` must be specified, or the attribute values must be disallowed.
- If a language does not support a "`style`" attribute, the `style` value for timeAction should not be allowed.
- When the `class` argument value is specified, the specified class name should be *added to* the class list of the element when the element is active or frozen. Other values in the class list must be preserved. If the specified class name is specified in the class list (i.e. if it is specified in the `class` attribute), the class name should be removed from the class list of the element when the element is not active or frozen.

attribute), the class name should be removed from the class list of the element when the element is not active or frozen.

- When the `none` argument value is specified, no action is controlled by the timing, except any intrinsic behavior. This value is generally only useful on an element that also specifies a timeContainer value. In this case, the time containment semantic applies, but no presentation effect is applied to the element.

Certain special elements may have specific `intrinsic` semantics. For example, linking elements like a and area can have an `intrinsic` behavior that controls the sensitivity of the elements to actuation by the user. This may have presentation side-effects as well. In XHTML for example, making these elements insensitive also has the effect that the default styling (e.g. a color and underline) that is applied to sensitive links is removed when the element is not active or frozen.

Host language designers should carefully consider and define the behavior associated with applying timing to an element. For example, `script` elements could be defined to execute when the element begins, or the language could disallow the timeAction attribute on the element. Similarly, `link` elements could apply a linked stylesheet when the element begins or the language could disallow the timeAction attribute on `link`.

For details of the CSS properties `visibility` and `display`, see [CSS2].

**Examples:**

The following example shows a simple case of controlling visibility over time. The text is hidden from 0 to 3 seconds, shown normally for 5 seconds, and then hidden again.

```
<span timeAction="visibility" begin="3s" dur="5s">
   Show this text for a short period.
</span>
```

The following example shows a simple case of controlling display over time. Each list element is shown for 5 seconds, and is removed from the layout when not active or frozen. The ordered list element is set to be a sequence time container as well (note that each list element retains its ordinal number even though the others are not displayed):

```
<ol timeContainer="seq" repeatDur="indefinite">
   <li timeAction="display" dur="5s">
      This is the first thing you will see. </li>
   <li timeAction="display" dur="5s">
      You will see this second. </li>
   <li timeAction="display" dur="5s">
      Last but not least, you will see this. </li>
</ol>
```

The following example shows how an element specific style can be applied over time. The respective style is applied to each HTML `label` for 5 seconds after a focus event is raised on the element:

```
<form ...>
...
   <label for="select_red" begin="focus" dur="5s" timeAction="style"
         style="color:red; font-weight:bold" >
      Make things RED.
   </label>
   <input id="select_red" .../>
   <label for="select_green" begin="focus" dur="5s" timeAction="style"
         style="color:green; font-weight:bold" >
      Make things GREEN.
   </label>
   <input id="select_green" .../>
...
</form>
```

### 10.3.2 Elements

*This section is informative*

SMIL 2.0 specifies three types of time containers. These can be declared with the elements par, seq, and excl, or in some integration profiles with a timeContainer attribute. Media elements with timed children are defined to be "media time containers", and have semantics based upon the par

This document refers in general to time containers by reference to the elements, but the same semantics apply when declared with an attribute, and for media time containers.

**The par element**

*This section is normative*

par
: A par container, short for "parallel", defines a simple time grouping in which multiple elements can play back at the same time.

The implicit syncbase of the child elements of a par is the begin of the par. The default value of begin for children of a par is `"0"`. This is the same element introduced with SMIL 1.0.

The par element supports all element timing.

*Implicit duration of par*

The implicit duration of a par is controlled by endsync. By default, the implicit duration of a par is defined by the endsync=`"last"` semantics. The implicit duration ends with the last active end of the child elements.

**The seq element**

*This section is normative*

seq
: A seq container defines a sequence of elements in which elements play one after the other.

This is the same element introduced with SMIL 1.0, but the semantics (and allowed syntax) for child elements of a seq are clarified.

- The implicit syncbase of the child elements of a seq is the active end of the *previous* element. *Previous* means the element that occurs before this element in the sequence time container. For the first child of a sequence (i.e. where no previous sibling exists), the implicit syncbase is the
- For children of a sequence, the only legal value for begin is a (single) non-negative offset value. The default begin value is "0". None of the following begin values may be used:

  **disallowed begin-values:**

  ( `syncbase-value` | `event-value` | `media-marker-value` | `wallclock-sync-value` | `repeat-value` | `accesskey-value` | `"indefinite"` )

- Note however that child elements *may* define an end that references other syncbases, event-bases, etc.

The seq element itself supports all element timing except endsync.

When a hyperlink traversal targets a child of a seq, and the target child is not currently active, part of the seek action must be to enforce the basic semantic of a seq that only one child may be active at a given time. For details, see Hyperlinks and timing and specifically Implications of beginElement() and hyperlinking for seq and excl time containers.

*Implicit duration of seq containers*

- The implicit duration of a seq ends with the active end of the last child of the seq.
- If any child of a seq has an indefinite active duration, the implicit duration of the seq is also indefinite.

**The excl element**

SMIL 2.0 defines a new time container, excl.

*This section is normative*

excl
: This defines a time container with semantics based upon par, but with the additional constraint that only one child element may play at any given time. If any element begins playing while another is already playing, the element that was playing is either paused or stopped. Elements in an excl are grouped into categories, and the pause/interruption behavior of each category can be controlled using the new grouping element priorityClass.

The implicit syncbase of the child elements of the excl is the begin of the excl. The default value of begin for children of excl is `"indefinite"`. This means

The excl element itself supports all element timing.

*This section is informative*

With the excl time container, common use cases that were either difficult, or impossible, to author are now easier and possible to create. The excl time container is used to define a mutually exclusive set of clips, and to describe pausing and resuming behaviors among these clips. Examples include:

**interactive playlist**
: A selection of media clips is available for the user to choose from, only one of which plays at a time. A new selection replaces the current selection.

**audio descriptions**
: For visually impaired users, the current video is paused and audio descriptions of the current scene are played. The video resumes when the audio description completes.

**interactive video sub-titles**
: Multiple language sub-titles are available for a video. Only one language version can be shown at a time with the most recent selection replacing the previous language choice, if any.

The interactive playlist use case above could be accomplished using a par whose sources have interactive begin times and end events for all other sources. This would require a prohibitively long list of values for end to maintain. The excl time container provides a convenient short hand for this - the element begin times are still interactive, but the end events do not need to be specified because the excl, by definition, only allows one child element to play at a time.

The audio descriptions use case is not possible without the pause/resume behavior provided by excl and priorityClass. This use case would be authored with a video and each audio description as children of the excl. The video element would be scheduled to begin when the excl begins and the audio descriptions, peers of the video element, would start at scheduled begin times or in response to stream events raised at specific times.

The dynamic video sub-titles use case requires the "play only one at a time" behavior of excl. In addition, the child elements are declared in such a way so as to preserve the sync relationship to the video:

```
<smil ...>
...
<par>
   <video id="vid1" .../>
   <excl>
      <audio begin="vid1.begin" src="english.au" />
      </par>
      <par begin="frenchBtn.activateEvent" >
         <audio begin="vid1.begin" src="french.au" />
      </par>
      <par begin="swahiliBtn.activateEvent" >
         <audio begin="vid1.begin" src="swahili.au" />
      </par>
   </excl>
</par>
...
</smil>
```

The three par elements are children of the excl, and so only one can play at a time. The audio child in each par is defined to begin when the video begins. Each audio can only be active when the parent time container (par) is active, but the begin still specifies the synchronization relationship. This means that when each par begins, the audio will start playing at some point in the middle of the audio clip, and in sync with the video.

The excl time container is useful in many authoring scenarios by providing a declarative means of describing complex clip interactions.

*Implicit duration of excl containers*

*This section is normative*

- The implicit duration of an excl container is defined the same as for a par container, using the endsync="last" semantics. However, since the default timing for children of excl is interactive, the implicit duration for excl time containers with only default timing on the children will be 0.

*The priorityClass element*

*This section is informative*

Using priority classes to control the pausing behavior of children of the excl allows the author to group content into categories of content, and then to describe rules for how each category will interrupt or be interrupted by other categories. Attributes of the new grouping element priorityClass describe the intended interactions.

Each priorityClass element describes a group of children, and the behavior of those children when interrupted by other time-children of the excl. The behavior is described in terms of *peers*, and *higher* and *lower* priority elements. *Peers* are those elements within the same priorityClass element.

When one element within the excl begins (or would normally begin) while another is already active, several behaviors may result. The active element may be paused or stopped, or the interrupting element may be deferred, or simply blocked from beginning.

The careful choice of defaults makes common use cases very simple. See the examples below.

*This section is normative*

priorityClass
: Defines a group of excl time-children, and the pause/interrupt behavior of the children. If a priorityClass element appears as the child of an excl, then the excl can only contain priorityClass elements (i.e. the author cannot mix timed children and priorityClass elements within an excl).

If no priorityClass element is used, all the children of the excl are considered to be *peers*, with the default peers behavior "stop".

- priorityClass elements can only be children of an excl.
- priorityClass elements can only have media elements as (timed) children. That is, priorityClass elements cannot be nested.
- The priorityClass element is transparent to timing, and does not participate in or otherwise affect the normal timing behavior of its children (i.e. it only defines how elements interrupt one another).
- Child elements of the priorityClass elements are time-children of the excl element (i.e. the parent excl of the priorityClass elements).
- The priorityClass elements are assigned priority levels based upon the order in which they are declared within the excl. The first priorityClass element has highest priority, and the last has lowest priority.
- When elements are paused or deferred, they are added to a queue of pending elements. When an active element completes its active duration, the first element (if any) in the queue of pending elements is madeactive. The queue is ordered according to rules describedin Pause queue semantics.

**The peers, higher, and lower attributes**

*This section is informative*

Note that the rules define the behavior of the currently active element and the interrupting element. Any elements in the pause queue are not affected (except that their position in the queue may be altered by new queue insertions).

*This section is normative*

Controls how child elements of this priorityClass will interrupt one another.
Legal values for the attribute are:

`stop`
: If a child element begins while another child element is active, the active element is simply stopped.
**This is the default for peers.**

`pause`
: If a child element begins while another child element is active, the active element is paused and will resume when the new (interrupting) element completes its active duration (subject to the constraints of the excl time container). The paused element is added to the pause queue.

`defer`
: If a child element attempts to (i.e. would normally) begin while another child element is active, the new (interrupting) element is deferred until the active element completes its active duration.

`never`
: If a child element attempts to (i.e. would normally) begin while another child element is active, the new (interrupting) element is prevented from beginning. The begin of the new (interrupting) element is ignored.

higher = " `stop | pause` "
Controls how elements with higher priority will interrupt child elements of this priorityClass.
Legal values for the attribute are:

`stop`
: If a higher priority element begins while a child element of this priorityClass is active, the active child element is simply stopped.

`pause`
: If a higher priority element begins while a child element of this priorityClass is active, the active child element is paused and will resume when the new (interrupting) element completes its active duration (subject to the constraints of the `excl` time container). The paused element is added to the pause queue.
**This is the default for the higher attribute.**

lower = " `defer | never` "
Controls how elements defined with lower priority will interrupt child elements of this priorityClass.
Legal values for the attribute are:

`defer`
: If a lower priority element attempts to (would normally) begin while a child element of this priorityClass is active, the new (interrupting)

below.
**This is the default for the lower attribute.**

`never`
If a lower priority element attempts to begin while a child element of this priorityClass is active, the new (interrupting) element is prevented from beginning. The begin of the new (interrupting) element is ignored, and it is not added to the queue.

When an element begin is blocked (ignored) because of the "never" attribute value, the blocked element does not begin in the time model. The time model should not propagate begin or end activations to time dependents, nor should it raise begin or end events.

**The pauseDisplay attribute**

*This section is informative*

The pauseDisplay attribute controls the *behavior when paused* of the children of a priorityClass element. When a child of a priorityClass element is paused according to excl and priorityClass semantics, the pauseDisplay attribute controls whether the paused element will continue to *show* or *apply* the element (i.e. the state of the element for the time at which it is paused), or whether it is removed altogether from the presentation (i.e. *disabled*) while paused.

*This section is normative*

pauseDisplay = " `disable | hide | show` "
Controls how child elements of the priorityClass element behave when paused. This attribute only applies if peers="pause" or higher="pause".
Legal values for the attribute are:

`disable`
Continue to display visual media when the element is paused by the excl and priorityClass, but appear disabled. It is implementation dependent how a disabled element appears (rendered in some different way to distinguish from the active state -- e.g., grayed out); disabled elements do not respond to mouse events.

`hide`
Remove the effect of the element (including any rendering) when the element is paused by the excl and priorityClass semantics.

`show`
Continue to show the effect of the element (including any rendering) when the element is paused by the excl and priorityClass semantics. This value has no effect on a syncMaster.
**This is the default.**

*Examples using excl and priorityClass*

*This section is informative*

Note that because of the defaults, the simple cases work without any additional syntax. In the basic case, all the elements default to be peers, and stop one another:

```
<excl dur="indefinite">
   <audio id="song1" .../>
   <audio id="song2" .../>
   <audio id="song3" .../>
   ...
   <audio id="songN" .../>
</excl>
```

is equivalent to the following with explicit settings:

```
<excl dur="indefinite">
   <priorityClass peers="stop">
      <audio id="song1" .../>
      <audio id="song2" .../>
      <audio id="song3" .../>
      ...
      <audio id="songN" .../>
   </priorityClass>
</excl>
```

If the author wants elements to pause rather than stop, the syntax is:

```
<excl dur="indefinite">
   <priorityClass peers="pause">
      <audio id="song1" .../>
      <audio id="song2" .../>
      <audio id="song3" .../>
      ...
      <audio id="songN" .../>
   </priorityClass>
</excl>
```

The audio description use case for visually impaired users would look very similar to the previous example:

```
<excl dur="indefinite">
   <priorityClass peers="pause">
      <video id="main_video" .../>
      <audio id="scene1_description" begin="20s"  dur="30s".../>
      <audio id="scene2_description" begin="2min" dur="30s" .../>
      ...
      <audio id="sceneN_description" .../>
   </priorityClass>
</excl>
```

This example shows a more complex case of program material and several commercial insertions. The program videos will interrupt one another. The ads will pause the program, but will not interrupt one another.

```
<excl dur="indefinite">
   <priorityClass id="ads" peers="defer">
      <video id="advert1" .../>
      <video id="advert2" .../>
   </priorityClass>
   <priorityClass id="program" peers="stop" higher="pause">
      <video id="program1" .../>
      <video id="program2" .../>
      <video id="program3" .../>
      <video id="program4" .../>
```

```
    </priorityClass>
</excl>
```

The following example illustrates how defer semantics and priority groups can interact. When "alert1" tries to begin at 5 seconds, the "program" priorityClass will force "alert1" to defer, and so "alert1" will be placed upon the queue. When "alert2" tries to begin at 6 seconds, the same semantics will force "alert2" onto the queue. Note that although the "alerts" priorityClass defines the peers rule as "never", "alert1" is not active at 6 seconds, and so the interrupt semantics between "alert1" and "alert2" are not evaluated. The resulting behavior is that when "prog1" ends at 20 seconds, "alert1" will play, and then when "alert1" ends, "alert2" will play.

```
<excl dur="indefinite">
   <priorityClass id="program" lower="defer">
     <video id="prog1" begin="0" dur="20s" .../>
   </priorityClass>
   <priorityClass id="alerts" peers="never">
     <video id="alert1" begin="5s" .../>
     <video id="alert2" begin="6s" .../>
   </priorityClass>
</excl>
```

This example illustrates pauseDisplay control. When an element is interrupted by a peer, the interrupted element pauses and is shown in a disabled state. It is implementation dependent how the disabled video is rendered. Disabled elements do not respond to mouse events.

```
<excl dur="indefinite">
   <priorityClass peers="pause" pauseDisplay="disable">
      <video id="video1" .../>
      <video id="video2" .../>
      <video id="video3" .../>
      ...
      <video id="videoN" .../>
   </priorityClass>
</excl>
```

In this example, when a child of a higher priorityClass element interrupts a child of the "program" priorityClass, the child of "program" pauses and remains onscreen. If a peer of the "program" priorityClass interrupts a peer, the element that was playing stops and is no longer displayed.

```
<excl dur="indefinite">
   <priorityClass id="ads" peers="defer">
      <video id="advert1" .../>
      <video id="advert2" .../>
   </priorityClass>
   <priorityClass id="program" peers="stop" higher="pause" pauseDisplay="show">
      <video id="program1" .../>
      <video id="program2" .../>
      <video id="program3" .../>
      <video id="program4" .../>
   </priorityClass>
</excl>
```

*Pause queue semantics*

*This section is normative*

Elements that are paused or deferred are placed in a priority-sorted queue of waiting elements. When an active element ends its active duration and the queue is not empty, the first (i.e. highest priority) element in the queue is *pulled* from the queue and resumed or activated.

The queue semantics are described as a set of invariants and the rules for insertion and removal of elements. For the purposes of discussion, the child elements of a priorityClass element are considered to have the priority of that priorityClass, and to have the behavior described by the peers, higher and lower attributes on the priorityClass parent.

QUEUE INVARIANTS

1. The queue is sorted by priority, with higher priority elements before lower priority elements.
2. An element may not appear in the queue more than once.
3. An element may not simultaneously be active and in the queue.

ELEMENT INSERTION AND REMOVAL

1. Elements are inserted into the queue sorted by priority (by invariant 1).
    a. Paused elements are inserted *before* elements with the same priority,
    b. Deferred elements are inserted *after* elements with the same priority.
2. Where the semantics define that an active element must be paused, the element is paused at the current simple time (position) when placed on the queue. When a paused element is pulled normally from the queue, it will resume from the point at which it was paused.
3. Where the semantics define that an element must be deferred, the element is inserted in the queue, *but is not begun*. When the element is pulled normally from the queue, it will begin (i.e. be activated).
4. When an element is placed in the queue any previous instance of that element is removed from the queue (by invariant 2).
5. When the active child (i.e. time-child) of an excl ends normally (i.e. not when it is *stopped* by another, interrupting element), the element on the front of the queue is pulled off the queue, and resumed or begun (according to rule 2 or 3).

Note that if an element is active and restarts (subject to the restart rule), it does not interrupt itself in the sense of a peer interrupting it. Rather, it simply restarts and the queue is unaffected.

RUNTIME SYNCHRONIZATION BEHAVIOR AND PAUSE/DEFER SEMANTICS

The runtime synchronization behavior of an element (described in The syncBehavior, syncTolerance, and syncMaster attributes: controlling runtime synchronization) does not affect the queue semantics. Any element that is paused or deferred according to the queue semantics will behave as described. When a paused element is resumed, the synchronization relationship will be reestablished according to the runtime synchronization semantics. The synchronization relationship for a deferred element will be established when the element actually begins.

### Calculated times and pause/defer semantics

When an element is paused, the calculated end time for the element may change or even become resolved, and the time model must reflect this. This is detailed in Paused elements and the active duration. In some cases, the end time is defined by other elements unaffected by the pause queue semantics. In the following example, the "foo" element will be paused at 8 seconds, but it will still end at 10 seconds (while it is paused):

```
<img "joe" end="10s" .../>
<excl dur="indefinite">
   <priorityClass peers="pause">
      <img id="foo" end="joe.end" .../>
      <img id="bar" begin="8s" dur="5s" .../>
   </priorityClass>
</excl>
```

If an element ends while it is in the pause queue, it is simply removed from the pause queue. All time dependents will be notified normally, and the end event will be raised at the end time, as usual.

When an element is deferred, the begin time is deferred as well. Just as described in Paused elements and the active duration, the begin time of a deferred element may become unresolved, or it may simply be delayed. In the following example, the "bar" element will initially have an unresolved begin time. If the user clicks on "foo" at 8 seconds, "bar" would resolve to 8 seconds, but will be deferred until 10 seconds (when "foo" ends):

```
<html ...>
...
<excl dur="indefinite">
   <priorityClass peers="defer">
      <img id="foo" begin="0s" dur="10s" .../>
      <img id="bar" begin="foo.click" .../>
   </priorityClass>
</excl>
...
</html>
```

If there is enough information to determine the new begin time (as in the example above), an implementation must compute the correct begin time when an element is deferred. The change to the begin time that results from the element being paused must be propagated to any sync arc time dependents (i.e. other elements with a begin or end defined relative to the begin of the deferred element). See also the Propagating changes to times section.

One exception to normal processing is made for deferred elements, to simplify the model: a deferred element ignores propagated changes to its begin time. This is detailed in the Deferred elements and propagating changes to begin section.

### Scheduled begin times and excl

Although the default begin value for children of an excl is indefinite, scheduled begin times are permitted. Scheduled begin times on children of the excl cause the element to begin at the specified time, pausing or stopping other siblings depending on the priorityClass settings (and default values).

### Handling Simultaneous Begins within excl

If children of an excl attempt to begin at the same time, the evaluation proceeds in document order. For each element in turn, the priorityClass semantics are considered, and elements may be paused, deferred or stopped.

*This section is informative*

The following examples both exhibit this behavior (it can result from any combination of scheduled times, interactive timing, hyperlink or DOM activation):

```
<smil ...>
...
<excl>
   <img src="image1.jpg" begin="0s" dur="5s"/>
   <img src="image2.jpg" begin="0s" dur="5s"/>
   <img src="image3.jpg" begin="0s" dur="5s"/>
</excl>

<excl>
   <img id="img1" src="image1.jpg" begin="foo.activateEvent" dur="5s"/>
   <img id="img2" src="image2.jpg" begin="img1.begin" dur="5s"/>
   <img id="img3" src="image3.jpg" begin="img2.begin" dur="5s"/>
</excl>
...
</smil>
```

In the first example, the images are scheduled to begin immediately, where in the second, they will all begin once the user activates the "foo" element. The end result of the two (other than the begin time) is the same. Given the default interrupt semantics for excl, the first image will begin and then be immediately stopped by the second image, which will in turn be immediately stopped by the third image. The net result is that only the third image is seen, and it lasts for 5 seconds. Note that the begin and end events for the first two images are raised and propagated to all time dependents. If the behavior is set to "pause" as in this example, the declared order is effectively reversed:

```
<excl>
   <priorityClass peers="pause">
      <img src="image1.jpg" begin="0s" dur="5s"/>
      <img src="image2.jpg" begin="0s" dur="5s"/>
      <img src="image3.jpg" begin="0s" dur="5s"/>
   </priorityClass>
</excl>
```

In this case, the first image will begin and then be immediately paused by the second image, which will in turn be immediately paused by the third image. The net result is that the third image is seen for 5 seconds, followed by the second image for 5 seconds, followed by the first image for 5 seconds. Note that the begin events for the first two images are raised and propagated to all time dependents when the excl begins.

In the following slideshow example, images begin at the earlier of their scheduled begin time or when activated by a user input event:

```
<html ...>
...
<excl>
   <img src="image1.jpg" begin="0s"...>
   <img src="image2.jpg" begin="10s; image1.click"...>
```

```
  <img src="image3.jpg" begin="20s; image2.click" .../>
</excl>
...
</html>
```

Note, some surprising results may occur when combining scheduled and interactive timing within an excl. If in the above example, the user clicks on image1 and then on image2 before ten seconds have elapsed, image 2 will re-appear at the ten second mark. Image 3 will appear at twenty seconds. The likely intent of this particular use-case would be better represented with a seq time container.

### SIDE EFFECTS OF ACTIVATION

Children of the excl can be activated by scheduled timing, hyperlinks, events or DOM methods calls. For all but hyperlink activation, the excl time container must be active for child elements of the excl to be activated. With hyperlink activation, the document may be seeked to force the parent excl to be active, and a seek may occur to the begin time target child if it has a resolved begin time. That is, the normal hyperlink seek semantics apply to a timed child of an excl.

*This section is normative*

With activation via a DOM method call (e.g. the `beginElement()` method), the element will be activated at the current time (subject to the priorityClass semantics), even if the element has a scheduled begin time. The exclusive semantics of the time container (allowing only one active element at a time) and all priorityClass semantics are respected nevertheless.

See also Hyperlinks and timing and specifically Implications of beginElement() and hyperlinking for seq and excl time containers.

#### Implicit duration of media element time containers

*This section is normative*

The implicit duration of a media time container combines the intrinsic duration of the media with the children to define the implicit simple duration. For the "*ID-REF*" value of endsync, the semantics are the same as for a normal time container. For the "media" value of endsync, implicit simple duration is equal to the intrinsic duration of the media directly associated with the element. For the values "first", "last" and "all", the media element acts as a par time container, but treats the element's associated media as an additional condition as far as determining when the criteria for "first", "last" and "all" endsync values have been satisfied.

- For endsync={ "media" or "*ID-REF*"}: This is defined as for par elements.
- For endsync={"last" or "all"}: The time children and the intrinsic media duration of the associated media define the implicit duration of the media element time container. If the associated media duration is longer than the extent of all the time children, the media duration defines the implicit duration for the media element time container. If the associated media is discrete, this is defined as for par elements.
- For endsync="first": The time children and the intrinsic media duration define the implicit duration of the media element time container. The element ends when the first active duration ends, as defined above for endsync on a par. If the media is discrete, this is defined as for par elements.

If the implicit duration defined by endsync is *longer* than the intrinsic duration for a continuous media element, the ending state of the media (e.g. the

*This section is informative*

This semantic is similar to the case in which the author specifies a simple duration that is longer than the intrinsic duration for a continuous media element. Note that for both cases, although the media element is effectively frozen for the remainder of the simple duration, the time container simple time is not frozen during this period, and any children will run normally without being affected by the media intrinsic duration.

*Examples:*

Assume that "vid1" is 10 seconds long in the following examples.

The default value of endsync for media elements is "media", and so the simple duration in the following example is 10 seconds. This will cut short the animate child 8 seconds into its simple duration:

```
<video src="vid1.mpg" >
   <animate begin="2s" dur="12s" .../>
</video>
```

Specifying endsync="first" in the example below causes the simple duration of the video element to be 10 seconds, since the media finishes before the animate child.

```
<video src="vid1.mpg" endsync="first" >
   <animate begin="2s" dur="12s" .../>
</video>
```

Specifying endsync="last" in the following example causes the simple duration of the video element to be 14 seconds. The video will show a still frame (the last frame) for the last 4 seconds of this:

```
<video src="vid1.mpg" endsync="last" >
   <set dur="8s" .../>
   <animate begin="2s" dur="12s" .../>
</video>
```

Specifying endsync="all" in the following example causes the simple duration of the video element to last at least 10 seconds (the intrinsic duration of the video), and at most until 5 seconds after the user clicks on the video. The video will show a still frame (the last frame) for any duration in excess of 10 seconds:

```
<html ...>
...
<video src="vid1.mpg" endsync="all" >
   <set dur="8s" .../>
   <animate begin="click" dur="5s" .../>
</video>
...
</html>
```

Thus if the user clicks on the video after 1 second, the simple duration is 10 seconds. If the user does not click until 15 seconds, the simple duration is 20 seconds, and the last frame will be shown between 10 and 20 seconds. The video can still be clicked even though it stops normal play at 10 seconds.

*Media time containers of other types*

In some language integrations, it will be possible to declare a media time container to have sequence or exclusive semantics, in addition to the default parallel semantics described above. For example:

```
<html ...>
...
<video src="vid1.mpg" timeContainer="seq" endsync="first" >
   <animate dur="4s" .../>
   <animate end="click" .../>
</video>
...
</html>
```

The animate children of the video will act in sequence. The endsync semantics define a simple duration for the video that is no more than 10 seconds (the intrinsic duration of the video) but may be just over 4 seconds, if the user clicks on the video as soon as the last animate begins.

### 10.3.3 Semantics of the Timing Model

*Except as noted, this entire section is normative*

**Resolving times**

A begin or end time is said to be unresolved when either an associated begin or end event has not yet occurred (within the constraints of Event sensitivity), or the begin or end time is dependent upon another element's begin or end time that is unresolved. The begin or end time becomes resolved as soon as the syncbase element's time is resolved, or when the event occurs (within the constraints of Event sensitivity).

If a begin or end value resolves to a time in the past, this value is propagated to other synchronization dependents. Similarly, a simple or active duration can be unresolved but can become resolved when end conditions are met or the parent time container constrains the element's duration.

*Definite times*

A resolved time is said to be *definite* if it is not the value "indefinite".

**Defining the simple duration**

The *simple duration* of an element is determined by the dur attribute, the implicit duration of the element, and one special-case rule to ensure SMIL 1.0 backward compatibility. Apply the first rule in the table that matches the given criteria.

Computation of the simple duration is based on the information available at the time the calculation is made. Unresolved quantities may require the simple duration to be recomputed when an unresolved quantity becomes resolved.

| dur | implicit element duration | repeatDur and repeatCount | Simple Duration |
|---|---|---|---|
| unspecified | (ignored) | unspecified, end specified | `indefinite` |
| Clock-value | (ignored) | (ignored) | dur or Clock-value |
| `indefinite` | (ignored) | (ignored) | `indefinite` |
| unspecified | resolved | (ignored) | implicit element duration or Clock-value |
| unspecified | unresolved | (ignored) | unresolved |
| media | resolved or unresolved | (ignored) | implicit element duration |

***Simple Duration Table***

*repeatCount and unresolved simple duration*

When repeatCount is specified, it is understood to represent a count of iterations of simple duration. Each iteration of the simple duration may be different, and so a simple multiplication of the repeatCount and a given simple duration may not yield an accurate active duration. In the case of a partial repeatCount and a simple duration that is not resolved, the most recent simple duration should be multiplied by the fractional part of the repeatCount to constrain the last simple duration. If the last iteration of the simple duration otherwise ends before this time, the repeatCount should be considered to be complete. If a repeatCount is less than 1 and the simple duration is unresolved, the repeatCount cannot be correctly respected, and will behave as though a repeatCount of "1" were specified.

*This section is informative*

If an element specifying audio media has a simple duration of 0 (e.g., because of `clipBegin` and `clipEnd` values), nothing should be played even if the repeatDur specifies an active duration. The time model behaves according to the description, but no audio should be played.

If a repeatDur is shorter than the simple duration, or if repeatCount is less than 1, the active duration can cut short the defined simple duration.

If repeatDur is "indefinite" and neither of repeatCount or end are specified, the active duration is indefinite. If repeatCount is indefinite, the simple duration is greater than 0 and neither of repeatDur or end are specified, then the active duration is indefinite.

Note that unlike in SMIL 1, when an element defines a begin offset and repeat behavior with repeatCount or repeatDur, the begin offset is *not included* in each repeat.

**Computing the active duration**

The *active duration* of an element defines the entire period that an element's timeline is active. It takes into account the element *simple duration* [illegible] attributes, and any repeat behavior defined by the [illegible] attributes.

*Active duration arithmetic rules*

Computing the active duration requires defining arithmetic operations on all of the possible values that simple duration can have.

MULTIPLICATION

- zero value * value = zero value
- zero value * indefinite = zero value
- non-zero value * non-zero value = non-zero value
- non-zero value * indefinite = indefinite
- indefinite * indefinite = indefinite
- unresolved * *anything* = unresolved

ADDITION AND SUBTRACTION

- value +/- value = value
- indefinite +/- value = indefinite
- value +/- indefinite = indefinite
- indefinite +/- indefinite = indefinite
- unresolved +/- *anything* = unresolved
- *anything* +/- unresolved = unresolved

MINIMIZATION FUNCTION

- MIN( non-zero value, non-zero value) = non-zero value
- MIN( non-zero value, indefinite) = non-zero value
- MIN( non-zero value, unresolved) = non-zero value
- MIN( indefinite, unresolved) = indefinite

Where *anything* means zero value, non-zero value, indefinite, or unresolved.

MAXIMIZATION FUNCTION

- MAX( numeric value A, numeric value B) = B if B > A, otherwise A
- MAX( numeric value, indefinite) = indefinite
- MAX( numeric value, unresolved) = unresolved
- MAX( indefinite, unresolved) = unresolved

*Active duration algorithm*

*This section is informative*

In this section, references to begin and end values should be understood as the current effective values in each respective value list. These values are determined by the rules described in Evaluation of begin and end time lists.

*This section is normative*

The following symbols are used in the algorithm as a shorthand:

**B**
: The begin of an element.

**d**
: The simple duration of an element.

**PAD**
: [illegible]

**AD**
: The active duration of an element.

Computation of the active duration is based on the information available at the time the calculation is made. Unresolved quantities may require the active duration to be recomputed when an unresolved quantity becomes resolved.

To compute the active duration, use the following algorithm:

If end is specified, and none of dur, repeatDur, and repeatCount are specified, then the simple duration is `indefinite` from the simple duration table above, and the active duration is defined by the end value, according to the following cases:

If end is resolved to a value, then **PAD** = end - **B**,

else, if end is `indefinite`, then **PAD** = `indefinite`,

else, if end is unresolved, then **PAD** is unresolved, and needs to be recomputed when more information becomes available.

Else, if no end value is specified, or the end value is specified as `indefinite`, then the active duration is determined from the *Intermediate Active Duration* computation given below:

**PAD** = *Result from Intermediate Active Duration Computation*

Otherwise, an end value not equal to `indefinite` is specified along with at least one of dur, repeatDur, and repeatCount. Then the **PAD** is the minimum of the result from the *Intermediate Active Duration Computation* given below and duration between end and the element begin:

**PAD** = MIN( *Result from Intermediate Active Duration Computation*, end - **B**)

Finally, the computed active duration **AD** is obtained by applying min and max semantics to the preliminary active duration **PAD**. In the following expression, if there is no min value, substitute a value of 0, and if there is no max value, substitute a value of "indefinite":

**AD** = MIN( max, MAX( min, **PAD** ))

*Intermediate Active Duration Computation*

We define three intermediate quantities, p0, p1, and p2, and produce an intermediate result, the *Intermediate Active Duration* (**IAD**) to be used in the computation above.

**p0** is the simple duration from the Simple Duration Table, given above.

If repeatCount is not specified, **p1** has the value `indefinite`. Otherwise, **p1** is the accumulated sum of the specified number of simple durations of the iterations of this element. **p1** will have a value of unresolved until the simple duration for each iteration is resolved. Partial iterations will contribute the specified fraction of the simple duration to the sum. This product can be based on either the known fixed simple duration of the media, or if unknown, the simple duration from the previous iteration of the current set of repetitions. In general for media without a fixed simple duration, **p1** will not be resolved until the specified integral number of simple durations has passed.

**p2** is the value of repeatDur. If repeatDur is unspecified, then **p2** will have a value of `indefinite`.

Then **IAD** is given by:

If **p0** equals 0, then

**IAD** = **0**

Else if repeatDur *and* repeatCount are unspecified then:

**IAD** = **p0**

else:

**IAD** = MIN( **p1**, **p2**, `indefinite`)

*This section is informative*

As an example, if an element specifies:

```
<smil ...>
...
<audio dur="5s" end="foo.activateEvent" .../>
...
</smil>
```

The active duration is initially defined as 5 seconds, based upon the specified simple duration. If the user activates "foo" before 5 seconds, the end value becomes resolved and the active duration is re-evaluated. This causes the element to end at the time of the activation.

Some of the rules and results that are implicit in the algorithm, and that should be noted in particular are:

- It is possible to have an indefinite simple duration and a defined, finite active duration, or a simple duration that is greater than the active duration. In these cases, the active duration will *constrain* (cut short) the simple duration, but the active duration does not re-define the simple duration, or change its value.
- If the begin time for an element is not resolved, it may not be possible to compute the simple or active duration.

It is possible to combine scheduled and interactive timing. For example:

```
...
<smil ...>
...
<par dur="30s">
   <img id="mutebutton" src="mute.jpg"/>
   <text  src="description.html" />
   <audio src="audio.au" end="mutebutton.activateEvent"/>
</par>
...
</smil>
```

The image and the text appear for the specified duration of the par (30 seconds). The active duration of the audio is initially defined to be indefinite because its end time is unresolved. The audio will stop early if the image is activated (e.g., clicked) before the implicit end of the audio. If the image is not activated, the dur attribute on the parent time container will constrain playback.

It is possible to declare both a scheduled duration, as well as an event-based active end. This facilitates what are sometimes called "lazy interaction" use-cases, such as a slideshow that will advance in response to user clicks, or on its own after a specified amount of time:

```
<html ...>
...
<seq>
   <img src="slide1.jpg" dur="10s" end="click" />
   <img src="slide2.jpg" dur="10s" end="click" />
   <img src="slide3.jpg" dur="10s" end="click" />
   <!-- etc., etc. -->
</seq>
...
</html>
```

In this case, the active end of each element is defined to be the earlier of the specified duration, or a click on the element. This lets the viewer sit back and watch, or advance the slides at a faster pace.

**Paused elements and the active duration**

An element can be paused while it is active. This may happen in a number of ways, including via a DOM method call or because of excl semantics. When an element is paused, a resolved end time for the element may change, or it may become unresolved. The synchronization relationship between the paused element and its parent time container is re-established when the paused element is resumed. If for example the element below is paused with a DOM method call, there is no way to know when the element will end, and so the end time must be considered unresolved:

```
<img dur="30s" .../>
```

However, in the following case, the "bar" element will still end at 10 seconds, even if it is paused at 8 seconds. In this case, the end time does not change:

```
<img id="foo" dur="10s" .../>
<img id="bar" end="foo.end" .../>
```

Finally, in the following case the "foo" element will initially be computed to end at 10 seconds. If the "bar" element begins (i.e. if the user activates or clicks on "foo"), at 8 seconds, "foo" will be paused. However, since the duration of "bar" is known, and the semantics of the excl pause queue are well defined, the end of "foo" can be computed to be 15 seconds:

```
<smil ...>
...
<excl dur="indefinite">
   <priorityClass peers="pause">
      <img id="foo" dur="10s" .../>
      <img id="bar" begin="foo.activateEvent" dur="5s" .../>
   </priorityClass>
</excl>
...
</smil>
```

If there is enough information to determine the new end time (as in the example above), an implementation must compute the correct end time when an element is paused. Any change to the end time that results from the element being paused must be propagated to any sync arc time dependents (i.e. other elements with a begin or end defined relative to the active end of the paused element). See also the Propagating changes to times section.

In addition, when an element is paused, the accumulated synchronization offset will increase to reflect the altered sync relationship. See also The accumulated synchronization offset.

Finally, when an element is paused it may end because the parent time container ends. In this case, any fill behavior is interpreted using the element active time when the element ends (that is, it will use the element active time at which it was paused to determine what to display).

**Evaluation of begin and end time lists**

*This section is informative*

Children of par and excl time containers can have multiple begin and end values. We need to specify the semantics associated with multiple begin and end times, and how a dynamic timegraph model works with these multiple times.

The model is based around the idea of *intervals* for each element. An interval is defined by a begin and an end time. As the timegraph is played, more than one interval may be created for an element with multiple begin and end times. At any given moment, there is one *current interval* associated with each element. Intervals are created by evaluating a list of begin times and a list of end times, each of which is based upon the *conditions* described in the begin and end attributes for the element.

The list of begin times and the list of end times used to calculate new intervals are referred to as lists of "instance times". Each instance time in one of the lists is associated with the specification of a begin or end condition defined in the attribute syntax. Some conditions - for example offset-values - only have a single instance in the list. Other conditions may have multiple instances if the condition can happen more than once. For example a syncbase-value can have multiple instance times if the *syncbase* element has played several intervals, and an event-value may have multiple instance times if the event has happened more than once.

The instance times lists for each element are initialized when the timegraph is initialized, and exist for the entire life of the timegraph. Some instance times such as those defined by offset-values remain in the lists forever, while others may come and go. For example, times associated with event-values are only added when the associated event happens, and are removed when the element *resets*, as described in Resetting element state. Similarly, Instance times for syncbase-values are added to the list each time a new interval is created for the syncbase element, but these instance times are not removed by a reset, and remain in the list.

When the timegraph is initialized, each element attempts to create a first current interval. The begin time will generally be resolved, but the end time may often be unresolved. If the element can restart while active, the current interval can end (early) at the next begin time. This interval will play, and then when it ends, the element will review the lists of begin and end instance times. If the element should play again, another interval will be created and this new interval becomes the *current interval*. The history of an element can be thought of as a set of intervals.

Because the begin and end times may depend on other times that can change, the current interval is subject to change, over time. For example, if any of the instance times for the *end* changes while the current interval is playing, the current interval end will be recomputed and may change. Nevertheless, once a time has *happened*, it is fixed. That is, once the current interval has begun, its begin time can no longer change, and once the current interval has ended, its end time can no longer change. For an element to restart, it must end the current interval and then create a new current interval to effect the restart.

When a begin or end condition defines a time dependency to another element (e.g. with a syncbase-value), the time dependency is generally thought of as a relationship between the two elements. This level of dependency is important to the model when an element creates a new current interval. However, for the purposes of propagating changes to individual times, time dependencies are more specifically a dependency from a given *interval of the syncbase element* to a particular *instance time* in one of the dependent element's instance time lists. Since only the current interval's begin and end times can change, only the current interval will generate time-change notices and propagate these to the dependent instance times.

end times can change, only the current interval will generate time change notices and propagate these to the dependent instance times.

When this section refers to the begin and end times for an element, the times are described as being in the space of the *parent simple duration*. All sync-arcs, event arcs, wallclock values, etc. must be converted to this time space for easy comparison. This is especially important when referring to begin times "before 0", which assumes that "0" is the beginning of the parent simple duration. The model does not depend upon this definition - e.g. an implementation could do everything in global document time.

Cycles in the timegraph must be detected and broken to ensure reasonable functioning of the implementation. A model for how to do this in the general case is described (it is actually an issue that applies even to SMIL 1.0). A mechanism to support certain useful cyclic dependencies falls out of the model.

The rest of this section details the semantics of the instance times lists, the element life cycle, and the mechanisms for handling dependency relationships and cycles.

*The instance times lists*

*This section is normative*

Instance lists are associated with each element, and exist for the duration of the document (i.e. there is no *life cycle* for instance lists). Instance lists may change, and some times may be added and removed, but the begin and end instance times lists are persistent.

Each element can have a begin attribute that defines one or more conditions that can begin the element. In addition, the timing model describes a set of rules for determining the end of the element, including the effects of an end attribute that can have multiple conditions. In order to calculate the times that should be used for a given interval of the element, we must convert the begin times and the end times into parent simple time, sort each list of times (independently), and then find an appropriate pair of times to define an interval.

The instance times can be resolved or unresolved. In the case of the end list, an additional special value "indefinite" is allowed. The lists are maintained in sorted order, with "indefinite" sorting after all other resolved times, and unresolved times sorting to the end.

For begin, the list interpretation is straightforward, since begin times are based only upon the conditions in the attribute or upon the default begin value if there is no attribute. However, when a begin condition is a syncbase-value, the syncbase element may have multiple intervals, and we must account for this in the list of begin times associated with the conditions.

For end, the case is somewhat more complex, since the end conditions are only one part of the calculation of the end of the active duration. The instance times list for end are used together with the other SMIL Timing semantics to calculate the actual end time for an interval.

If an instance time was defined as syncbase-values, the instance time will maintain a time dependency relationship to the associated interval for the syncbase element. This means that if the associated begin or end time of the syncbase current interval changes, then the dependent instance time for this element will change as well.

When an element creates a new interval, it notifies time dependents and provides the begin and end times that were calculated according to the semantics described in "Computing the active duration". Each dependent element will create a new instance time tied to (i.e. with a dependency relationship to) the new syncbase current interval.

BUILDING THE INSTANCE TIMES LISTS

The translation of begin or end conditions to instance times depends upon the type of condition:

- **offset-values** are the simplest. Each offset-value condition yields a single instance time. This time remains in the list forever, and is unaffected by reset of the element, or by repeat or restart of the parent (or other ascendants).
- **wallclock-sync-values** are similar to offset values. Each wallclock-sync-value condition yields a single instance time. This time remains in the list forever, however each time an ascendant restarts or repeats, these times are *reconverted* from the wallclock time space to the new parent simple time space.
- **event-values, accesskey-values and repeat**-values are all treated similarly. These conditions do not yield an instance time unless and until the associated event happens. Each time the event happens, the condition yields a single instance time. The event time plus or minus any offset is added to the list. If the event happens multiple times during a parent simple duration, there may be multiple instance times in the list associated with the event condition. However, an important distinction is that event times are cleared from the list each time the element is reset (see also Resetting element state). Within this section, these three value types are referred to collectively as *event value conditions*.
- **syncbase-values and media-marker-values** are treated similarly. These conditions do not yield an instance time unless and until the associated syncbase element creates an interval. Each time the syncbase element creates a new interval, the condition yields a single instance time. The time plus or minus any offset is *added* to the list. Unlike event times, syncbase times are *not* cleared from the element's lists simply because the element is reset. Instead, a resolved syncbase time is removed from the list when a common ascendant of the syncbase *and* the dependent element restarts or repeats. Also each time an ascendant restarts or repeats, the remaining syncbase times are re-converted from the syncbase time space to the new parent simple time space, since syncbase times are always relative to the current parent simple time space (see also Resetting element state). Within this section, these three value types are referred to collectively as *syncbase value conditions*.
- The special value **"indefinite"** does not yield an instance time in the begin list. It will, however yield a single instance with the value "indefinite" in an end list. This value is not removed by a reset.

If no attribute is present, the default begin values must be evaluated. For children of par, this is equivalent to an offset-value of 0, and yields one persistent instance value. For children of excl, this is equivalent to "indefinite", and so does not yield an instance value.

If a DOM method call is made to begin or end the element (`beginElement()`, `beginElementAt()`, `endElement()` or `endElementAt()`), each method call creates a single instance time (in the appropriate instance times list). These time instances are cleared upon reset just as for event times. See Resetting element state.

When a new time instance is added to the begin list, the current interval will evaluate restart semantics and may ignore the new time or it may end the current interval (this is detailed in Interaction with restart semantics). In contrast, when an instance time in the begin list changes because the syncbase (current interval) time moves, this does not invoke restart semantics, but may change the current begin time: If the current interval has not yet begun, a change to an instance time in the begin list will cause a re-evaluation of the begin instance lists, which may cause the interval begin time to change. If the interval begin time changes, a *time-change* notice must be propagated to all dependents, and the current interval end must also be re-evaluated.

When a new instance time is added to the end list, or when an instance time in the end list changes, the current interval will re-evaluate its end time. If it changes, it must notify dependents.

If an element has already played all intervals, there may be no current interval. In this case, additions to either list of instance times, as well as changes to any instance time in either list cause the element to re-evaluate the lists just as it would at the end of each interval (as described in End of an interval below). This may or may not lead to the creation of a new interval for the element.

When times are added to the instance times lists, they may or may not be resolved. If they are resolved, they will be converted to parent simple time. If an instance time changes from unresolved to resolved, it will be similarly converted.

There is a difference between a unresolved instance time, and a begin or end condition that has no associated instance. If, for example, an event value condition is specified in the end attribute, but no such event has happened, there will be no associated instance time in the end list. However, if a syncbase value condition is specified for end, and if the syncbase element has a current interval, there will be an associated instance time in the end list. Since the syncbase value condition can be relative to the end of the syncbase element, and since the end of the syncbase current interval may not be resolved, the associated instance time in the end list can be unresolved. Once the syncbase current interval actually ends, the dependent instance time in the end list will get a time-change notification for the resolved syncbase interval end. The dependent instance time will convert the newly resolved syncbase time to a resolved time in parent simple time. If the instance lists did not include the unresolved instance times, some additional mechanism would have to be defined to add the end instance time when the syncbase element's current interval actually ended, and resolved its end time.

The list of resolved times includes historical times defined relative to sync base elements, and so can grow over time if the sync base has many intervals. Implementations may filter the list of times as an optimization, so long as it does not affect the semantics defined herein.

*Principles for building and pruning intervals*

*This section is informative*

The following set of principles underlie the interval model. This is not a complete model - it is just meant provide an additional view of the model.

First we define the terms *pruning* and *cutting off* an interval - these concepts should not be confused.

In some cases, after an interval has been created, it must later be *pruned* (deleted/removed from the timegraph) as more information becomes known and semantic constraints must be applied. When an interval is *pruned*, it will not be shown, it will not raise begin or end events, and any associated instance times for syncbase time dependents must be removed from the respective instance times lists. It is as though the *pruned* interval had not been specified.

In other cases, especially related to negative begin times on parent time containers, a valid interval for a child may not be shown, even though it is otherwise legal with respect to the parent time constraints. For example:

```
<par begin="-10s" dur="20s">
   <img id="slide1" src="slide1.jpg" dur="3s" />
   <img id="note1" src="note1.jpg" begin="slide1.beginEvent" dur="20s" />
</par>
```

The "slide1" image will be *cut off*, but is not *pruned*. It is *cut off* because the par could not have been started 10s before its parent time container, and instead will be started at 0s into its parent time synced at 10s into its simple duration. The "slide1" image begins and ends before 10s into the par, and so cannot be shown and is *cut off*. Intervals that are *cut off* are not shown and do not raise begin or end events, but still create valid instance times for any syncbase time dependents. Thus, "slide2" *will* be shown (the interval is from minus 4 seconds to 6 seconds, document time, and so will be shown for 6 seconds, from 0 seconds to 6 seconds), but "note1" will not be shown.

The principles underlying the interval life cycle model are:

1. Try to build the current interval as early as possible.
    A. The "next" interval can be computed no earlier than the end of the current interval.
2. Do not change any interval time that is in the past. Do not prune an interval that has already begun. Note that this refers to **intervals** and not **instance times**.
3. When building an interval from a set of instance times, if the duration is resolved and negative, reject the interval; do not propagate the interval to time dependents.
    A. When the current interval has not yet begun, if the interval times change such that the duration is negative, prune the interval.
4. When building an interval from a set of instance times, if the end is resolved and is <= 0 (in parent simple time), reject the interval; do not propagate the interval to time dependents.
    A. When the current interval has not yet begun, if the interval times change such that the end is <= 0, prune the interval.
5. When building an interval from a set of instance times, if the interval begin is >= the (resolved) simple end of the parent time container, reject the interval.
    A. When the current interval has not yet begun, if the interval times change such that the begin is >= the parent time container simple end, prune the interval.
    B. When the current interval has not yet begun, if the parent simple end time changes such that the current interval begin is >= the parent time container simple end, prune the interval.

An implication of principle 5 is that we will get no intervals with **unresolved** begin times, since these will necessarily compare >= the parent simple end.

*Element life-cycle*

This section is normative

The life cycle of an element can be thought of as the following basic steps:

1. Startup - getting the first interval
2. Waiting to begin the current interval
3. Active time - playing an interval
4. End of an interval - compute the next one and notify dependents
5. Post active - perform any fill and wait for any next interval

Steps 2 to 5 can loop for as many intervals as are defined before the end of the parent simple duration. At any time during step 2, the begin time for the current interval can change, and at any time during steps 2 or 3, the end time for the current interval can change. When either happens, the changes are propagated to time dependents.

When the document and the associated timegraph are initialized, the instance lists are empty. The simple offset values and any "indefinite" value in an end attribute can be added to the respective lists as part of initialization, as they are independent of the begin time of parent simple time.

When an element has played all allowed instances, it can be thought of as stuck in step 5. However any changes to the instance lists during this period cause the element to jump back to step 4 and consider the creation of a new current interval.

### Startup - getting the first interval

An element life cycle begins with the beginning of the simple duration for the element's parent time container. That is, each time the parent time container (or more generally *any* ascendant time container) repeats or restarts, the element resets (see also Resetting element state) and starts "life" anew.

Three things are important about the beginning of the life-cycle:

1. Any and all resolved times defined as event-values, repeat-values, accesskey-values or added via DOM method calls are cleared.
2. Any and all resolved times defined as syncbase-values, wallclock-sync-values or media-marker-values must be reconverted from the syncbase time space to the parent simple time space.
3. The first current interval is computed.

Action 1) is also described in Resetting element state. This action also happens each time the element restarts, although in that case the element must not clear an event time that defined the current begin of the interval.

Action 2) Simply updates values to reflect the current sync relationship of the parent simple duration to the rest of the document.

The third action requires some special consideration of the lists of times, but is still relatively straightforward. It is similar to, but not the same as the action that applies when the element ends (this is described in End of an interval). The basic idea is to find the first interval for the element, and make that the current interval. However, the model should handle three edge cases:

> The element can begin before the parent simple begin time (i.e. before 0 in parent simple time), and so appears to begin part way into the local timeline (somewhat like a clipBegin effect on a media element). The model must handle begin times before the parent begin.
> The element has one or more intervals defined that begin *and end* before the parent simple begin (before 0). These are filtered out of the model.
> The element has one or more intervals defined that begin after the parent simple end. These are filtered out of the model. Note that if the parent simple end is unresolved, any resolved begin time happens before the parent simple end.

Thus the strict definition of the first acceptable interval for the element is the first interval that ends after the parent simple begin, and begins before the parent simple end. Here is some pseudo-code to get the first interval for an element. It assumes an abstract type "Time" that supports a compare function. It can be a resolved numeric value, the special value INDEFINITE (only used with end), and it can be the special value UNRESOLVED. Indefinite compares "greater than" all resolved values, and UNRESOLVED is "greater than" both resolved values and INDEFINITE. The code uses the instance times lists associated with the begin and end attributes, as described in the previous section.

```
// Utility function that returns true if the end attribute specification
// includes conditions that describe event-values, repeat-values or accesskey-values.
boolean endHasEventConditions();

// Calculates the first acceptable interval for an element
// Returns:
//    Interval if there is such an interval
//    FAILURE if there is no such interval
Interval getFirstInterval()
{
Time beginAfter=-INFINITY;

while( TRUE ) // loop till return
{
   Set tempBegin = the first value in the begin list that is >= beginAfter.
   If there is no such value  // No interval
      return FAILURE;

   If ( tempBegin == INDEFINITE ) // Can't begin after the parent simple end
      return FAILURE;

   If there was no end attribute specified
      // this calculates the active end with no end constraint
      tempEnd = calcActiveEnd( tempBegin );
   else
   {
      // We have a begin value - get an end
      Set tempEnd = the first value in the end list that is >= tempBegin.
      // Allow for non-0-duration interval that begins immediately
      // after a 0-duration interval.
      If tempEnd == tempBegin && tempEnd has already been used in
        an interval calculated in this method call
      {
         set tempEnd to the next value in the end list that is > tempEnd
      }
      If there is no such value
      {
         // Events leave the end open-ended. If there are other conditions
         // that have not yet generated instances, they must be unresolved:
         if endHasEventConditions()
            OR if the instance list is empty
            tempEnd = UNRESOLVED;
         // if all ends are before the begin, bad interval
         else
            return FAILURE;
      }
      // this calculates the active dur with an end constraint
      tempEnd = calcActiveEnd( tempBegin, tempEnd );
   }

   // We have an end - is it after the parent simple begin?
   if( tempEnd > 0 )
      return( Interval( tempBegin, tempEnd ) );

   else
      // Change beginAfter to find next interval, and loop
      beginAfter = tempEnd;

} // close while loop

} // close getFirstInterval
```

Note that while we might consider the case of `restart=always` separately from `restart=whenNotActive`, it would just be busy work since we need to find an interval that begins *after* `tempEnd`.

If the model yields no first interval for the element, it will never begin, and so there is nothing more to do at this point. However if there is a valid interval, the element must notify all time dependents that there is a new interval of the element. This is a notice from this element to all elements that are direct time dependents. This is distinct from the propagation of a changed time.

When a dependent element gets a "new interval" notice, this includes a reference to the new interval. The new interval will generally have a resolved begin time and may have a resolved end time. An associated instance time will be added to the begin or end instance time list for the dependent element, and this new instance time will maintain a time dependency relationship to the syncbase interval.

### Waiting to begin the interval

This period only occurs if the current interval does not begin immediately when (or before) it is created. While an interval is waiting to begin, any changes to syncbase element current interval times will be propagated to the instance lists and may result in a change to the current interval.

If the element receives a "new interval" notice while it is waiting to begin, it will *add* the associated time (i.e. the begin or end time of the syncbase interval) to the appropriate list of resolved times.

When an instance time changes, or when a new instance time is added to one of the lists, the element will re-evaluate the begin or end time of the current interval (using the same algorithm described in the previous section). If this re-evaluation yields a changed interval, time change notice(s) will be sent to the associated dependents.

It is possible during this stage that the begin and end times could change such that the interval would never begin (e.g. the interval end is before the interval begin). In this case, the interval must be *pruned* and all dependent instance times must be removed from the respective instance lists of dependent elements. These changes to the instance lists will cause re-evaluation of the dependent element current intervals, in the same manner as a changed instance time does.

One exception to normal processing is made for elements that are *deferred* according to excl interrupt semantics: a deferred element ignores propagated changes to its begin time. This is detailed in the Deferred elements and propagating changes to begin section.

### Active time - playing an interval

This period occurs when the current interval is active (i.e. once it has begun, and until it has ended). During this period, the end time of the interval can change, but the begin time cannot. If any of the instance times in the begin list change after the current interval has begun, the change will not affect the current interval. This is different from the case of *adding* a new instance time to the begin list, which *can* cause a restart.

If the element receives a "new interval" notice while it is active, it will *add* the associated time (i.e. the begin or end time of the syncbase interval) to the appropriate list of resolved times. If the new interval adds a time to the begin list, restart semantics are considered, and this may end the current interval.

If restart is set to "always", then the current interval will end early if there is an instance time in the begin list that is before (i.e. earlier than) the defined end for the current interval. Ending in this manner will also send a changed time notice to all time dependents for the current interval end. See also Interaction with restart semantics.

### End of an interval

If an element specifies `restart="never"` then no further action is taken at the end of the interval, and the element sits in the "post interval" state unless and until an ascendant time container repeats or restarts.

If an element specifies other values for `restart`, when it ends the current interval the element must reconsider the lists of resolved begin and end times. If there is another legal interval defined to begin at or after the just completed end time, a new interval will be created. When a new interval is created it becomes the *current interval* and a new interval notice is sent to all time dependents.

The algorithm used is very similar to that used in step 1, except that we are interested in finding an interval that begins after the most recent end.

```
// Calculates the next acceptable interval for an element
// Returns:
//    Interval if there is such an interval
//    FAILURE if there is no such interval
Interval getNextInterval()
{
// Note that at this point, the just ended interval is still the "current interval"
Time beginAfter=currentInterval.end;

   Set tempBegin = the first value in the begin list that is >= beginAfter.
   If there is no such value  // No interval
      return FAILURE;

   If tempBegin >= parentSimpleEnd // Can't begin after parent ends
      return FAILURE

   If there was no end attribute specified
      // this calculates the active end with no end constraint
      tempEnd = calcActiveEnd( tempBegin );
   else
   {
      // We have a begin value - get an end
      Set tempEnd = the first value in the end list that is >= tempBegin
      // Allow for non-0-duration interval that begins immediately
      // after a 0-duration interval.
      If tempEnd == currentInterval.end
      {
         set tempEnd to the next value in the end list that is > tempEnd
      }
      If there is no such value
      {
         // Events leave the end open-ended. If there are other conditions
         // that have not yet generated instances, they must be unresolved.
         if endHasEventConditions()
            OR if the instance list is empty
            tempEnd = UNRESOLVED;
         // if all ends are before the begin, bad interval
         else
```

```
            return FAILURE;
        }
        // this calculates the active dur with an end constraint
        tempEnd = calcActiveEnd( tempBegin, tempEnd );
    }

    return( Interval( tempBegin, tempEnd ) );

} // close getNextInterval
```

### Post active

This period can extend from the end of an interval until the beginning of the next interval, or until the end of the parent simple duration (whichever comes first). During this period, any fill behavior is applied to the element. The times for this interval can no longer change. Implementations may as an optimization choose to break the time dependency relationships since they can no longer produce changes.

*Interaction with restart semantics*

There are two cases in which restart semantics must be considered:

1. When the current interval is playing, if `restart="always"` then any instance time (call it T) in the begin list that is after (i.e. later than) the current interval begin but earlier than the current interval end will cause the current interval to end at time T. This is the first step in restarting the element: when the current interval ends, that in turn will create any following interval.
2. When a new instance time is added to the begin list of instance times, restart rules can apply. The new instance times may result from a begin condition that specifies one of the syncbase value conditions, for which a new instance notice is received. It may also result from a begin condition that specifies one of the event value conditions, for which the associated event happens.
   In either case, the restart setting and the state of the current interval controls the resulting behavior. The new instance time is computed (e.g. from the syncbase current interval time or from the event time, and including any offset), and added to the begin list. Then:
   - If the current interval is waiting to play, the element recalculates the begin and end times for the current interval, as described in the Element life-cycle step 1 (for the first interval) or step 4 (for all later intervals). If either the begin or end time of the current interval changes, these changes must be propagated to time dependents accordingly.
   - If the current interval is playing (i.e. it is active), then the restart setting determines the behavior:
     - If `restart="never"` then nothing more is done. It is possible (if the new instance time is associated with a syncbase value condition) that the new instance time will be used the next time the element life cycle begins.
     - If `restart="whenNotActive"` then nothing more is done. If the time falls within the current interval, the element cannot restart, and if it falls after, then the normal processing at the end of the current interval will handle it. If the time falls before the current interval, as can happen if the time includes a negative offset, the element does not restart (the new instance time is effectively ignored).
     - If `restart="always"` then case 1 above applies, and will cause the current interval to end.

*Cyclic dependencies in the timegraph*

There are two types of cycles that can be created with SMIL 2.0, *closed* cycles and *open* or *propagating* cycles. A *closed* cycle results when a set of elements has mutually dependent time conditions, and no other conditions on the affected elements can affect or change this dependency relationship, as in examples 1 and 2 below. An *open* or *propagating* cycle results when a set of elements has mutually dependent time conditions, but at least one of the conditions involved has more than one resolved condition. If any one of the elements in the cycle can generate more than one interval, the cycle can propagate. In some cases such as that illustrated in example 3, this can be very useful.

Times defined in a closed cycle are unresolved, unless some external mechanism resolves one of the element time values (for example a DOM method call or the traversal of a hyperlink that targets one of the elements). If this happens, the resolved time will propagate through the cycle, resolving all the associated time values.

Closed cycles are an error, and may cause the entire document to fail. In some implementations, the elements in the cycle may just not begin or end correctly. Examples 1 and 2 describe the most forgiving behavior, but implementations may simply reject a document with a closed cycle.

*Detecting Cycles*

Implementations can detect cycles in the timegraph using a *visited* flag on each element as part of the processing that propagates changes to time dependents. As a changed time notice is propagated, each dependent element is marked as having been *visited*. If the change to a dependent instance time results in a change to the current interval for that element, this change will propagate in turn to its dependents. This second *chained* notice happens in the context of the first time-change notice that caused it. The effect is like a stack that builds as changes propagate throughout the graph, and then unwinds when all changes have propagated. If there is a dependency cycle, The propagation path will traverse an element twice during a given propagation chain. This is a common technique use in graph traversals.

A similar approach can be used when building dependency chains during initialization of the timegraph, and when propagating new interval notices - variations on the theme will be specific to individual implementations.

When a cycle is detected, the change propagation is ignored. The element that detected the second visit ignores the second change notice, and so breaks the cycle.

*Examples*

Example 1: In the following example, the 2 images define begin times that are mutually dependent. There is no way to resolve these, and so the images will never begin.

```
<img id="foo" begin="bar.begin" .../>
<img id="bar" begin="foo.begin" .../>
```

Example 2: In the following example, the 3 images define a less obvious cycle of begin and end times that are mutually dependent. There is no way to resolve these. The image "joe" will begin but will never end, and the images "foo" and "bar" will never begin.

```
<img id="foo" begin="joe.end" .../>
<img id="bar" begin="foo.begin" dur="3s" .../>
<img id="joe" begin="0" end="bar.end" .../>
```

the cycle to propagate forwards. The image "foo" will first be displayed from 0 to 3 seconds, with the second image "bar" displayed from 2 to 5 seconds. As each new current interval of "foo" and "bar" are created, they will add a new instance time to the other element's begin list, and so the cycle keeps going forward. As this overlapping "ping-pong" behavior is not otherwise easy to author, these types of cycles are not precluded. Moreover, the correct behavior will fall out of the model described above.

```
<img id="foo" begin="0; bar.begin+2s" dur="3s" .../>
<img id="bar" begin="foo.begin+2s" dur="3s" .../>
```

Example 4: In the following example, an open cycle is described that propagates backwards. The intended behavior does not fall out of the model, and is not supported. In this example, however, each time the parent time container repeats, the video elements will begin one second earlier than they did in the previous parent iteration. This is because the begin instance times associated with syncbase value conditions are not cleared when the parent repeats. By the last iteration of the parent time container, both video elements would begin so early that they will be completely cut off by the parent begin constraint.

```
<par dur="10s" repeatCount="11" >
   <video id="foo" begin="0; bar.begin-1s" dur="10s" .../>
   <video id="bar" begin="foo.begin-1s" dur="10s" .../>
</par>
```

**Timing and real-world clock times**

*This section is informative*

In this specification, elements are described as having local "time". In particular, many offsets are computed in the simple time of a parent time container. However, simple durations can be repeated, and elements can begin and restart in many ways.

*This section is normative*

- There is no direct relationship between the local "time" for an element, and the real world concept of time as reflected on a clock.

**Interval timing**

*This section is informative*

The SMIL timing model assumes the most common model for *interval timing*

*This section is normative*

- Interval timing describes intervals of time (i.e. durations) in which the begin time of the interval is included in the interval, but the end time is excluded from the interval.

*This section is informative*

This is also referred to as *end-point exclusive* timing. This model makes arithmetic for intervals work correctly, and provides sensible models for sequences of intervals.

*Background rationale*

In the real world, this is equivalent to the way that seconds add up to minutes, and minutes add up to hours. Although a minute is described as 60 seconds, a digital clock never shows more than 59 seconds. Adding one more second to "00:59" does not yield "00:60" but rather "01:00", or 1 minute and 0 seconds. The theoretical end time of 60 seconds that describes a minute interval is excluded from the actual interval.

In the world of media and timelines, the same applies: Let "A" be a video, a clip of audio, or an animation. Assume "A" begins at 10 and runs until 15 (in any units - it does not matter). If "B" is defined to follow "A", then it begins at 15 (and not at 15 plus some minimum interval). When a runtime actually renders out frames (or samples for audio), and must render the time "15", it should not show both a frame of "A" and a frame of "B", but rather should only show the new element "B". This is the same for audio, or for any interval on a timeline. If the model does not use endpoint-exclusive timing, it will draw overlapping frames, or have overlapping samples of audio, of sequenced animations, etc.

Note that transitions from "A" to "B" also adhere to the interval timing model. They *do* require that "A" not actually end at 15, and that both elements actually overlap. Nevertheless, the "A" duration is simply extended by the transition duration (e.g. 1 second). This new duration for "A" is *also* endpoint exclusive - at the end of this new duration, the transition will be complete, and only "B" should be rendered - "A" is no longer needed.

*Implications for the time model*

For the time model, several results of this are important: the definition of repeat, and the state of the element applied or displayed when the element is "frozen".

```
<video dur="4s" repeatCount="4" .../>
```

At time 0, the simple duration is also at 0, and the first frame of video is presented. This is the *inclusive* begin of the interval. The simple duration proceeds normally up to 4 seconds.

*This section is normative*

- The appropriate way to map time on the active duration to time on the simple duration is to use the remainder of division by the simple duration:

  `simpleTime = REMAINDER( t, d )` where t is within the active duration

  Note: `REMAINDER( t, d )` is defined as `t - (d*floor(t/d))`

Using this, a time of **4** (or 8 or 12) maps to the time of **0** on the simple duration. The endpoint of the simple duration is *excluded* from (i.e. not actually sampled on) the simple duration.

For most continuous media, this aligns to the internal media model, and so no frames (or audio samples) are ever excluded. However for sampled timeline media (like animation), the distinction is important, and requires a specific semantic for elements that are frozen.

- If the active duration is an even multiple of the simple duration, the media to show when frozen is the last frame (or last value) defined for the simple duration.

The effect of this semantic upon animation functions is detailed in the [SMIL-ANIMATION] module.

**Event sensitivity**

*This section is informative*

The SMIL 2.0 timing model supports synchronization based upon unpredictable events such as DOM events or user interface generated events. The model for handling events is that the notification of the event is delivered to the timing element, and the timing element uses a set of rules to resolve any synchronization dependent upon the event.

*This section is normative*

The semantics of element sensitivity to events are described by the following set of rules:

1. While a time container is not active (i.e. before the time container begin or after the time container active end), child elements do *not* respond to events (with respect to the Time model). Note that while a parent time container is frozen, it is not active, and so children do not handle begin or end event specifications.
    a. If an element and an ascendant time container are both specified to begin with the same event, the behavior is not predictable (based upon DOM event semantics). Authors are discouraged from authoring these cases.
2. If an element is not active (but the parent time container is), then events are only handled for begin specifications. Thus if an event is raised and begin specifies the event, the element begins. While the element is not active, any end specification of the event is ignored.
3. If an element is (already) active when an event is raised, and begin specifies the event, then the behavior depends upon the value of restart:
    a. If restart="always", then a new begin time is resolved for the element based on the event time. Any specification of the event in end is ignored for this event instance.
    2. If restart="never" or restart="whenNotActive", then any begin specification of the event is ignored for this instance of the event. If end specifies the event, an end value is resolved based upon the event time, and the active duration is re-evaluated (according to the rules in Computing the active duration).

It is important to notice that in no case is a single event occurrence used to resolve both a begin and end time on the same element.

*This section is informative*

Rule 1a discourages the use of cases such as the following:

```
<smil ...>
...
<par id="bad_example" begin="link9.activateEvent">
   <img begin="link9.activateEvent" .../>
</par>
...
</smil>
```

Various alternative approaches can be used. One possible approach is to define the descendent element to begin relative to the ascendant begin, as in the following example (the begin rule for the image could be simpler, but this illustrates the general point):

```
<smil ...>
...
<par id="better_example" begin="link9.activateEvent">
   <img begin="better_example.begin" .../>
...
</par>
</smil>
```

The event sensitivity rules can be used with the restart attribute to describe "toggle" activation use cases, as described in the section: Using restart for toggle activation.

Since the same event instance cannot be used to resolve both the begin and end time on a single element, uses like the following will have behavior that may seem non-intuitive to some people:

```
<html ...>
...
<audio src="bounce.wav" begin="foo.click"
       end="foo.click+3s" restart="whenNotActive"/>
...
</html>
```

This example will begin repeating the audio clip when "foo" is clicked, and stop the audio clip 3 seconds after "foo" is clicked *a second time*. It is incorrect to interpret this example as playing the audio clip for 3 seconds after "foo" is clicked. For that behavior, the following markup should be used:

```
<html ...>
...
<audio src="bounce.wav" begin="foo.click" dur="3s"
       restart="whenNotActive"/>
...
</html>
```

*User event sensitivity and timing*

The timing model and the user event model are largely orthogonal. While the timing model does reference user events, it does not define how these events are generated, and in particular does not define semantics of keyboard focus, mouse containment, "clickability", and related issues. Because timing can affect the presentation of elements, it may impact the rules for user event processing, however it only has an effect to the extent that the presentation of the element is affected.

In particular, many user event models will make no distinction between an element that is "playing" and one that is "frozen". The effects of the fill attribute apply only to the timing semantics. If an element is still visible while frozen, it behaves normally with respect to other semantics such as user event processing. In particular, elements such as a and area are still sensitive to user activation (e.g. clicks) when frozen.

*Link Activation compared to Event activation*

Related to event-activation is *link-activation*. Hyperlinking has defined semantics in SMIL 1.0 to seek a document to a point in time. When combined with interactive timing (e.g. begin="indefinite"), hyperlinking yields a variant on user-interactive content.

*This section is normative*

- A hyperlink can be targeted at an element that does not have a scheduled begin time.
- When the link is traversed, the element begins.
- Note that unlike event activation, the hyperlink activation is not subject to the constraints of the parent time container.

The details of when hyperlinks activate an element, and when they seek the document timeline are presented in the section Hyperlinks and timing.

**Converting between local and global times**

*This section is normative*

To convert a document time to an element local time, the original time is converted to a simple time for each time container from the root time container down to the parent time container for the element. This recursive algorithm allows for a simple model of the conversion from parent simple time to element active and element simple time. The first step calculates element active time, and the second step calculates element simple time.

The steps below assume that the associated times are resolved and not indefinite. If a required time is not resolved or is indefinite, then the conversion is not defined, and cannot be performed.

*Element active time calculation*

The input time is a time in parent simple time. This is normalized to the element active duration, adjusting for the accumulated synchronization offset (described in The accumulated synchronization offset).

Let $t_{ps}$ be a time in parent simple time, **B** be the begin time for an element, and **O** be the accumulated synchronization offset for an element, measured in parent simple time.

The element active time $t_a$ for any child element is:

$$t_a = t_{ps} - B - O$$

*Element simple time calculation*

The element simple time is the time that is used to establish runtime synchronization for a media element, or to compute an animation function's input value or sampling time. If the element is a time container, this is also the time that is seen by all children of a time container (as the time container element's simple time).

To compute the element simple time $t_s$ from an element active time $t_a$, accounting for any repeat behavior:

If there is no repeating behavior:

$$t_s = t_a$$

Else, the element simple time is just computed from the begin time of the most recent iteration - call this $t_{last\text{-}repeat}$. Some other mechanism (such as endsync logic or a media player) must note when the simple duration ends, and reset the value of $t_{last\text{-}repeat}$. If the element has not yet repeated, a value of 0 is used in place of $t_{last\text{-}repeat}$.

$$t_s = t_a - t_{last\text{-}repeat}$$

Note that the above semantic covers the special (ideal) case when the simple duration **dur** is fixed and does not vary. In this case (and this case only) $t_{last\text{-}repeat}$ can be obtained directly for the simple duration **dur** and so the expression can be reduced to:

$$t_s = \mathbf{REMAINDER}(t_a, dur)$$

where **REMAINDER( t, d )** is defined as **(t - d*floor(t/d))**.

*Converting wall-clock values*

When the document begins, the current wall-clock time is noted and saved as $t_{wallclock\text{-}begin}$. To convert a wall-clock value $t_{wc}$ to an element active simple time $t_s$, first convert $t_{wc}$ to a document global time $t_{ra}$ (i.e. an element active time for the root time container):

$$t_{ra} = t_{wc} - t_{wallclock\text{-}begin}$$

This may yield a negative time if the wallclock value is a time before the document began. Nevertheless, this is a legal value.

The time $t_{ra}$ is then converted normally to element active time or element local time as needed.

*Converting from event time to element time*

Event times are generally stamped with a time relative to system time or when the document began. The conversion is as for wallclock values, in that the event time is converted to an active time for the root time container, and then converted normally to an element time.

*Converting from element time to element time*

To convert from one element timespace to another, the time for the first element $t_{e1}$ must first be converted to a simple time on the closest ascendant time container that contains both elements. Converting from an element time to the parent time reverses the process described above. Again, it is recursive, and so the conversions are described generically from element simple to element active time, and from element active to parent simple time.

recursive, and so the conversions are described generically from element simple to element active time, and from element active to parent simple time.

To convert from element simple time to element active time requires the begin time of the most recent iteration, $t_{last\text{-}repeat}$. If the element does not repeat or has not yet repeated, a value of 0 is used in place of $t_{last\text{-}repeat}$.

$$t_a = t_s + t_{last\text{-}repeat}$$

Conversion from element active time to parent simple time uses the associated begin of the element and the accumulated synchronization offset.

$$t_{ps} = t_a + B + O$$

*Time conversions and sampling the time graph*

Note that the pure conversions do not take into account the clamping of active durations, nor the effects of fill (where time is frozen). Global to local time conversions used to translate between timespaces must ignore these issues, and so may yield a time in the destination local timespace that is well before or well after the simple duration of the element.

An alternate form of the conversion is used when actually sampling the time graph. A time container is only sampled if it is active or frozen, and so no times will be produced that are before a time container begins. If the global to local time conversion for a time container yields a time during which the time container is frozen, the time is clamped to the value of the active end.

**Hyperlinks and timing**

*This section is informative*

Hyperlinking semantics must be specifically defined within the time model in order to ensure predictable behavior. Earlier hyperlinking semantics, such as those defined by SMIL 1.0 are insufficient because they do not handle unresolved times, nor do they handle author-time restart restrictions. Here we extend SMIL 1.0 semantics for use in presentations using elements with unresolved timing (including interactive timing) and author-time restart restrictions.

*This section is normative*

A hyperlink may be targeted at an element by specifying the value of the id attribute of an element in the fragment part of the link locator. Traversing a hyperlink that refers to a timed element will behave according to the following rules:

1. If the target element is active, seek the document time back to the begin time of the current interval for the element.
2. Else if the target element begin time is resolved (i.e. there is at least one interval defined for the element), seek the document time (forward or back, as needed) to the begin time of the first interval for the target element. Note that the begin time may be resolved as a result of an earlier hyperlink, DOM or event activation. Once the begin time is resolved (and until the element is reset, e.g. when the parent repeats), hyperlink traversal always seeks. For a discussion of "reset", see Resetting element state. Note also that for an element begin to be resolved, the begin time of all ancestor elements must also be resolved.
3. Else (i.e. there are no defined intervals for the element), the target element begin time must be resolved. This may require seeking and/or resolving ancestor elements as well. This is done by recursing from the target element up to the closest ancestor element that has a resolved begin time (again noting that for an element to have a resolved begin time, all of its ancestors must have resolved begin times). Then, the recursion is "unwound", and for each ancestor in turn (beneath the resolved ancestor) as well as the target element, the following steps are performed:
    1. If the element begin time is resolved, seek the document time (forward or back, as needed) to the begin time of the first interval for the target element.
    2. Else (if the begin time is not resolved), just resolve the element begin time at the current time on its parent time container (given the current document position). Disregard the sync-base or event base of the element, and do not "back-propagate" any timing logic to resolve the element, but rather treat it as though it were defined with begin="indefinite" and just resolve begin time to the current parent time. This should create an interval and propagate to time dependents.

In the above rules, the following additional constraint must also be respected:

1. If a begin time to be used as the seek target occurs before the beginning of the parent time container, the seek-to time is *clamped* to the begin time of the parent time container. This constraint is applied recursively for all ascendant time containers.
2. If a begin time to be used as the seek target occurs after the end of any ascendant time container's simple duration, then the seek-to time is *clamped* to the time container simple end time.

*This section is informative*

Note that the first constraint means that a hyperlink to a child of a time container will never seek to a time earlier than the beginning of the time container. The second constraint implies that a hyperlink to a child that begins after the end of the parent simple duration will seek to the end of the parent, and proceed from there. While this may produce surprising results, it is the most reasonable fallback semantic for what is essentially an error in the presentation.

If a seek of the presentation time is required, it may be necessary to seek either forward or backward, depending upon the resolved begin time of the element and the presentation current time at the moment of hyperlink traversal.

*This section is normative*

- After seeking a document forward, the document should be in largely the same state as if the user had allowed the presentation to run normally from the current time until reaching the element begin time (but had otherwise not interacted with the document). One exception relates to event-based timing in the document, and is described below.
- Seeking the presentation time forward should also begin any other elements that have resolved begin times between the current time and the seeked-to time. The elements that are begun in this manner may still be active, may be frozen, or may already have ended at the seeked-to time.
- If an element begins and ends within the seek interval (between the current time before the hyperlink traversal and the seeked-to time) it logically begins and ends during the seek.
    - In this case, the associated `beginEvent`, `endEvent` and any `repeatEvent` events are not raised.
    - Dependent times defined relative to these events will not be resolved as a result of the seek.
    - DOM events (in profiles that support a DOM) will not be raised, and script or other listeners associated with these events will not be called.
- Any elements currently active at the time of hyperlinking should "fast-forward" over the seek interval. These elements may also be active, frozen or already ended at the seeked-to time.
    - If the element ends during the seek interval, an `endEvent` is raised. The associated time for the event is the document time before the seek.

- If the element repeats within the seek interval, any associated `repeatEvents` are not raised.
- Automatic hyperlinks (i.e. those with an actuate value of `onLoad`) will not be actuated when they are seeked over during hyperlink traversal (the active duration of the hyperlink begins and ends during the seek interval). However, automatic hyperlinks that are only partially seeked over will be actuated (seeking into the active duration of an automatic hyperlink will actuate the hyperlink). This models the effects of seeking and automatic hyperlinks in the same manner as timing events.

The net effect is that seeking forward to a presentation time puts the document into a state largely identical to that as if the document presentation time advanced undisturbed to reach the seek time. If the presentation is authored with no `beginEvent`, `endEvent` or `repeatEvent` based timing and no automatic hyperlinks, then state of the document after a seek should be identical to that had the document presentation time advanced undisturbed to reach the seeked-to time.

If the resolved activation time for an element that is the target of a hyperlink traversal occurs in the past, the presentation time must seek backwards. Seeking backwards will rewind any elements active at the time of hyperlinking.

- Just as for forward seeks, elements that begin and end within the seek intervals will not raise `beginEvent`, `endEvent` or `repeatEvent` events.
- If an element is active at the time of hyperlinking and the element's current interval begins during the seek interval, the element is turned off and an `endEvent` is raised. The associated time for the event is the document time before the seek.
- If the element repeats within the seek interval, any associated `repeatEvents` are not raised.
- Resolved begin times (e.g. a begin associated with an event) are not cleared or lost by seeking to an earlier time.
- Seeking to a time before a resolved begin time does not affect the interpretation of a "restart=never" setting for an element; once the begin time is resolved, it cannot be changed or restarted.
- When the document seeks backwards before a resolved begin for an element time, this does not reset the element.
- Once resolved, begin times are not cleared by hyperlinking. However, they can be overwritten by subsequent resolutions driven by multiple occurrences of an event (i.e. by restarting).

*This section is informative*

These hyperlinking semantics assume that a record is kept of the resolved begin time for all elements, and this record is available to be used for determining the correct presentation time to seek to. For example:

```
<html ...>
...
<par begin="0">
   <img id="A" begin="10s" .../>
   <img id="B" begin="A.begin+5s" .../>
   <img id="C" begin="B.click" .../>
   <img id="D" begin="C.begin+5s" .../>
   ...
   <a href="#D">Begin image D</a>
</par>
...
</html>
```

The begin time of elements **A** and **B** can be immediately resolved to be at 10 and 15 seconds respectively. The begin of elements **C** and **D** are unresolved when the document starts. Therefore activating the hyperlink will have no effect upon the presentation time or upon elements **C** and **D**. Now, assume that **B** is clicked at 25 seconds into the presentation. The click on **B** resolves the begin of **C**; this in turn resolves **D** to begin at 30 seconds. From this point on, traversing the hyperlink will cause the presentation time to be seeked to 30 seconds.

If at 60 seconds into the presentation, the user again clicks on **B**, **D** will become re-resolved to a presentation time of 65 seconds. Subsequent activation of the hyperlink while **D** is active will result in the seeking the presentation to 65 seconds. If the hyperlink is activated when **D** is no longer active, the presentation will seek to the earliest resolved begin time of **D**, at 30 seconds.

*Implications of beginElement() and hyperlinking for seq and excl time containers*

*This section is normative*

For a child of a sequence time container, if a hyperlink targeted to the child is traversed, this seeks the sequence to the beginning of the child.

- If the seek is forward in time and the child does not have a resolved begin time, the document time must seek past any scheduled active end on preceding elements, and then activate the referenced child. In such a seek, if the currently active element does not have a resolved active end, it should be ended at the current time. An `endEvent` event is raised, with the current time as the associated event time.
- If there are other intervening siblings (between the currently playing element and the targeted element), the document time must seek past all scheduled times, and resolve any unresolved times as seek proceeds (time will resolve to intermediate values of "now" as this process proceeds).
- As times are resolved, all associated time dependents get notified. Note however, that since no events are raised for elements that begin and end (or repeat) within the seek interval, time dependents defined relative to these events will not be notified and dependent times will not be resolved.
- When `beginElement()` or `beginElementAt()` is called for the child of a sequence time container (subject to restart semantics), any currently active or frozen child is stopped and the new child is begun at the current time (even if the element has a scheduled begin time). Unlike hyperlinking, no seek isperformed. The sequence will play normally following the child that is begun with the method call (i.e. as though the child had begun at its normal time).
- Note that the presentation agent need not actually prepare any media for elements that are seeked over, but it does need to propagate the sync behavior to all time dependents so that the effect of the seek is correct.

*This section is informative*

Note that if a hyperlink targets (or if `beginElement()` or `beginElementAt()` is called for) an element **A** defined to begin when another element **B** ends, and the other element **B** has (e.g.) an event-base or syncbase end, the hyperlink or method call will not end element **B**. It will only activate element **A**. If the two elements are siblings within a seq or excl time container, the parent time container enforces its semantics and stops (or pauses) the running element.

If a hyperlink targets a child of an excl time container, activating the link will seek to the earliest computed begin. This means that pause/defer stack semantics do not need to be accounted for when linking to an element. Instead the document timeline will simply be seeked to the first resolved time for the element, or seeked to the start of the time container and the target element simply started if there is no resolved begin time.

**Propagating changes to times**

*This section is informative*

There are several cases in which times may change as the document is presented. In particular, when an element time is defined relative to an event

There are several cases in which times may change as the document is presented. In particular, when an element time is defined relative to an event, the time (i.e. the element begin or active end) is resolved when the event occurs. Another case arises with restart behavior - the element gets a new begin and active end time when it restarts. Since the begin and active end times of one element can be defined relative to the begin or active end of other elements, any changes to times must be propagated throughout the document.

When an element "foo" has a begin or active end time that specifies a syncbase element (e.g. "bar" as below):

```
<img id="foo" begin="bar.end" .../>
```

we say that "foo" is a *time-dependent* of "bar" - that is, the "foo" begin time depends upon the active end of "bar". Any changes to the active end time of "bar" must be propagated to the begin of "foo" so that "foo" begins properly when "bar" ends. The effect on "foo" of the propagated change depends upon the state of "foo" when the change happens.

*This section is normative*

- If an element begin time or end time changes, this change must be propagated to any other elements that are defined relative to the changed time (i.e. to all time dependents). More specifically, when the begin or end of the current interval for an element changes, it must propagate the change to all dependent instance times. If the dependent element current interval begin or end times change as a result, these changes must in turn also be propagated. For details, see Evaluation of begin and end time lists.
- If an element ends later than its specified endtime (e.g. if a negative offset is specified with a userevent), the propagated time must be the computed time and not the observed time.
- If an element ends earlier than its specified endtime (e.g. if a parent time container constraint cuts short the element, or a DOM method call ends theelement), the propagated time must be the constrained time.

*Deferred elements and propagating changes to begin*

One exception to normal processing is made for elements that are *deferred* according to excl interrupt semantics. This exception is made to simplify the model: once an element is deferred, it will stop normal handling of time change notices that are propagated to the element begin conditions, as time dependents of syncbase elements. That is, with respect to the behavior of the element as a time dependent, the element behaves as though it had already begun. This exception is made so that the deferred element cannot change its begin time due to syncbase element changes, while it is deferred. In effect, the element *should have begun* at the time it was deferred, and so it should no longer handle changed time notices.

*This section is informative*

*Restart and propagating changes to times*

In some cases, the semantics of restart may preclude the correct propagation of changes to time, as in the following example:

```
<html ...>
...
<par>
   <img id="img1" dur="10s" end="click" .../>
   <video begin="img1.end-3s" restart="whenNotActive" .../>
</par>
...
</html>
```

If the user clicks the image at 8 seconds, the image will end at that point, and the changed end time will propagate to the video. However, the video will have begun at 7 seconds (3 seconds before the calculated end of 10 seconds), and cannot restart. The propagated change will be ignored. See also Interaction with restart semantics in the section on Evaluation of begin and end time lists.

**Time container duration**

*This section is informative*

The implicit duration of a time container is defined in terms of the children of the container. The children can be thought of as the "media" that is "played" by the time container element. The semantics are specific to each of the defined time container variants, and are described in the respective sections: The par element, the seq element, and the excl element.

Note that the term "computed values" should not be confused with the values of times that are dynamic within the time graph. In the following example, the video will be cut short if the user activates (e.g., clicks on) it before 10 seconds. If the user does not click, the par has a simple duration of 10 seconds. If the user activates the video at 5 seconds, the par has a simple duration of 8 seconds. Although the original end time for the video could be computed by an implementation as 10 seconds, the endsync semantics must be evaluated with the updated times that account for the user events.

```
<smil ...>
...
<par endsync="last" >
   <audio dur="6s" .../>
   <video begin="0" dur="10s" end="click" .../>
</par>
...
</smil>
```

**Time container constraints on child durations**

*This section is informative*

Time containers place certain overriding constraints upon the child elements. These constraints can cut short the active duration of any child element.

*This section is normative*

All time containers share the basic overriding constraint:

- A child element may not be active before the beginning, nor after the end of the parent simple duration.
- Note the time container is itself subject to the same constraints, and so may be cut short by some ascendant time container. When this happens, the children of the time container are also cut off, in the same manner as for the last partial repeat in the example below.

*This section is informative*

While the child may define a sync relationship that places the begin before the parent begin, the child is not active until the parent begins. This is

equivalent to the semantic described in Negative begin delays.

If the child defines an active duration (or by the same token a simple duration) that extends beyond the end of the parent simple duration, the active duration of the child will be cut short when the parent simple duration ends. Note that this does not imply that the child duration is automatically shortened, or that the parent simple duration is "inherited" by the child.

For example:

```
<par dur="10s" repeatDur="25s">
  <video dur="6s" repeatCount="2" .../>
  <text id="text1" begin="5s" dur="indefinite" .../>
  <audio begin="text1.end" .../>
</par>
```

The video will play once for 6 seconds, and then a second time but only for 4 seconds - the last 2 seconds will get cut short and will not be seen. The text shows up for the last 5 seconds of the par, and the indefinite duration is cut short at the end of the simple duration of the par. The audio will not show up at all, since it is defined to begin at the end of the active duration of the previous element (the text element). Since the text element ends when the time container ends, the audio would begin after the time container has ended, and so never is heard. When the par repeats the first time, everything has happens just as it did the first time. However the last repeat is only a partial repeat (5 seconds), and so only the video will be seen, but it will not be seen to repeat, and the last second of the video will be cut off.

In addition, excl time containers allow only one child to play at once. Subject to the priorityClass semantics, the active duration of an element may be cut short when another element in the time container begins.

*The min attribute and time container constraints on child durations*

The min attribute has no effect on the time container constraint on child duration. This constraint still applies even if a child's active duration does not satisfy a min value constraint. In the following example, the image is displayed between 5 and 8 seconds:

```
<par dur="5s">
  <img id="img" min="7s" dur="4s" .../>
</par>
```

**Time container constraints on sync-arcs and events**

*This section is informative*

SMIL 1.0 defined constraints on sync-arc definition (e.g., begin="id(image1)(begin)"), allowing references only to qualified siblings. SMIL 2.0 explicitly removes this constraint. SMIL 2.0 also adds event-based timing. Both sync-arcs and event-timing are constrained by the parent time container of the associated element as described above.

*Specifics for sync-arcs*

*This section is normative*

While a sync-arc is explicitly defined relative to a particular element, if this element is not a sibling element, then the sync is resolved as a sync-relationship *to the parent* (i.e. to an offset from the parent begin).

- If the defined sync would place the resolved element begin before the parent time container begin, part of the element will simply be cut off when it first plays. This is like the behavior obtained using clipBegin.
- However unlike with clipBegin, if the sync-arc defined child element also has repeat specified, only the first iteration will be cut off, and subsequent repeat iterations will play normally. See also Negative begin delays.

*This section is informative*

Note that in particular, an element defined with a sync-arc begin will not automatically force the parent or any ancestor time container to begin.

For the case that an element with a sync-arc is in a parent (or ancestor) time container that repeats: for each iteration of the parent or ancestor, the element is played as though it were the first time the parent timeline was playing. With each repeat of the parent, the sync-arc will be recalculated to yield a begin time relative to the parent time container. See also the section Resetting element state.

*Specifics for event-based timing*

*This section is informative*

The specifics for event-based timing are discussed in the Event Sensitivity section.

**Behavior of 0 duration elements**

*This section is normative*

- Media elements with an active duration of zero or with the same begin and end time trigger begin and end events, and propagate to time dependents. If an element's end time is before its begin time, no events are triggered (see also Evaluation of begin and end time lists).

Whether or not media with zero duration and no fill period is retrieved and/or briefly rendered is implementation dependent.

### 10.3.4 Clarifications and surprising results

When an element begins, any event-based begin times are cleared. In the following example, if an activate event occurs and then one second later bar ends, then foo begins immediately and the element does not restart four seconds later regardless of the restart setting. However, if an activate event occurs and bar does not end during the next five seconds, the element will restart at the end of that time.

```
<audio id="foo" begin="bar.end; activateEvent+5s" .../>
```

See Evaluation of begin and end time lists.

## 10.4 Integrating SMIL Timing and Synchronization into a host language

*This section is informative*

This section describes what a language designer must actually do to specify the integration of SMIL Timing and Synchronization support into a host language. This includes basic definitions, constraints upon specification, and allowed/supported events.

*This section is informative*

The host language designer must define some basic concepts in the context of the particular host language. These provide the basis for timing and presentation semantics.

*This section is normative*

- Any host language that includes SMIL 2.0 Timing and Synchronization markup (either via a hybrid DTD or schema, or via namespace qualified extensions) must preserve the semantics of the model defined in this specification.
- Only SMIL *document user agents* must support the deprecated SMIL 1.0 attribute names as well as the new SMIL 2.0 names. A SMIL *document user agent* is an application that supports playback of SMIL Language documents (i.e. documents with the associated MIME type "application/smil").
- The host language designer must define what "presenting a document"means. A typical example is that the document is displayed on a screen.
- The host language designer must define the *document*begin. Possible definitions are that the document begins when the complete document has been received by a client over a network, or that the document begins when certain document parts have been received.
- The host language designer must define the *document*end. This is typically when the associated application exits or switches context to another document.

### 10.4.2 Required definitions and constraints on element timing

*This section is normative*

- A host language must specify which elements support timing
- A host language must specify the semantics of the top level time container, even if the language does not otherwise include time containers.
- A host language must specify the semantics of an element being active, frozen, paused, and not active in the sense of timing. This may include support for additional syntax to indicate this semantic. See also the timeAction attribute.
- A host language must specify which elements define media directly. This defines which elements may take the "`media`" argument value to the dur attribute.

**Supported events for event-base timing**

- The host language must specify which event names are legal in event base values.
- If the host language defines no allowed event names, event-based timing is effectively precluded for the host language.
- Host languages may specify that dynamically created events (as per the [DOM2Events] specification) are legal as event names, and not explicitly list the allowed names.

### 10.4.3 Error handling semantics

*This section is normative*

- The host language designer may impose stricter constraints upon the error handling semantics.
- In the case of syntax errors, the host language may specify additional or stricter mechanisms to be used to indicate anerror. An example would be to stop all processing of the document, or to halt all animation.
- Host language designers may not relax the error handling specifications, or the error handling response (as described in "Handling syntax errors"). For example, host language designers may not define error recovery semantics for missing or erroneous values in the begin or end attribute values.

## 10.5 Document object model support

*This section is informative*

Any XML-based language that integrates SMIL Timing will inherit the basic interfaces defined in DOM [DOM2]. SMIL Timing specifies the interaction of timing functionality and DOM. SMIL Timing also defines constraints upon the basic DOM interfaces. A separate document will define specific DOM interfaces to support SMIL Timing, however this document presumes that there is a mechanism to begin and end elements, and to pause and resume them.

*This section is normative*

No syntax support is required to make use of the presumed interfaces, although the "indefinite" argument value on the begin and end attributes can be used to describe timing that will be initiated by DOM methods. In any case, the actions of DOM timing methods will be subject to the constraints of the time model, as described in this document.

A language integrating SMIL Timing and Synchronization need not require a DOM implementation

### 10.5.1 Element and attribute manipulation, mutation and constraints

If the timing attributes of timed elements are manipulated through DOM interfaces while the timegraph is running, the behavior is not defined by this document. Similarly, if timed elements are inserted into or removed from the document while the timegraph is running, the behavior is not defined. The behavior and any constraints related to this will be specified in a future document.

### 10.5.2 Events and event model

*This section is informative*

SMIL event-timing assumes that the host language supports events, and that the events can be bound in a declarative manner. DOM Level 2 Events [DOM2Events] describes functionality to support this.

*This section is normative*

The specific events supported are defined by the host language. If no events are defined by a host language, event-timing is effectively omitted.

This module defines a set of events that may be included by a host language. These include:

**beginEvent**
: This event is raised when the element local timeline begins to play. It will be raised each time the element begins the active duration (i.e. when it restarts, but not when it repeats). It may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was begun with a DOM method.

**endEvent**
: This event is raised at the active end of the element. Note that this event is not raised at the simple end of each repeat. This event may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was ended with a DOM method.

**repeat**
: This event is raised when the element local timeline repeats. It will be raised each time the element repeats, after the first iteration. Associated with the repeat event is an integer that indicates which repeat iteration is beginning. The value is a 0-based integer, but the repeat event is not raised for the first iteration and so the observed values will be >= 1.

If an element is restarted while it is currently playing, the element will raise an `endEvent` and then a `beginEvent`, as the element restarts.

In order to make the model operate consistently and remove the effects of synchronization slew in a chain of event times, the timestamp value associated with events such as the `beginEvent`, `endEvent`, and `repeat` events is not (necessarily) the actual time that the event is raised, nor is it the time when a time dependent is actually notified of the event. Rather the event timestamp is the *earliest* time that the event *could* be raised (given the timing model semantics, and assuming that elements would begin and end *precisely* when they are defined to). There are three basic cases corresponding to begin and end conditions with zero, positive, and negative offsets respectively:

**Example 1**

These examples assume video and audio media that are recorded to be in exact sync with one another.

```
<par dur="indefinite">
  <img id="foo" end="click" .../>
  <video id="bar" begin="foo.endEvent" .../>
  <audio id="copy" begin="foo.end" .../>
</par>
```

The image "foo" will end when the user clicks on it. The defined time of the end is actually the time of the click event (even if it takes a while to propagate the click event through the presentation mechanism). The "foo" element will raise an `endEvent` with a timestamp equal to the time of the click event. The behavior in this example is that "bar" and "copy" will be in precise synchronization (although "bar" may actually begin very slightly later, since it can take a while to propagate the events through a system).

**Example 2**

```
<par dur="indefinite">
  <img id="foo" .../>
  <video id="bar" begin="foo.click+3s" .../>
  <audio id="copy" begin="bar.beginEvent" .../>
</par>
```

The video "bar" will begin 3 seconds after the user clicks on "foo". The `beginEvent` for "bar" will have a timestamp equal to the "foo.click" event timestamp plus 3 seconds. The behavior is that in the example above, "bar" and "copy" will be in precise synchronization (although "copy" may actually begin slightly later, since it can take a while to propagate the events through a system).

**Example 3**

```
<par dur="indefinite">
  <img id="foo" .../>
  <video id="bar" begin="foo.click-3s" .../>
  <audio id="copy" begin="bar.beginEvent" .../>
</par>
```

The video "bar" will begin when the user clicks on "foo". The video will begin to play at a 3 second offset into the actual content, because it is defined to begin 3 seconds before the click. However, since "bar" cannot begin any sooner than "now" when the event is raised, it will raise a `beginEvent` that has the same time as the "foo.click" event. Thus in this case, the audio element "copy" will be precisely three seconds behind (out of sync with) the video.

Additional time model constraints can cause the `beginEvent` (or `endEvent`) event timestamp to differ from the calculated begin (or end) time for an element. For example the element can specify a begin time before the beginning of its parent time container (either with a negative offset value, or with a syncbase time that resolves to a time before the parent begin). In this case, a time dependent of the begin syncbase time will be defined relative to the calculated begin time. However, the element is constrained to not actually begin before the parent time container. The `beginEvent` will be raised when the element actually begins - in the example case when the parent time container begins. Similarly, the `endEvent` is raised when the element actually ends, which may differ from the calculated end time (e.g. when the end is specified to be after the end of the parent simple duration).

The distinction between syncbase and event times can be useful in certain situations. Consider the following example:

```
<par>
  <par begin="5s">
    <par begin="-5s">
      <img id="foo" begin="1s; 8s" dur="3s" .../>
    </par>
  </par>
  <img id="bar" begin="foo.begin" dur="1s" .../>
  <audio id="beep" begin="foo.beginEvent" dur="1s" .../>
</par>
```

The "foo" element defines two intervals. The inner par cuts off - but does not prune - the first interval, because the innermost par is constrained by the middle par and cannot actually begin until 5s into the document. However the inner par is still synchronized to the document time of 0s. As such, "bar" will play twice: once at 1 second, and again at 8 seconds, because syncbase values use calculated interval times. However the "beep" audio will only play once at 8 seconds which is when "foo" is actually displayed, because intervals that are cut off do not raise events.

While authors are unlikely to author the above example, similar cases can easily arise using syncbase timing. When it is important to distinguish the observed begin time from the scheduled begin time, event-value timing with the `beginEvent` or `endEvent` can be used. However, the author must be aware of the constraints on event-value timing. These include the event sensitivity constraints, and the fact that many implementations will not optimize scheduling and media preparation for elements with event-value timing as well as for elements with scheduled syncbase-value timing. See also the discussion Propagating changes to times.

### 10.5.3 Reserved DOM methods

*This section is normative*

SMIL Timing reserves four DOM methods for controlling the timing of elements: `beginElement()`, `beginElementAt()`, `endElement()`, and `endElementAt()`, and describes their effect on the timing model. Full definition of these methods is left to a future document describing DOM functionality.

The four DOM methods are used to begin and end the active duration of an element. Authors can (but are not required to) declare the timing to respond to the DOM using the following syntax:

```
<img begin="indefinite" end="indefinite" .../>
```

The `beginElement()`, `beginElementAt()`, `endElement()`, and `endElementAt()` methods are all subject to time container constraints in much the same way that event-based times are. If any of these methods are called when the parent time container is not active, the methods have no effect.

If a DOM method call is made to begin or end the element (`beginElement()`, `beginElementAt()`, `endElement()` or `endElementAt()`), each method call creates a single instance time (in the appropriate instance times list). These times are then interpreted as part of the semantics of lists of times, as described in Evaluation of begin and end time lists. These time instances are cleared upon reset just as for event times.

- The instance time associated with a `beginElement()`or `endElement()` call is the current presentation time at the time of the DOM method call.
- The instance time associated with a `beginElementAt()` or `endElementAt()` call is the current presentation time at the time of the DOM method call, plus or minus the specified offset. The offset is measured in parent simple time.
- Note that `beginElement()` and `beginElementAt()`are subject to restart semantics. Refer also to the section The restart attribute.

## 10.6 Glossary

### 10.6.1 General concepts

*This section is informative*

The following concepts are the basic terms used to describe the timing model.

**Synchronization relationship**

A synchronization relationship is defined by the author to express that two or more elements' playback is synchronized.

**Time graph**

A time graph is used to represent the temporal relations of elements in a document with SMIL timing. Nodes of the time graph represent elements in the document. Parent nodes can "contain" children, and children have a single parent. Siblings are elements that have a common parent. The links or "arcs" of the time graph represent synchronization relationships between the nodes of the graph.

**Descriptive terms for times**

The time model description uses a set of adjectives to describe particular concepts of timing:

***implicit***
: This describes a time that is defined intrinsically by the element media (e.g. based upon the length of a movie), or by the time model semantics (e.g., duration of par time container).

***explicit***
: This describes a time that has been specified by the author, using the SMIL syntax.

***desired***
: This is a time that the author intended - it is generally the explicit time if there is one, or the implicit time if there is no explicit time.

***effective***
: This is a time that is actually observed at document playback. It reflects both the constraints of the timing model as well as real-world issues such as media delivery.

***definite***
: A time is definite if it is resolved to a finite, non-indefinite value.

**Local time and global time**

*Global time* is defined relative to the common reference for all elements, the document root. This is sometimes also referred to as *document time*.

Within a document, when a given element is active or "plays", the contents of that element progress from the beginning of the active duration to the end of the active duration. There will also be a progression from the beginning to the end of each simple duration (the distinction is clearest when the element repeats). It is often convenient to talk about times in terms of a given element's simple duration or its active duration. Generically, this is referred to as *local time*, meaning that times are relative to an element-local reference.

The following terms are used to more precisely qualify local times:

***active time***
: Time as measured relative to the element's active duration. A time is measured as an offset from the active begin of the element.

***simple time***
: Time as measured relative to the element's simple duration. A time is measured as an offset from the beginning of a particular instance of the simple duration.

***media time***
: Time as measured relative to the element's media duration. A time is measured as an offset from the beginning of the media, as modified by any clipBegin or clipEnd attributes.

To be meaningful, these terms are described relative to some element. For example, when describing timing semantics, *element active time* refers to active time for the element under discussion, and *parent simple time* refers to simple time for that element's parent.

Conversion from global (document) time to an element time, or from one element time to another element time, is described in Converting between local and global times.

When measuring or calculating time, a reference element and the local time form (active, simple or media time) are specified. The measured time or duration is defined in terms of the element time progress. E.g. if the reference element pauses, this may impact the semantics of times or durations measured relative to the element.

**Linear and Non-linear media**

Linear media is continuous media that cannot be played in a random-access manner. For example, most Internet streaming video and audio are linear.

Non-linear media can be played in a random access manner. For example, algorithmic animation is non-linear. Discrete media may behave in a non-linear manner.

The linear or non-linear behavior of the media is not a function of the media type, but rather of the renderer or playback engine, and often depends upon the delivery mechanism for the media.

**Scheduled timing**

An element is considered to have scheduled timing if the element's start time is given relative to the begin or active end of another element. A scheduled element can be inserted directly into the time graph.

*This section is normative*

**document begin**

The start of the interval in which the document is presented is referred to as the *document begin*.

**document end**

The end of the interval in which the document is presented is referred to as the *document end*.

**document duration**

The difference between the end and the begin is referred to as the *document duration*.

*This section is informative*

**Events and interactive timing**

Begin and active end times in SMIL 2.0 can be specified to be relative to events that are raised in the document playback environment. This supports declarative, interactive timing. *Interactive* in this sense includes user events such as mouse clicks, events raised by media players like a `mediaComplete` event, and events raised by the presentation engine itself such as a `pause` event.

**Syncbases**

In scheduled timing, elements are timed relative to other elements. The syncbase for an element *A* is the other element *B* to which element *A* is relative. More precisely, it is the begin or active end of the other element. The syncbase is not simply a scheduled point in time, but rather a point in the time graph.

**Sync arcs**

"Sync-arc" is an abbreviation for "synchronization arc". Sync-arcs are used to relate nodes in the time graph, and define the timing relationship between the nodes. A sync-arc relates an element to its syncbase. The sync-arc may be defined implicitly by context, explicitly by Id-value or event name, or logically with special syntax.

**Clocks**

A Clock is a particular timeline reference that can be used for synchronization. A common example that uses real-world local time is referred to as **wall-clock** timing (e.g. specifying 10:30 local time). Other clocks may also be supported by a given presentation environment.

**UTC: Coordinated Universal Time**

"Universal Time" (abbreviated UT) is sometimes also referred to as "Greenwich Mean Time" (abbreviated GMT). The two terms are often used loosely to refer to time kept on the Greenwich meridian (longitude zero), five hours ahead of Eastern Standard Time. Times given in UT are almost always given in terms of a 24-hour clock. Thus, 14:42 is 2:42 p.m., and 21:17 is 9:17 p.m.

**Hyperlinking and timing**

A hyperlink into or within a timed document may cause a seek of the current presentation time or may activate an element (if it is not in violation of any timing model rules).

**Activation**

During playback, an element may be activated automatically by the progression of time, via a hyperlink, or in response to an event. When an element is activated, playback of the element begins.

**Discrete and continuous Media**

SMIL includes support for declaring media, using element syntax defined in "The SMIL Media Object Module". The media that is described by these elements is described as either discrete or continuous.

***discrete***
: The media does not have intrinsic timing, or intrinsic duration. These media are sometimes described as "rendered" or "synthetic" media. This includes images, text and some vector media.

***continuous***
: The media is naturally time-based, and generally supports intrinsic timing and an intrinsic notion of duration (although the duration may be indefinite). These media are sometimes described as "time-based" or "played" media. This includes most audio, movies, and time-based animations.

## 10.6.2 Timing concepts

**Time containers**

Time containers group elements together in time. They define common, simple synchronization relationships among the grouped child elements. In addition, time containers constrain the time that children may be active. Several containers are defined, each with specific semantics and constraints on its children.

**Content/Media elements**

SMIL timing and synchronization support ultimately controls a set of content or media elements. The content includes things like video and audio, images and vector graphics, as well as text or HTML content. SMIL documents use the SMIL media elements to reference this content. XML and HTML documents that integrate SMIL 2.0 functionality may use SMIL media elements and/or content described by the integrated language (e.g. paragraphs in HTML).

**Basic markup**

All elements - content/media as well as time containers - support timing markup to describe a begin time and a duration, as well as the ability to play repeatedly. There are several ways to define the begin time. The semantics vary somewhat depending upon an element's time container.

**Simple and active durations**

The time model defines two concepts of duration for each element - the simple duration and the active duration. These definitions are closely related to the concept of playing something repeatedly.

***simple duration***
: This is the duration defined by the basic begin and duration markup. It does not include any of the effects of playing repeatedly, or of fill. The simple duration is defined by the explicit begin and duration, if one is specified. If the explicit times are not specified, the simple duration is defined to be the implicit duration of the element.

***active duration***
: This is the duration during which the element plays normally. If no repeating behavior is specified, and end is not specified, the active duration is the same as the simple duration. If the element is set to play repeatedly, the simple duration is repeated for the active duration, as defined by the repeat markup.
The active duration does not include the effect of fill, except when the effect of the min attribute extends a shorter active duration. See The min and max attributes: more control over the active duration.

The constraints of a parent time container may override the duration of its children. In particular, a child element may not play beyond the simple end of the time container.

The terms for these durations can be modified with the Descriptive Terms for Times, to further distinguish aspects of the time graph.

**Hard and soft sync**

SMIL 1.0 introduced the notion of synchronization behavior, describing user agent behavior as implementing either "hard synchronization" or "soft synchronization". Using hard sync, the entire presentation would be constrained to the strict description of sync relationships in the time graph. Soft sync allowed for a looser (implementation dependent) performance of the document.

While a document is playing, network congestion and other factors will sometimes interfere with normal playback of media. In a SMIL 1.0 hard sync environment, this will affect the behavior of the entire document. In order to provide greater control to authors, SMIL 2.0 extends the hard and soft sync model to individual elements. This support allows authors to define which elements and time containers must remain in strict or "hard" sync, and which elements and time containers can have a "soft" or slip sync relationship to the parent time container.

See also the section: The syncBehavior, syncTolerance, and syncMaster attributes: controlling runtime synchronization.

**Pruning and cutting off an interval**

The concepts of interval *pruning* and *cutting off* are distinct and should not be confused.

In some cases, after an interval has been created, it must later be *pruned* (deleted/removed from the timegraph) as more information becomes known and semantic constraints must be applied. When an interval is *pruned*, it will not be shown, it will not raise begin or end events, and any associated instance times for syncbase time dependents must be removed from the respective instance times lists. It is as though the *pruned* interval had not been specified.

In other cases, especially related to negative begin times on parent time containers, a valid interval for a child may not be shown, even though it is otherwise legal with respect to the parent time constraints. These intervals are said to be *cut off*.

For example:

```
<par begin="-10s" dur="20s">
  <img id="slide1" src="slide1.jpg" dur="3s" />
  <img id="slide2" src="slide2.jpg" begin="slide1.end+3s" dur="10s" />
  <img id="note1" src="note1.jpg" begin="slide1.beginEvent" dur="20s" />
</par>
```

The "slide1" image will be *cut off*, but is not *pruned*. It is *cut off* because the par could not have been started 10s before its parent time container, and instead will be started at 0s into its parent time synced at 10s into its simple duration. The "slide1" image begins and ends before 10s into the par, and so cannot be shown and is *cut off*. Intervals that are *cut off* are not shown and do not raise begin or end events, but still create valid instance times for any syncbase time dependents. Thus, "slide2" *will* be shown (the interval is from minus 4 seconds to 6 seconds, document time, and so will be shown for 6 seconds, from 0 seconds to 6 seconds), but "note1" will not be shown.

## 10.7 Appendix A: SMIL Timing and Synchronization modules

***This section is normative.***

This section defines the nineteen SMIL 2.0 Timing Modules, which include the BasicInlineTiming module and eighteen other modules that combine to provide full SMIL 2.0 timing support. The separation of the SMIL 2.0 Timing modules is based on the inclusion of the syntactic expression of features using elements, attributes, and attribute values. Including a module in a profile adds both the syntax and associated semantics defined elsewhere in this specification to that profile.

**AccessKeyTiming**

This module defines the attribute value syntax for the begin and end attributes that allow elements to begin and end based upon the user actuating a designated access key.

**Module dependencies**

None.

**Included features**

begin and end with access key values.

**Other module specific integration requirements**

The access key requested by the author may not be made available by the player (for example it may not exist on the device used, or it may be used by the user agent itself). Therefore the user agent should make the specified key available, but may map the access key to a different interaction behavior. The user agent must provide a means of identifying the access keys that can be used in a presentation. This may be accomplished in different ways by different implementations, for example through direct interaction with the application or via the user's guide.

**BasicInlineTiming**

This module defines the attributes that make up basic timing support for adding timing to XML elements.

**Module dependencies**

None.

**Included features**

dur with all allowed values, and begin and end attributes with simple offset values, and "indefinite".

**Other module specific integration requirements**

None.

**BasicTimeContainers**

This module defines basic time container elements, attributes that describe an element's display behavior within a time container, and end conditions for time containers.

**Module dependencies**

None.

**Included features**

par, seq elements, fill, endsync attributes.

**Other module specific integration requirements**

`fill=transition` is only supported when BasicTransitions or InlineTransitions is included in the language profile. If FillDefault is not included in the profile, `fill=default` is interpreted the same as `fill=auto`.

**EventTiming**

This module defines the attribute value syntax for begin and end attributes that allow elements to begin and end in response to an event.

**Module dependencies**

None.

**Included features**

begin and end with event values.

**Other module specific integration requirements**

None. A Host language may specify that it does not support offsets on event values.

**ExclTimeContainers**

This module includes a time container that defines a mutually exclusive set of elements, and describes interrupt semantics among these elements.

**Module dependencies**

None.

**Included features**

excl, priorityClass elements, fill, endsync attributes.

**Other module specific integration requirements**

`fill=transition` is only supported when BasicTransitions or InlineTransitions is included in the language profile. If FillDefault is not included in the profile, `fill=default` is interpreted the same as `fill=auto`.

**FillDefault**

This module defines syntax for specifying default display behavior for elements.

**Module dependencies**

BasicTimeContainers or ExclTimeContainers or TimeContainerAttributes.

**Included features**

fillDefault attribute.

**Other module specific integration requirements**

`fill=transition` is only supported when BasicTransitions or InlineTransitions is included in the language profile.

**MediaMarkerTiming**

contained in the source content.

**Module dependencies**
None.
**Included features**
begin and end with media marker values.
**Other module specific integration requirements**
None.

**MinMaxTiming**
This module defines the attributes that allow setting minimum and maximum bounds on element active duration.

**Module dependencies**
None.
**Included features**
The max and min attributes.
**Other module specific integration requirements**
None.

**MultiArcTiming**
This module extends the attribute value syntax for the begin and end attributes to allow multiple semicolon-separated values. Any combination of the simple begin and end value types provided by the other timing modules included in the profile are allowed.

**Module dependencies**
At least one of: AccessKeyTiming, BasicInlineTiming, EventTiming, MediaMarkerTiming, RepeatValueTiming, SyncbaseTiming, WallclockTiming.
**Included features**
Any combination of the individual begin and end attribute values included in the profile, separated by semicolons.
**Other module specific integration requirements**
None.

**RepeatTiming**
This module defines the attributes that allow repeating an element for a given duration or number of iterations.

**Module dependencies**
None.
**Included features**
The repeatDur, repeatCount , and repeat attributes.
**Other module specific integration requirements**
repeat is deprecated and only requires inclusion in SMIL Host Language conformant profiles.

**RepeatValueTiming**
This module defines the attribute value syntax for begin and end attributes that allow elements to begin and end in response to repeat events with a specific iteration value.

**Module dependencies**
None.
**Included features**
begin and end with repeat values.
**Other module specific integration requirements**
None.

**RestartDefault**
This module defines syntax for specifying default restart semantics for elements.

**Module dependencies**
RestartTiming.
**Included features**
restartDefault attribute.
**Other module specific integration requirements**
None.

**RestartTiming**
This module defines an attribute for controlling the begin behavior of an element that has previously begun.

**Module dependencies**
None.
**Included features**
restart attribute.
**Other module specific integration requirements**
If this module is not included, the integrating profile must define the semantics of attempting to restart and element that has already begun.

**SyncBehavior**
This module defines syntax for specifying the runtime synchronization behavior among elements.

**Module dependencies**
BasicTimeContainers or ExclTimeContainers or TimeContainerAttributes.
**Included features**
syncBehavior, syncTolerance attributes.
**Other module specific integration requirements**
None.

**SyncBehaviorDefault**
This module defines syntax for specifying default synchronization behavior for elements and all descendents.

**Module dependencies**

SyncBehavior.

**Included features**

syncBehaviorDefault, syncToleranceDefault attributes.

**Other module specific integration requirements**

None.

**SyncbaseTiming**

This module defines the attribute value syntax for the begin and end attributes that allow elements to begin and end relative to each other.

**Module dependencies**

None.

**Included features**

begin and end with syncbase values.

**Other module specific integration requirements**

None.

**SyncMaster**

This module defines syntax for specifying the synchronization master for a timeline.

**Module dependencies**

SyncBehavior.

**Included features**

syncMaster attribute.

**Other module specific integration requirements**

None.

**TimeContainerAttributes**

This module defines attributes for adding time container support to any XML language elements.

**Module dependencies**

None.

**Included features**

timeContainer, timeAction, fill and endsync attributes.

**Other module specific integration requirements**

The profile must define on what elements these attributes are legal. `fill=transition` is only supported when BasicTransitions or InlineTransitions is included in the language profile. If FillDefault is not included in the profile, `fill=default` is interpreted the same as `fill=auto`.

**WallclockTiming**

This module the attribute value syntax for the begin and end attributes that allow elements to begin and end relative to real world clock time.

**Module dependencies**

None.

**Included features**

begin and end with wallclock times.

**Other module specific integration requirements**

None.

## 10.8 Appendix B: Annotated examples

### 10.8.1 Example 1: Simple timing within a Parallel time container

This section includes a set of examples that illustrate both the usage of the SMIL syntax, as well as the semantics of specific constructs. This section is informative.

Note: In the examples below, the additional syntax related to layout and other issues specific to individual document types is omitted for simplicity.

All the children of a par begin by default when the par begins. For example:

```
<par>
   <img id="i1" dur="5s"  src="img.jpg" />
   <img id="i2" dur="10s" src="img2.jpg" />
   <img id="i3" begin="2s" dur="5s" src="img3.jpg" />
</par>
```

Elements "i1" and "i2" both begin immediately when the par begins, which is the default begin time. The active duration of "i1" ends at 5 seconds into the par. The active duration of "i2" ends at 10 seconds into the par. The last element "i3" begins at 2 seconds since it has an explicit begin offset, and has a duration of 5 seconds which means its active duration ends 7 seconds after the par begins.

### 10.8.2 Example 2: Simple timing within a Sequence time container

Each child of a seq begins by default when the previous element ends. For example:

```
<seq>
   <img id="i1" begin="0s" dur="5s" src="img1.jpg" />
   <img id="i2" dur="10s" src="img2.jpg" />
   <img id="i3" begin="1s" dur="5s" src="img3.jpg" />
</seq>
```

The element "i1" begins immediately, with the start of the seq, and ends 5 seconds later. Note: specifying a begin time of 0 seconds is optional since the default begin offset is always 0 seconds. The second element "i2" begins, by default, 0 seconds after the previous element "i1" ends, which is 5 seconds into the seq. Element "i2" ends 10 seconds later, at 15 seconds into the seq. The last element, "i3", has a begin offset of 1 second specified, so it begins 1 second after the previous element "i2" ends, and has a duration of 5 seconds, so it ends at 21 seconds into the seq.

### 10.8.3 Example 3: excl time container with child timing variants

1. Exclusive element, children activated via link-based activation.

```
<par>
```

```
        <par id="p1">
        ...
        </par>
        <par id="p2">
        ...
        </par>
    </excl>
    <a href="p1"><img src="Button1.jpg"/></a>
    <a href="p2"><img src="Button2.jpg"/></a>
</par>
```

This example models jukebox-like behavior. Activating the first image hyperlink activates the media items of parallel container "p1". If the link on the second image is traversed, "p2" is started (thereby deactivating "p1" if it would still be active) from time 0.

2. Exclusive element combined with event-based activation:

```
<smil ...>
...
<par>
    <excl>
        <par begin="btn1.activateEvent">
        ...
        </par>
        <par begin="btn2.activateEvent">
        ...
        </par>
    </excl>
    <img id="btn1" src=... />
    <img id="btn2" src=... />
</par>
...
</smil>
```

The same jukebox example, using event-based activation.

3. Exclusive element using scheduled timing:

```
<excl>
    <ref id="a" begin="0s" ... />
    <ref id="b" begin="5s" ... />
</excl>
```

In the example above, the beginning of "b" deactivates "a" (assuming that a is still active after 5 seconds). Note that this could also be modeled using a sequence with an explicit duration on the children. While the scheduled syntax is allowed, this is not expected to be a common use-case scenario.

### 10.8.4 Example 4: default duration of discrete media

For simple media elements (i.e., media elements that are not time containers) that reference discrete media, the implicit duration is defined to be 0. This can lead to surprising results, as in this example:

```
<seq>
   <img src="img1.jpg" />
   <video src="vid2.mpg" />
   <video src="vid3.mpg" />
</seq>
```

The implicit syncbase of a sequence is defined to be the effective active end of the previous element in the sequence. In the example, the implicit duration of the image is used to define the simple and active durations. As a result, the default begin of the second element causes it to begin at the same time as the image. Thus, the image will not show at all! Authors will generally specify an explicit duration for any discrete media elements.

### 10.8.5 Example 5: end specifies end of active dur, *not* end of simple dur

There is an important difference between the semantics of end and dur. The dur attribute, in conjunction with the begin time, specifies the simple duration for an element.

This is the duration that is repeated when the element also has a repeat behavior specified. The attribute end on the other hand overrides the active duration of the element. If the element does not have repeat behavior specified, the active duration is the same as the simple duration. However, if the element has a repeat behavior specified, then the end will override the repeat, but will not affect the simple duration. For example:

```
<html ...>
...
<seq repeatCount="10" end="stopBtn.click">
   <img src="img1.jpg" dur="2s" />
   <img src="img2.jpg" dur="2s" />
   <img src="img3.jpg" dur="2s" />
</seq>
...
</html>
```

The sequence will play for 6 seconds on each repeat iteration. It will play through 10 times, unless the user clicks on a "stopBtn" element before 60 seconds have elapsed.

### 10.8.6 Example 6: DOM-initiated timing

When an implementation supports the DOM methods described in this document, it will be possible to make an element begin or end the active duration using script or some other browser extension. When an author wishes to describe an element as interactive in this manner, the following syntax can be used:

```
<audio src="song1.au" begin="indefinite" />
```

The element will not begin until the `beginElement()` method is called.

## 10.9 Appendix C: Differences from SMIL 1.0

SMIL 1.0 defines the model for timing, including markup to define element timing, and elements to define parallel and sequence time containers. This version introduces some syntax variations and additional functionality, including:

- A new time container for hypermedia interactions
- Additional control over the repeat behavior
- A syntax for interactive (event-based) timing
- Change in constraints on sync-arcs
- A means of specifying a logical time-base relationship
- Support for wall-clock timing
- Support for time manipulations
- Fill is now allowed on time containers as well as "leaf" elements

The complete syntax is described here, including syntax that is unchanged from SMIL 1.0.

## 10.10 Appendix D: Unifying event based and scheduled timing

A significant motivation for SMIL 2.0 is the desire to integrate declarative, determinate scheduling with interactive, indeterminate scheduling. The goal is to provide a common, consistent model and a simple syntax.

Note that "interactive" content does not refer simply to hypermedia with support for linking between documents, but specifically to content within a presentation (i.e. a document) that is *activated* by some interactive mechanism (often user-input events, but including local hyperlinking as well).

SMIL 2.0 describes extensions to SMIL 1.0 to support interactive timing of elements. These extensions allow the author to specify that an element should begin or end in response to an event (such as a user-input event like "activateEvent" or "click"), or to a hyperlink activation, or to a DOM method call.

The syntax to describe this uses event-value specifications and the special argument value "`indefinite`" for the begin and end attribute values. Event values describe user interface and other events. If an element should only begin (or end) with a DOM method call, the begin and end attributes allow the special value "`indefinite`" to indicate this. Setting begin="`indefinite`" can also be used when a hyperlink will be used to begin the element. The element will begin when the hyperlink is actuated (usually by the user clicking on the anchor). It is not possible to control the active end of an element using hyperlinks.

### 10.10.1 Background

SMIL 2.0 represents an evolution from earlier multimedia runtimes. These were typically either pure, static schedulers or pure event-based systems. Scheduler models present a linear timeline that integrates both discrete and continuous media. Scheduler models tend to be good for storytelling, but have limited support for user-interaction. Event-based systems, on the other hand, model multimedia as a graph of event bindings. Event-based systems provide flexible support for user-interaction, but generally have poor scheduling facilities; they are best applied to highly interactive and experiential multimedia.

The SMIL 1.0 model is primarily a scheduling model, but with some flexibility to support continuous media with unknown duration. User interaction is supported in the form of timed hyperlinking semantics, but there was no support for activating individual elements via interaction.

### 10.10.2 Modeling interactive, event-based content in SMIL

To integrate interactive content into SMIL timing, the SMIL 1.0 scheduler model is extended to support several new concepts: *indeterminate timing* and *event-activation*.

With *indeterminate timing*, an element has an undefined begin or end time. The element still exists within the constraints of the document, but the begin or end time is determined by some external activation. Activation may be event-based (such as by a user-input event), hyperlink based (with a hyperlink targeted at the element), or DOM based (by a call to the `beginElement()` or `beginElementAt()` methods). From a scheduling perspective, the time is described as *unresolved*.

The event-activation support provides a means of associating an event with the begin or end time for an element. When the event is raised (e.g. when the user clicks on something), the associated time is *resolved* to a *determinate* time. The actual begin or end time is computed as the time the event is raised plus or minus any specified offset.

The computed time defines the synchronization for the element relative to the parent time container. It is possible for the computed begin or end time to occur in the past, e.g. when a negative offset value is specified, or if there is any appreciable delay between the time the event is raised and when it is handled by the SMIL implementation. See also the section Handling negative offsets for begin.

Note that an event based end will not be activated until the element has already begun. Any specified end event is ignored before the element begins.

The constraints imposed on an element by its time container are an important aspect of the event-activation model. In particular, when a time container is itself inactive (e.g. before it begins or after it ends), no events are handled by the children. If the time container is frozen, no events are handled by the children. No event-activation takes place unless the time container of an element is active. For example:

```
<smil ...>
...
<par begin="10s" dur="5s">
   <audio src="song1.au" begin="btn1.activateEvent" />
</par>
...
</smil>
```

If the user activates (e.g., clicks on) the "btn1" element before 10 seconds, or after 15 seconds, the audio element will not play. In addition, if the audio element begins but would extend beyond the specified active end of the par container, it is effectively cut off by the active end of the par container.

See also the discussion of Event sensitivity.

# 11. The SMIL 2.0 Time Manipulations Module

***Editors:***
Patrick Schmitz (cogit@judicrum.org), (Microsoft)

## 11.1 Introduction

This module introduces new attributes for advanced manipulation of time behavior, such as controlling the *speed* or *rate* of time for an element. These *time manipulations* are especially suited to animation and non-linear or discrete media. Not all continuous media types will fully support time manipulations. For example, streaming MPEG 1 video will not generally support backwards play. A fallback mechanism is described for these kinds of media.

Four new attributes add support for time manipulations to SMIL timing modules, including control over the speed of an element, and support for acceleration and deceleration. The impact on overall timing and synchronization is described. A definition is provided for reasonable fallback mechanisms for media players that cannot support the time manipulations.

An accessibility requirement for control of the playback speed is related to the speed control, but may also be controlled through some other mechanism such as DOM interfaces.

### 11.1.1 Background

*This section is informative*

A common application of timing supports animation. The recent integration of SMIL timing with SVG is a good example of the interest in this area. Animation in the more general sense includes the time-based manipulation of basic transforms, applied to a presentation. Some of the effects supported include motion, scaling, rotation, color manipulation, as well as a host of presentation manipulations within a style framework like CSS.

Animation is often used to model basic mechanics. Many animation use-cases are difficult or nearly impossible to describe without a simple means to control pacing and to apply simple effects that emulate common mechanical phenomena. While it is possible to build these mechanisms into the animation behaviors themselves, this requires that every animation extension duplicate this support. This makes the framework more difficult to extend and customize. In addition, a decentralized model allows any animation behavior to introduce individual syntax and semantics for these mechanisms. The inconsistencies that this would introduce make the authoring model much harder to learn, and would complicate the job of any authoring tool designer as well. Finally, an ad hoc, per-element model precludes the use of such mechanisms on structured animations (e.g. applying time manipulations to a time container of synchronized animation behaviors).

A much simpler model for providing the necessary support centralizes the needed functionality in the timing framework. This allows all timed elements to support this functionality, and provides a consistent model for authors and tool designers. The most direct means to generalize pacing and related functionality is to transform the pacing of *time* for a given element. This is an extension of the general framework for element time (sometimes referred to as "local time"), and of the support to convert from time on one element to time on another element. Thus, to control the pacing of a motion animation, a temporal transform is applied that adjusts the pacing of time (i.e., the *rate of progress*) for the motion element. If time is scaled to advance faster than normal presentation time, the motion will appear to run faster. Similarly, if the pacing of time is dynamically adjusted, acceleration and deceleration effects are easily obtained.

The time manipulations are based upon a model of cascading time. That is, each element defines its active and simple time as transformations of the parent simple time. This recurses from the root time container to each "leaf" in the time graph. If a time container has a time manipulation defined, this will be reflected in all children of the time container, since they define their time in terms of the parent time container. In the following example a sequence time container is defined to run twice as fast as normal (i.e. twice as fast as its respective time container).

```
<seq speed="2.0">
   <video src="movie1.mpg" dur="10s" />
   <video src="movie2.mpg" dur="10s" />
   <img src="img1.jpg" begin="2s" dur="10s">
      <animateMotion from="-100,0" to="0,0" dur="10s" />
   </img>
   <video src="movie4.mpg" dur="10s" />
</seq>
```

The entire contents of the sequence will be observed to play (i.e., to progress) twice as fast. Each video child will be observed to play at twice the normal rate, and so will only last for 5 seconds. The image child will be observed to delay for 1 second (half of the specified begin offset). The animation child of the image will also "inherit" the speed manipulation from the sequence time container, and so will run the motion twice as fast as normal, leaving the image in the final position after only 5 seconds. The simple duration and the active duration of the sequence will be 21 seconds (42 seconds divided by 2).

### 11.1.2 Overview of support

*This section is informative*

Three general time manipulations are defined:

accelerate, decelerate
: These two attributes provide a simple shorthand for controlling the rate of progress of element simple time to simulate common mechanical acceleration and deceleration effects. The model is sometimes presented to authors as *"Ease-In, Ease-Out"*. This model has the advantage that it preserves the simple duration. The model is sometimes presented to authors as *"Ease-In, Ease-Out"*.

autoReverse
: When set to "true" causes the simple duration to be played once forward, and then once backward. This will double the declared or implicit simple duration. Support for autoReverse is often presented to authors as *"Play Forward, then Backward"*.

speed
: Controls the pacing (or *speed*) of element active time. The speed effectively scales the rate of progress of the element active duration, relative to the rate of progress of the parent time container.

*This section is normative*

When the speed of time is *filtered* with any of the time manipulations, this affects how a document time is converted to an element simple time. To understand this, think of the contents of an element *progressing* at a given rate. An unmodified input time value is converted to an *accumulated progress* for the element contents. Element *progress* is expressed as *filtered time*. This allows the effect of any rate (including acceleration and deceleration) to cascade to any timed children. If element progress is advancing at a constant rate (e.g. with a speed value of 2), the filtered time calculation is just the input time multiplied by the rate. If the rate is changing, the filtered time is computed as an integral of the changing rate. The equations used to calculate the filtered time for a given input time are presented in Converting document time to element time.

The accelerate and decelerate features are applied locally on the simple duration, and have no side effects upon the length of the simple duration or active duration of the element. When applied to a time container, accelerate and decelerate apply to the time container simple duration and all its timed children, affecting the observed duration of the timed children.

The autoReverse feature is applied directly to the simple duration, and *doubles* a declared or implicit simple duration. Thus if the simple duration is

defined (according to the normal timing semantics) to be 5 seconds, setting the autoReverse to "true" will cause the simple duration to be 10 seconds. Thus if the active duration is defined in terms of the simple duration (for example by specifying a repeatCount), then autoReverse will also double the active duration. However if the active duration is defined independent of the simple duration (for example by specifying a repeat duration, an end value and/or min and max values), then autoReverse may not affect the active duration. The active duration is computed according to the semantics of the Timing and Synchronization model; the only change is to use the modified (doubled) simple duration value.

The speed attribute scales the progress of element time for the element. When applied to a time container, the contents of the time container collectively play at the scaled speed. If an element plays twice as fast as normal, the observed simple duration will be only half as long as normal. This may or may not affect the active duration, depending upon how it is defined for the element. The attributes repeatDur, min and max are all measured in element active time, and so the associated values will be scaled by the element speed. Similarly, an element defined with a repeatCount will also be scaled, since the simple duration is scaled. However, if an element specifies an end attribute, the end value is not affected by the element speed.

Offset values for the end attribute are measured in parent simple time, and so are excluded from the effects of element speed. Other values (including syncarc-values, event-values, etc.) must be converted to parent simple time, and are similarly unaffected by the element speed.

Note that a speed attribute on an element does not affect the element begin time. Offset values for the begin attribute are measured in parent simple time, and so are excluded from the effects of element speed. (The begin time *is* affected by any time manipulations on the *parent* or other ascendant time containers).

When these time manipulations are applied to a time container, they affect the way that the entire contents of the time container proceeds - i.e. they affect all timed descendents of the time container. As a global time is converted to element time, the time manipulations for each ancestor are applied, starting with the root of the timegraph and proceeding down to the parent time container for the element. Thus the simple time and active time for a given element are ultimately computed as a function of the time manipulations of all ascendant time containers, as well as any time maniuplations defined on the element itself. This is described more completely in Details of timing model arithmetic.

The *net cascaded speed* of an element is a function of any time manipulations on the element and all of its ascendant time containers. Although this can be computed directly from the described values, the speed can also be thought of as the derivative of the rate of time (i.e. the rate of progress) at any point.

*This section is informative*

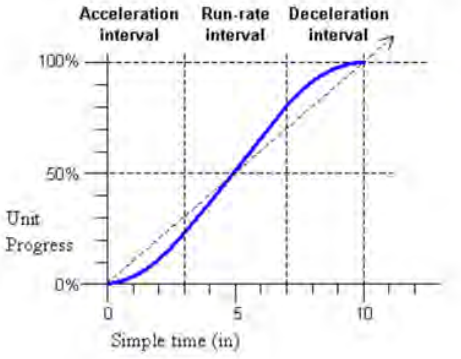
This model lends itself well to an implementation based upon "sampling" a timegraph, with *non-linear* media (also called "random access" media). The time manipulations model is based upon a model commonly used in graphics and animation, in which an animation graph is "sampled" to calculate the current values for the animation, and then the associated graphics are rendered. Some *linear* media players may not perform well with the time manipulations (e.g. players that can only play at normal play speed). A fallback mechanism is described in which the timegraph and syncbase-value times are calculated using the pure mathematics of the time manipulations model, but individual media elements simply play at the normal speed or display a still frame.

Some of the examples below include animation elements such as animate and animateMotion. These elements are defined in the Animation section of SMIL 2.0. Additional elements and attributes related to timing and synchronization are described in the Timing section of SMIL 2.0.

### 11.1.3 Attribute syntax

*This section is normative*

These attributes define a simple acceleration and deceleration of element time, within the simple duration. The values are expressed as a *proportion* of the simple duration (i.e. between 0 and 1), and are defined such that the length of the simple duration is not changed by the use of these attributes. The normal play speed within the simple duration is increased to compensate for the periods of acceleration and deceleration (this is how the simple duration is preserved). The modified speed is termed the *run rate*. As the simple duration progresses (i.e., plays back), acceleration causes the rate of progress to increase from a rate of 0 up to the run rate. Progress continues at the run rate until the deceleration phase, when progress slows from the run-rate down to a rate of 0. This is illustrated in Figure 1, below:




```
<animation dur="10s" accelerate="0.3" decelerate="0.3" .../>
```

**Figure 1: Effect of acceleration and deceleration upon the rate of progress.**

These attributes apply to the simple duration; if these attributes are combined with repeating behavior, the acceleration and/or deceleration occurs within each repeat iteration.

accelerate = " *number* "
Defines a simple acceleration of time for the element. Element time will accelerate from a rate of 0 at the beginning up to a *run rate*, over the course of the specified proportion of the simple duration.
The default value is 0 (no acceleration).
Legal values are floating point values between 0 and 1 (inclusive).

decelerate = " *number* "
Defines a simple deceleration of time for the element. Element time will decelerate from a *run rate* down to 0 at the end of the simple duration, over the course of the specified proportion of the simple duration.
The default value is 0 (no deceleration).

Legal values are floating point values between 0 and 1 (inclusive).

The sum of accelerate and decelerate must not exceed 1. If the individual values of the accelerate and decelerate attributes are between 0 and 1 and the sum is greater than 1, then both the accelerate and decelerate attributes will be ignored and the timed element will behave as if neither attribute was specified.

If the simple duration of the element is not resolved or if it is resolved to be indefinite, then the accelerate and decelerate attributes are ignored.

For details of computing the run-rate, and for converting from parent simple time to element simple time when accelerate and/or decelerate are specified, see Computing the element run-rate and Converting document time to element time.

*Examples*

In this example, a motion path will accelerate up from a standstill over the first 2 seconds, run at a faster than normal rate for 4 seconds, and then decelerate smoothly to a stop during the last 2 seconds. This makes the motion animation look more realistic.

```
<img ...>
    <animateMotion dur="8s" accelerate=".25" decelerate=".25" .../>
</img>
```

In the following example, the image will "fly in" from offscreen left, and then decelerate during the last second to "ease in" to place. This assumes a layout model that supports positioning (for example CSS positioning, or the position of a region in SMIL Layout).

```
<img ...>
    <animate attributeName="left" dur="4s" decelerate=".25"
             from="-1000" to="0" additive="sum" />
</img>
```

**The autoReverse attribute**

Another common mechanical phenomenon is that of a process that advances and reverses. Some examples include:

- pendulum motion - a partial rotation that advances and reverses
- pulsing effects - usually a scale transform, but sometimes an intensity or color change that advances and reverses
- simple bouncing - motion that advances and reverses

Because so many common use-cases apply repeat to the modified local time (as in the examples above), this function is modeled as modifying the simple duration. As such, autoReverse doubles the declared or implicit simple duration. When combined with repeating behavior, each repeat iteration will play once forwards, and once backwards.

autoReverse = " `true | false` "
Controls autoReverse playback mode.
Argument values are Booleans.
The default value is false (i.e. play normally).

When this is applied to a time container, it will play the time container forwards and then backwards. The semantics of playing a time container backwards are detailed in Implications of time manipulations on time containers.

The autoReverse time manipulation does not initially require a resolved simple duration, although it will not begin playing backwards until the simple duration is resolved and has completed. This can happen when the simple duration is defined by an unresolved implicit simple duration (such as the intrinsic media duration). In this case, the element will continue to play forward until the implicit simple duration is resolved (or until the active duration or the parent time container cuts short the simple duration, as described in the Timing section of SMIL 2.0). If the implicit simple duration becomes resolved before the end of the active duration, then the simple duration will be resolved to 2 times this implicit duration, and the implicit simple duration will play backwards.

Any time that the element will play backwards, including the second part of the autoReverse manipulation, the simple duration must be resolved. See also The speed attribute.

In this example, a motion path will animate normally for 5 seconds moving the element 20 pixels to the right, and then run backwards for 5 seconds (from 20 pixels to the right back to the original position). The repeating behavior causes it to repeat this 2 more times, finally leaving the element at its original location. The computed simple duration of the animation is 10 seconds, and the active duration is 30 seconds.

```
<img ...>
    <animateMotion by="20, 0" dur="5s" autoReverse="true" repeatCount="3"/>
</img>
```

In the following example the motion path will behave as above, but will end at the earlier of 15 seconds or when the user clicks on the image. If the element ends at 15 seconds (if the user does not click), the motion path will leave the element at the end of the defined path, 20 pixels to the right. Since the active duration is defined by the repeatDur and end, the active duration is not affected by the autoReverse attribute in this case. The semantics of fill are not changed by time manipulations: the media state at the end of the active duration is used during any fill period. The end of the active duration may fall part of the way through a *play forward* interval, or part of the way through a *play backward* interval.

```
<img ...>
    <animateMotion  by="20, 0" dur="5s" autoReverse="true"
          repeatDur="15" end="click" fill="freeze"/>
</img>
```

The accelerate and decelerate attributes can be combined with autoReverse, and are applied to the unmodified simple duration. For example:

```
<img ...>
    <animateMotion by="20, 0" dur="4s" autoReverse="true"
      accelerate=".25" decelerate=".25" />
</img>
```

This will produce a kind of elastic motion with the path accelerating for 1 second from the original position as it moves to the right, moving slightly faster than normal for 2 seconds, and then decelerating for 1 second as it nears the points 20 pixels to the right. It accelerates back towards the original position and decelerates to the end of the reversed motion path, at the original position.

The following example of a rotation (based upon the SVG *animateTransform* element) also demonstrates the combination of accelerate, decelerate and autoReverse. It will produce a simple pendulum swing on the target (assume that the target is a pendulum shape with the transform origin at the top):

```
<animateTransform type="rotate" from="20" to="-20" dur="1s" repeatCount="indefinite"
      accelerate=".5" decelerate=".5" autoReverse="true" ... />
```

The pendulum swings through an arc in one second, and then back again in a second. The acceleration and deceleration are applied to the unmodified simple duration, and autoReverse plays this modified simple duration forwards and then backwards. The effect is to accelerate all the way through the downswing, and then decelerate all through the upswing. The autoReverse feature then makes the same animation (i.e. the simple duration) play in reverse, and the pendulum swings back to the starting position. The entire modified simple duration repeats, producing continuous back and forth animation. This produces a realistic looking animation of real-world pendulum motion.

### The speed attribute

The speed attribute controls the local playback speed of an element, to speed up or slow down the effective rate of play relative to the parent time container. The speed attribute is supported on all timed elements. The argument value does not specify an absolute play speed, but rather is relative to the playback speed of the parent time container. The specified value cascades to all time descendents. Thus if a par and one of its children both specify a speed of 50%, the child will play at 25% of normal playback speed.

speed = " *number* "
Defines the playback speed of element time.
Legal values are signed floating point values. A value of 0 is not allowed (and will be ignored)
The default is "1.0" (no modification of speed).

Values less than 0 are allowed, and cause the element to play backwards. An element can only play backwards if there is sufficient information about the simple and active durations. Specifically:

- The element simple duration must be resolved and may not be indefinite. If the simple duration is defined by the implicit duration or by an explicit endsync attribute (for time containers), then all children considered in the endsync semantic must have a resolved begin time and a resolved active duration that is not indefinite.
- The element active duration must be resolved and not indefinite, OR the element active duration must be constrained by a resolved simple duration for its associated time container. There must be a means of defining active time running backwards.

If the cascaded speed value for the element is negative and if either of the above two conditions is not met, the element will begin and immediately end (i.e. it will behave as though it had a specified active duration of 0). If there is a min attribute specified, the time container will simply be frozen at the initial state for the specified minimum duration.

The details of the effect of the element speed upon the timing calculations are described in Details of timing model arithmetic.

### Examples

The following motion animation will move the target twice as fast as normal:

```
<animateMotion dur="10s" repeatCount="2" speed="2.0" path= ... />
```

The target will move over the path in 5 seconds (simple dur/speed = 10s/2.0 = 5s), and then repeat this motion (because repeatCount is set to 2). The active duration is thus 10 seconds.

When speed is applied to a time container, it scales the rate of progress through the time container timeline. This effect cascades. When descendents also specify a speed value, the parent speed and the child speed are multiplied to yield the result. For example:

```
<par speed=2.0>
   <animate begin="2s" dur="9s" speed=0.75 .../>
</par>
```

The observed rate of play of the animate element is 1.5 times the normal play speed (2.0 * 0.75 == 1.5). The element begins 1 second after the par begins (the begin offset is scaled only by the parent speed), and ends 6 seconds later (dur/speed = 9/1.5 = 6).

The following example shows how an event based end combines with time manipulations:

```
<par speed=2.0>
   <animate begin="2s" dur="9s" speed=0.75
        repeatCount="4" end="click" .../>
</par>
```

This behaves as in the first example, but the animate element will repeat 4 times for an observed time of 24 seconds (dur/cascaded speed = 9s/(2.0 * 0.75) = 6s, and 6s * 4 repeats = 24s). If a click occurs before this, the element ends at the time of the click. A variant on this demonstrates syncbase timing:

```
<par speed=2.0>
   <img id="foo" dur="30s" .../>
   <animate dur="9s" speed=0.75
        repeatCount="4" end="click; foo.end" .../>
</par>
```

The image will display for 15 seconds. The animate element plays at an observed rate of 1.5 times play speed (2.0 * 0.75), but it will end after 15 seconds, when the image ends. The observed simple duration will be 6 seconds long (9 seconds divided by the cascaded speed 1.5). The animation will repeat 2.5 times during the active duration. Note that although the animation has a speed value, this does not impact the semantic of the syncbase timing. When the syncbase, eventbase, wallclock or media marker time is observed to happen, it will be applied anywhere it is used at that actual time (although conversions are applied internally, e.g. from syncbase element active time to parent simple time - see also Converting document time to element time).

Note that in the examples above, the default duration of the par container is defined as endsync="last". This behavior is not affected by the speed modification in the manner that the slow speed of the element will cause the parent simple duration and the parent time container

The following example illustrates an important effect of offset time scaling:

```
<par speed=2.0>
   <img id="foo" dur="30s" .../>
   <animate begin="2s" dur="9s" speed=0.75
        repeatCount="4" end="foo.end+6s" .../>
</par>
```

The image will display for 15 seconds, as above. The animate element begins at 1 second, since the begin offset is scaled by the parent time container speed, but not by the element speed. The animate element will end at 18 seconds (15 seconds plus 6 seconds divided by the time container speed of 2.0). The "6s" offset added to "foo.end" is scaled by the parent time container speed, but not by the element speed.

### 11.1.4 Details of timing model arithmetic

*This section is normative*

**Timing and real-world clock times**

When the time manipulation attributes are used to adjust the speed and/or pacing within the simple duration, the semantics can be thought of as changing the pace of time in the given interval. An equivalent model is these attributes simply change the pace at which the presentation *progresses* through the given interval. The two interpretations are equivalent mathematically, and the significant point is that the notion of "time" as defined for the element simple duration should not be construed as real world clock time. For the purposes of SMIL Time manipulations (as for SMIL Timing and Synchronization), "time" can behave quite differently from real world clock time.

**Common definitions**

In the following discussion, several symbols are used as shorthand:

Let **a** be the value of accelerate, and **b** be the value of decelerate. Both take on (floating point) values 0 to 1, and will not sum to more than 1.

Let **dur** be the value of the simple duration as defined by the Timing and Synchronization model. This is the actual simple duration, and not simply the dur attribute. This value does not account for the effect of any time manipulations.

the unmodified simple duration. Note that with the described model for acceleration and deceleration, the observed duration during which time accelerates and/or decelerates may be greater than **dacc** and **ddec** respectively.

**dacc = dur * a**

**ddec = dur * d**

**Computing the element run-rate**

In order to preserve the simple duration, the speed through the simple duration must be increased to account for acceleration and deceleration. To compute the run rate over the course of the simple duration, the following formula is used. The run rate **r** is then:

**r = 1 / ( 1 - a/2 - b/2 )**

Thus, for example, if the value of accelerate is 1 (i.e. accelerate throughout the entire simple duration), the run rate is 2 (twice the normal play speed).

**r(t)** is the speed modification due to acceleration and deceleration, at any time **t** within the simple duration. The parameter time **t** must *not* already be modified to account for acceleration and deceleration. In the terms of the discussion below, Converting document time to element time, the parameter time **t** is in the $t_{su}$' space. The speed modification is defined as a function of the run rate **r**, as follows:

In the acceleration interval, where ( `0 <= t < dacc` )

**r(t) = r * ( t / dacc )**

In the run-rate interval, where ( `dacc <= t <= ( dur - ddec )` )

**r(t) = r**

In the deceleration interval, where ( `( dur - ddec ) < t <= dur` )

**r(t) = r * ( dur - t ) / ( ddec )**

The run-rate only describes the modification applied to account for any acceleration and deceleration. This is combined with any element speed, as well as the speed inherited from the parent time container. The combined or "net" speed is defined in the section Computing the net cascaded speed for an element.

**Converting document time to element time**

To convert a document time to an element time, the original time is converted to a simple time for each time container from the root time container down to the parent time container for the element. This recursive algorithm allows for a simple model of the conversion from parent simple time to element active and element simple time. The first step calculates element active time, and the second step calculates element simple time.

These steps are based upon a simpler, general model for time conversion that applies to the timing model independent of the time manipulations functionality (see also the Timing section of SMIL 2.0). The steps below describe the modified arithmetic for converting times, taking into account the semantics of time manipulations.

The steps below assume that the associated times are resolved and not indefinite. If a required time is not resolved or is indefinite, then the conversion is not defined, and cannot be performed.

*Filtered active time calculation*

In order to reflect the semantics of element speed, the element active time must be adjusted. The adjusted time is called the *filtered active time*, and is used by the element where the timing semantics refer to "element active time". The autoReverse and accelerate /decelerate attributes only affect the computation of the filtered simple time, and so do not come into play in this step.

The input time is a time in parent simple time. This is normalized to the element active duration, adjusting for the accumulated synchronization offset (described in The accumulated synchronization offset).

Let $t_{ps}$ be a time in parent simple time, and **B** be the begin time, and **O** be the accumulated synchronization offset for an element, measured in parent simple time.

The unfiltered active time $t_{au}$ for any child element is:

$t_{au} = t_{ps} - B - O$

Given an unfiltered active $t_{au}$, the filtered active time $t_{af}$ is only a function of the **speed** for the element (this is the value specified in a speed attribute, or the default, and not the net cascaded speed):

If( **speed** > 0 ) *i.e. if the local speed is forwards*

$t_{af} = t_{au} * speed$

Else *i.e. if the local speed is backwards*

$t_{af} = AD - t_{au} * ABS( speed )$

As expected, if the speed value is 1 (the default), this is an identity function, and so $t_{af} = t_{au}$. When speed is less than 0 (in the backwards direction), the active duration proceeds from the end of the active duration towards 0.

*Filtered simple time calculation*

In order to reflect the semantics of the autoReverse and accelerate /decelerate attributes, the element simple time must be adjusted. The adjusted time is called the *filtered simple time*. The filtered simple time is defined as a function of the filtered active time, and so reflects all the time manipulations on an element.

The element simple time is the time that is used to establish runtime synchronization for a media element, or to compute an animation function's input value or sampling time. If the element is a time container, this is also the time that is seen by all children of a time container (as the time container element's simple time).

The input time is a filtered active time $t_{af}$.

Let **dur'** be the modified simple duration that accounts for the effect of the autoReverse attribute. It is computed as follows:

If autoReverse is false:

**dur' = dur**

Else (if autoReverse is true)

The steps to compute the filtered simple time are described below.

1. Compute the unfiltered simple time $t_{su}$, accounting for any repeat behavior.

   If there is no repeating behavior:

   $t_{su} = t_{af}$

   Else, if the modified simple duration **dur'** is fixed and does not vary (ideal case):

   $t_{su} = REMAINDER( t_{af}, dur' )$

   where **REMAINDER( t, d )** is defined as **(t - d*floor(t/d))**.

   Else, if the modified simple duration **dur'** varies from repeat iteration to repeat iteration, or if it is unknown, then the unfiltered simple time is just computed from the begin time of the most recent iteration - call this $t_{last\text{-}repeat}$. Some other mechanism (such as endsync logic or a media player) must note when the simple duration ends, and reset the value of $t_{last\text{-}repeat}$. If the element has not yet repeated, a value of 0 is used in place of $t_{last\text{-}repeat}$.

   $t_{su} = t_{af} - t_{last\text{-}repeat}$

2. Account for autoReverse behavior. If autoReverse is false, $t_{su}' = t_{su}$.

   Else if autoReverse is true (note that the following steps use the unmodified duration **dur**, and not **dur'**):

   If $t_{su} < dur$

   $t_{su}' = t_{su}$

   Else ( $t_{su} >= dur$ )

3. Account for acceleration and/or deceleration behavior. This takes as input $t_{su}'$ (the result of steps 1 and 2).

   The filtered simple time $t_{sf}$ is computed as a function of the input time $t_{su}'$ and the run rates in effect over the interval from 0 to $t_{su}'$. The filtered simple time is the *accumulated progress* up to the input time, and is computed as the integral of the acceleration, run-rate and deceleration rates. Since the rate of acceleration and deceleration are constant, the integral simplifies to a function of the *average rate of progress* for each of the three intervals defined by the acceleration and deceleration values. The steps below compute the filtered time by multiplying the input time and the average rates of progress. In the acceleration interval, since acceleration is constant and the rate changes from 0 to **r**, the average rate is just 1/2 of the instantaneous rate **r(t)** defined above:

   average rate = **(r(t) + r(0) ) / 2) = r(t)/2**

In the deceleration interval, the average rate is similarly computed. In the run-rate interval the rate is constant, and so the average rate is equal to the run-rate.

In the acceleration interval, where ( **0 <= $t_{su}$' < dacc** ), the filtered simple time is the input time multiplied by the average run-rate during the acceleration interval:

$$t_{sf} = t_{su}' * r(t_{su}') / 2$$

In the run-rate interval, where ( **dacc <= $t_{su}$' <= ( dur - ddec )** ), the filtered simple time is computed from the input time and the rates in the acceleration and run-rate intervals. This adds the accumulated progress in the acceleration interval to the progress within the run-rate interval:

$$t_{sf} = dacc * r / 2 + ( t_{su}' - dacc ) * r$$

$$= r * ( dacc / 2 + ( t_{su}' - dacc ))$$

$$= r * ( t_{su}' - dacc / 2 )$$

In the deceleration interval, where ( **( dur - ddec ) < $t_{su}$' <= dur** ), the filtered simple time is computed from the input time and the rates in all three intervals. This sums the total progress in the acceleration interval, the total progress within the run-rate interval, and the progress within the deceleration interval.

To simplify the expressions, we define **tdec,** the time spent in the deceleration interval:

$$tdec = t_{su}' - ( dur - ddec )$$

We also define the proportional duration within the deceleration interval as:

$$pd = tdec / ddec$$

The filtered time within the deceleration interval is then:

$$t_{sf} = dacc * r / 2 + ( dur - dacc - ddec ) * r + tdec * (( r + r*(1 - pd )) / 2 ))$$

$$= r * ( dur - dacc / 2 - ddec + tdec * ( 2 - pd ) / 2 )$$

**Converting element time to document time**

To convert from one element timespace to another, the time for the first element $t_{e1}$ must first be converted to a simple time on the closest ascendant time container that contains both elements. Converting from an element time to the parent time reverses the process described above. Again, it is recursive, and so the conversions are described generically from element simple to element active time, and from element active to parent simple time.

To convert from element simple time to element active time requires the begin time of the most recent iteration, $t_{last\text{-}repeat}$. If the element does not repeat or has not yet repeated, a value of 0 is used in place of $t_{last\text{-}repeat}$.

$$t_a = t_s + t_{last\text{-}repeat}$$

Conversion from element active time to parent simple time uses the associated begin of the element and the accumulated synchronization offset.

$$t_{ps} = t_a + B + O$$

Note that the pure conversions do not take into account the clamping of active durations, nor the effects of fill (where time is frozen). Global to local time conversions used to translate between timespaces must ignore these issues, and so may yield a time in the destination local timespace that is well before or well after the simple duration of the element.

An alternate form of the conversion is used when actually sampling the time graph. A time container is only sampled if it is active or frozen, and so no times will be produced that are before a time container begins. If the global to local time conversion for a time container yields a time during which the time container is frozen, the time is clamped to the value of the active end.

**Computing the net cascaded speed for an element**

The net cascaded speed for a given element at a given point in time can be used to set the correct playback rate for a media element. It is not otherwise used directly in the time manipulations model.

To compute the net cascaded speed $speed_{nc}(t)$ for an element at a given point in time, we combine the net cascaded parent speed at the point in time $speed_{nc\text{-}parent}(t)$ with the element speed value **speed** and the instantaneous run rate **r(t)** computed from any acceleration and deceleration. If the element has no time parent, use **1** for $speed_{nc\text{-}parent}(t)$.

Note that the net cascaded parent speed will be computed in simple time for the parent, and so the element simple time will have to be converted to a parent simple time. This is described above in Converting element time to document time.

The parameter time value must be in the range of the simple duration. The time value must *not* already be modified to account for acceleration and deceleration. In the terms of the discussion above, Converting document time to element time, the parameter time is in the $t_{su}$' space.

The net cascaded speed $speed_{nc}(t)$ for a given unfiltered simple time $t_{su}$' is then:

$$speed_{nc}(t_{su}') = speed_{nc\text{-}parent}(t_{su}') * speed * r(t_{su}')$$

This definition is recursive up to the root of the time containment hierarchy, and so accounts for any speed settings on the parent and all other ascendant time containers.

## 11.1.5 Media fallback semantics

A theoretical model can be described that assumes that all element local timelines (including any media elements) are completely non-linear and have unconstrained ballistics (i.e. they can be sampled at any point at any moment, and can adjust the observed playback rate instantaneously). This ideal model can be applied to many applications, including pure rendered graphics, text, etc. Nevertheless, many common applications also include media with linear behavior and other limitations on playback. When the timegraph includes media elements that have linear behavior, the time manipulations model must accommodate these real world limitations.

While the model does *support* timegraphs with a mix of linear and non-linear behavior, and defines specific semantics for media elements that cannot support the ideal non-linear model, it is *not* a goal to provide an ideal alternative presentation for all possible timegraphs with such a mix. It is left to authors and authoring tools to apply the time manipulations in appropriate situations. This section describes both the ideal model as well as the semantics associated with linear-media elements.

**Ideal model**

*This section is informative*

In the ideal model, the pace or speed of local time can be manipulated arbitrarily. The graph advances (or is sampled, depending upon your perspective) as the presentation time advances. A time container samples each of its children in turn, so that a graph traversal is performed for each render time. Elements that are neither active nor frozen may be pruned from the traversal as an optimization. As the traversal moves down the graph (from time containers to children), each local timeline simply transforms the current time from the parent time-space to the local time space, and then samples the local timeline at the transformed current time. Note that the speed and effects of the time filters effectively cascade down the time graph, since each element transforms element time and element speed for itself and all descendents.

This is the model that is described by the arithmetic model in Details of timing model arithmetic.

When linear media are added to this model and the "current time" (sample) traversal encounters a media element, the media element is effectively told to "sample" at a particular position and a particular rate. Given that linear media can not sample arbitrarily (i.e., they cannot immediately seek to and display an arbitrary frame or sample) , the media element player may not be able to match the ideal model.

Many media elements cannot play off-speed (i.e. at other than normal play speed), and so must simply ignore the requested speed. As the element plays, it will fall out of sync with the sync relationship defined in the timing syntax. Within the limits defined by the SMIL syncTolerance attribute, divergence from the theoretical timeline position may be ignored. However, for further divergence beyond this tolerance the element will be considered out of sync; if the element is defined with `syncBehavior="locked"`, the playback engine will try to enforce the runtime synchronization semantics (and this media will likely wish to specify the syncBehavior for the linear media as "canSlip".

**Fallbacks for time filters on a media element**

*This section is normative*

The fallback semantics depend upon how much or how little the media player is capable of. Some players for some media may play forwards and backwards but only at the normal rate of play, others may only support normal forward play speed.

If the element speed (i.e. the cascaded value) is not supported by the media element, the media should play at the closest supported speed ("best effort"). If the element cannot play slower or faster than the normal play speed, the closest supported speed will be the normal play speed.

In any case, the computed simple duration, *as modified by* the time filters, is respected.

- If the computed simple duration is shorter than the intrinsic media duration at the fallback rate the media is cut short. This is just as for a dur value that overrides the intrinsic media duration in normal SMIL timing. As an example, if the speed is greater than 1 and the media plays at normal speed, then the modified simple duration will play faster and so will be observed to be shorter. If the media cannot play faster, then less of the media will be seen.
- If the computed simple duration is longer than the intrinsic media duration at the fallback rate, the media will freeze the last frame until the end of the computed simple duration (again, just as it does for dur in the normal timing framework). As an example, if the speed is less than 1 and the media plays at normal speed, then the modified simple duration will play slower and so will be observed to be longer. If the media cannot play slower and the intrinsic media duration is reached before the computed simple duration ends, the ending state of the media will be shown from the end of the intrinsic media duration until the end of the computed simple duration.

The semantics of clipBegin and clipEnd are not affected by time manipulations. The clipBegin and clipEnd semantics are always interpreted in terms of *normal forward* play speed. They are evaluated before any effects of time filters have been applied to the time model. This is consistent with the model that they can be evaluated by the media element handler, independent of the time model.

*Authoring considerations for the fallback semantics*

*This section is informative*

In this fallback model, some media elements may not be able to play at the computed speed defined by the time graph. The fallback semantics may cause the media element to fall out of visual synchronization with respect to the rest of the timegraph. For example, if an image element is defined to begin 10 seconds after a video element begins, and then a speed of 2.0 is applied to a time container for both elements, the image element will begin at 5 seconds (10s/2.0). If the video cannot play at twice the normal speed, it will not be 10 seconds into the video when the image shows up, and so the presentation may appear to be out of sync.

When time manipulations are used with linear media, authors can use media-marker-values to define the sync relationships. This can help to maintain the "visual" sync regardless of the fallback behavior. Since the media-marker-values are defined to happen when a particular point in the media is played, this timing will respect the actual behavior of the media, rather than the computed speed behavior.

**Implications of time manipulations on time containers**

*This section is normative*

The time manipulations can apply to any element, including time containers. There are two primary implications of this for the time model:

The rate (relative to normal play speed) at which time proceeds in the parent can affect the observed begin and end times for children of the time container. In the process of converting times from one element's time space in the graph to another element's time space, all time manipulations on ascendant time containers must be respected. Thus for example, if a time container has acceleration defined, children that are defined to begin with simple offset conditions will be observed (in real time) to begin later (than the specified offset), since time moves more slowly at the beginning of the parent simple duration. See also Details of timing model arithmetic.
When a time container is defined to play backwards, the begin and end times for the children must be calculated with a modified algorithm (described immediately below). This is not defined (only) relative to the speed or autoReverse attributes on the time container. Rather, when the net, cascaded speed for the time container is less than zero, the modified semantics are applied. See also Computing the net cascaded speed for an element.

*Handling negative speeds on time containers*

The following discussion is based upon the semantics of begin and end instance lists and the interpretation of lists of begin and end times described in the Timing section of SMIL 2.0

If the time container can play backwards (based upon the general constraints for backwards play upon the simple and active durations), then the children must play the defined intervals in reverse order. This is accomplished with the following modified life cycle for child elements. In the following description, the terms "begin" and "end" for intervals are used relative to the normal play direction. When used as a verb ("the interval begins"), begin and end refer to the current interval becoming active and inactive, respectively. Intervals are described as playing from the *end* of the interval to the *beginning*, and so they *begin at the interval end*, and *end at the interval begin*.

1. **Find first interval to play:** The child element will compute the first interval to play, which is actually the *last* interval defined by the instance times. This is done by considering the instance times lists in the normal order from earlier to later, calculating the *last interval that begins before* the end of the time container simple duration. The same basic approach is used for this as is used to determine the first interval for normal play, and accounts for restart semantics given the known times in the instance lists. One way to think of this is that with respect to the child element, the parent simple time is advanced from 0 to the end of the simple duration, and the last interval found is the one to use. No intervals are created in this process, until the correct times are found, and then one current interval is defined, and has side effects just the same as during normal play: all time-dependents are notified, and will create new instance times associated with this current interval.
2. **Wait to play interval:** As time proceeds from the end of the time container simple duration towards 0, the current interval may wait to be active. The current interval will become active when the time container simple time reaches the defined end time of the current interval. During the period that the current interval is waiting to begin, any fill effect must be applied. This is defined the same as for normal forward play, using the element state at the defined end of the current interval.
3. **Actively playing:** When the current interval becomes active, a beginEvent event is raised. The event is raised to indicate that the element has become active, even though it has become active at the end of the interval. The current interval will remain active until the parent simple time reaches the begin time of the current interval. **Once the current interval has become active, any changes to instance times will have no impact upon the current interval.** This is slightly different from the normal mechanism, but provides a significant simplification to the model without significant loss of functionality. Note that this semantic is only applied when the parent time container of an element is playing backwards.
4. **Ended - get next interval:** When the current interval becomes inactive, an endEvent event is raised. The event is raised to indicate that the element has become inactive, even though it has become inactive at the begin of the current interval. At this point, the element considers the instance times lists and calculates a next interval (if there is one). The same approach is used as in phase 1 above, except that the next interval must begin before the (just ended) current interval begins. In addition, the end of the next interval is constrained according to the restart semantics to end no later than the begin of the (just ended) current interval. Thus the next interval (if there is one) will begin before the begin of the current interval, and will end no later than the begin of the current interval. It is possible that this interval will be defined to begin before the parent begin.
5. **No more intervals:** Once all defined intervals between the end and the beginning of the parent simple duration have been played, the current life cycle for the child element is complete.

When a time container is defined to play backwards, a child element may define additional time manipulations that affect the speed, or even the direction of play. Any such additional time manipulations on the child element do not impact the model described above.

The life cycle is restarted each time the parent (or any ascendant) time container repeats or restarts, just as for the normal play direction.

Because of the reversed evaluation of intervals, some cyclic time dependencies that would correctly propagate forwards when played normally will not propagate correctly when played backwards. The rules for evaluating and halting cycles in the time graph are unchanged by the semantics of time manipulations.

# 12. The SMIL 2.0 Transition Effects Module

***Editors***
Eric Hyche (ehyche@real.com), (RealNetworks)
Debbie Newman (debbien@microsoft.com), (Microsoft)

## 12.1 Introduction

In most public descriptions of SMIL, the language is described as "allowing authors to bring TV-like content to the Web." However, one aspect of presentations commonly seen on television has been noticeably absent from SMIL: transitions such as fades and wipes. In SMIL 1.0, any representation of transitions had to be "baked into" the media itself and there was no method of coordinating transitions across multiple media elements according to the timing framework of SMIL 1.0. The purpose of this section is to specify the semantics and syntax for describing transitions within SMIL and other XML-based documents. Also, this specification describes a taxonomy of transitions based on SMPTE 258M-1993 [SMPTE-EDL] as well as a compact set of parameters which can be used to express this set of transitions.

Consider a simple still image slideshow of four images, each displayed for 5 seconds. Using SMIL Timing, this slideshow might look like the following:

```
...
<seq>
    <img src="butterfly.jpg" dur="5s" ... />
    <img src="eagle.jpg"     dur="5s" ... />
    <img src="wolf.jpg"      dur="5s" ... />
    <img src="seal.jpg"      dur="5s" ... />
</seq>
...
```

Currently when this presentation plays, we see a straight "cut" from one image to another, as shown in this animated image. However, what we would like to see are three left-to-right wipes in between the four images: in between butterfly.jpg and eagle.jpg at 5 seconds, in between eagle.jpg and wolf.jpg at 10 seconds, and in between wolf.jpg and seal.jpg at 15 seconds. This is illustrated by this animated image. The purpose of this document is to define the syntax and semantics of specifying transitions such as these in XML-based documents.

Although the transitions described in this document are *visual* transitions, the concepts apply to *audio* transitions as well by focusing on the overlap of the audio media in time rather than overlap in the layout. However, this document does not define any audio transition effects or specifically address how audio transitions should behave.

This document is organized as follows.

- Transition Model discusses the underlying model for transitions.
- Transition Taxonomy defines a taxonomy of transitions based on SMPTE 258M-1993.
- BasicTransitions Module defines a style-like shorthand method of specifying transitions.
- InlineTransitions Module enables a much finer level of control over transitions.
- TransitionModifiers Module provides additional control over the visual appearance of a transition.
- Integration issues provides considerations for integrating SMIL Transitions with other XML-based languages.
- Appendix: Taxonomy Table provides a detailed taxonomy table.

## 12.2 Transition Model

Transitions are modeled as animated filter behaviors. When a transition module is included in a language profile, all elements with renderable content implicitly have the *transition filter behavior* added to them. By default the behavior has no effect, but attributes and elements are provided to specify and control the effect of the transition behavior on the renderable content. Renderable content is declared in the SMIL Media Object Modules using media elements. Other languages, such as HTML, provide additional elements such as the span and div for rendering. In this document the terms "media element" and "media object" include all "renderable content", defined by the host language.

The transition filter behavior uses the background as one input. In this context, the background is whatever is currently present in the layout where the transition will be applied. Therefore, the background might include actively changing media, frozen media, or solid background colors. It also takes as input the media object to which the transition will be applied. The media object can be used as either the source or the destination input, with the background supplying the other input. The media object also defines the area in which the transition will occur. Certain transitions, such as fade-in from a solid color, will only take one input - the media object to which the transition is applied.

A free parameter common to all transition filter behaviors is the progress through the simple duration of the transition effect, which is abstractly considered to be the progress through the filter effect. We establish the convention that progress is a real number in the range 0.0-1.0, where a progress of 0.0 implies that the output of the filter is completely the background and where a progress of 1.0 implies the output of the filter is completely the destination media. Values in between result in an application of the transition filter behavior that combines the background and destination media in some manner. All other parameters of the transition are assumed to be part of the filter effect itself. *Progress is the only parameter which is animated. Other parameters are used to specify the filter effect, but are not animated.*

The distinction between animating only the progress of the filter versus animating one or more properties of the media is illustrated by the following. In the left-to-right wipe in the Introductory example, we could either think of this transition as:

1. A filter which clips the destination media. The left side of the clipping rectangle would always be coincident with the left side of the destination media and the right side of the clipping rectangle would vary. Therefore, this transition could be thought of as animating the right side of the clipping rectangle.
2. A filter which is predefined to be a left-to-right wipe whose progress varies in the range 0.0-1.0. When progress is 0.0, the background is shown. When progress is 0.5, the left half of the destination media is shown and the right half of the background is shown. When progress is 1.0, all of the destination media is shown.

This may seem to be a very minor distinction for a left-to-right wipe, but then think of the corresponding distinction for a cross-fade. We could think of a cross-fade transition as:

1. Animation of an "opacity" property of both the destination media and the background; or
2. Animation of progress in a filter which knows that a progress of 0.5 means a 50% blend, a progress of 0.75 means 75% of the destination media and 25% of the background, etc.

In some cases, it may seem convenient to think of animating a particular property unique to each type of transition. However, that model does not generalize well across the broad variety of transitions currently in use today. Therefore, in order to maintain simplicity of this model, we think of both the left-to-right wipe and the cross-fade as "black boxes" which both take the same inputs - the background, destination media, and the progress value.

XML elements and attributes are provided to control the properties of the transition. However, the transitions themselves are not a property of the attribute or elements used to control the transition behavior. In the model, the transitions are a behavioral property of the media element itself.

*Transitions are hints to the presentation.* Implementations must be able to ignore transitions if they so desire and still play the media of the presentation. This is equivalent to saying that the transition filter behavior does not execute, or has no effect. Transitions do not alter the active duration of the media elements that are involved in the transition. The transition behaviors operate within the active duration of their respective media elements. The behavior of multiple simultaneous transitions active on an element at a time is undefined.

We will introduce two methods of specifying transitions:

1. *"Style-like" transition shorthand.* In this case, the author defines a set of transition classes and then sets the transition filter behavior to one of these classes by using attributes on the media elements. The same transition class may be applied several times to different media via the transIn and transOut attributes as specified in the BasicTransitions module. Additionally, each of the transitions plays in a default manner; that is, the progress runs linearly from 0.0-1.0 over the specified transition duration.
2. *Inline transitions.* In this case, the author has full control over the progress of a transition. The progress can be accelerated, decelerated, animated forward, animated backwards, etc. These transitions are applied with a **transitionFilter** element as described in the InlineTransitions module. Inline transitions are based on the animation framework in the SMIL 2.0 BasicAnimation module and allow the progress of a media element's transition behavior to be explicitly controlled, much like the **animateMotion** element allows the spatial location of a media object to be directly manipulated.

## 12.3 Transition Taxonomy

We will classify transitions according to a two-level taxonomy of types and subtypes. Each of the transition types describe a group of transitions which are closely related. Within that type, each of the individual transitions are assigned a subtype which emphasizes the distinguishing characteristic of that transition. Usually, that distinguishing characteristic has something to do with the origin or direction of the geometric pattern of that transition. For

transition. Usually, that distinguishing characteristic has something to do with the origin or direction of the geometric pattern of that transition. For instance, one of the transition types is called "barWipe" and represents SMPTE Wipe Codes 1 and 2. SMPTE Wipe Code 1 is a wipe consisting of a vertical bar moving left to right. SMPTE Wipe Code 2 is a horizontal bar moving top to bottom. Therefore, the subtype for SMPTE Wipe Code 1 is called "leftToRight" and the subtype for SMPTE Wipe Code 2 is called "topToBottom".

Since the table of transition types and subtypes is quite extensive, we will not present the exhaustive list here. For the complete list of the predefined transition types and subtypes, as well as their mapping to SMPTE Wipe Codes, see the Appendix. Note that the mapping to SMPTE Wipe Codes are provided for reference only.

### 12.3.1 Default Transition Subtypes

For each of the types, one of the subtypes is labeled as the "default" subtype in the Appendix. If this transition class is not available or not implemented by the user agent, then the user agent should fall back on the default subtype for that transition family. This allows authors to specify a type for a transition class without requiring that they specify a subtype for the transition class. For more detail on parsing rules and fallback semantics, see the Transition Parsing Rules section.

### 12.3.2 Required Transitions

Implementations are required to implement the default subtype for each of the following transition types.

| Transition type | Default Transition subtype | SMPTE Wipe Code |
|---|---|---|
| barWipe | leftToRight | 1 |
| irisWipe | rectangle | 101 |
| clockWipe | clockwiseTwelve | 201 |
| snakeWipe | topLeftHorizontal | 301 |

Implementation of the rest of the transition types and subtypes listed in the Appendix is encouraged, but not required due to the large number of transitions.

## 12.4 BasicTransitions Module

Now that a taxonomy of transition types and subtypes is defined, we now discuss a "style-like" shorthand syntax for transitions. This shorthand syntax requires specification of the following:

1. The *class* of transition to be applied. For instance, to use a 1-second left-to-right wipe in a presentation, the wipe is defined as a transition class defined with the transition element.
2. The *media elements* to which this transition class is applied. In this shorthand syntax, the transition class is applied to the media element with the transIn and transOut attributes.

### 12.4.1 The transition element

The transition element defines a single transition class. This element may appear in different places in the document, depending upon the language profile. However in most cases, the transition element will be allowed only in the head of the document. For clarity, a grouping "container" element (such as the layout element in SMIL) may be desired in order to group all of the transition elements together. Since there may be multiple transition classes used in a document, then there may be multiple transition elements in the head of the document.

**Element attributes**

type
: This is the type or family of transition. This attribute is required and must be one of the transition families listed in the Taxonomy section (or it must be an extended transition type provided by the user agent). See the Transition Parsing Rules for an algorithm to determine which transition to use.

subtype
: This is the subtype of the transition. This parameter is optional and if specified, must be one of the transition subtypes appropriate for the specified type as listed in the Appendix (or it must be an extended transition subtype provided by the user agent). If this parameter is not specified then the transition reverts to the default subtype for the specified transition type. See the Transition Parsing Rules for an algorithm to determine which transition to use.

dur
: This is the duration of the transition. The value of this attribute must be a clock-value as defined by the SMIL Timing and Synchronization Module. The default duration is the intrinsic duration built into the transition. All of the transitions defined in the Appendix have a default duration of 1 second.

startProgress
: This is the amount of progress through the transition at which to begin execution. Legal values are real numbers in the range 0.0-1.0. For instance, we may want to begin a crossfade with the destination image being already 30% faded in. For this case, startProgress would be 0.3.
The default value is 0.0.

endProgress
: This is the amount of progress through the transition at which to end execution. Legal values are real numbers in the range 0.0-1.0 and the value of this attribute must be greater than or equal to the value of the startProgress attribute. If endProgress is equal to startProgress, then the transition remains at a fixed progress for the duration of the transition.
The default value is 1.0.

direction
: This specifies the direction the transition will run. The legal values are "forward" and "reverse". The default value is "forward". Note that this does not impact the media being transitioned to, but only affects the geometry of the transition. For instance, if you specified a type of "barWipe" and a subtype of "leftToRight", then the media would be wiped in by a vertical bar moving left to right. However, if you specified direction="reverse", then it would be wiped in by the same vertical bar moving right to left. Another example is the type of "starWipe" and subtype of "fourPoint". For this transition, running the transition forward reveals the destination media on the inside of a four-point star which starts small and gets larger as the transition progresses. Running this transition in reverse would reveal the destination media in the area outside of a large four-point star. The star begins large and gets smaller as the transition progresses. Note that not all transitions will have meaningful reverse interpretations. For instance, a crossfade is not a geometric transition, and therefore has no interpretation of reverse direction. Transitions which do not have a reverse interpretation should ignore the direction attribute and assume the default value of "forward".

fadeColor
If the value of the "type" attribute is "fade" and the value of the "subtype" attribute is "fadeToColor" or "fadeFromColor", then this attribute specifies the starting or ending color of the fade. If the value of the "type" attribute is not "fade", or the value of the "subtype" attribute is not "fadeToColor" or "fadeFromColor", then this attribute is ignored. Legal color values are CSS2 system color values.
The default value is "black".

**Element content**

The transition element can have the param element as a child.

**Examples of the transition element**

For example, suppose we wanted to define two transition classes: a simple 2-second fade-to-black and a 5-second keyhole-shaped iris wipe. These transition classes can be expressed as:

```
...
<transition id="ftb2" type="fade" subtype="fadeToColor"
        dur="2s" color="#000000" />
<transition id="star5" type="starWipe" subtype="fivePoint"
        dur="5s" />
...
```

## 12.4.2 The param element

The set of parameters discussed above are adequate for expressing all the transitions defined in this document. However, an implementation may choose to extend the set of transitions and define their own transition types and subtypes. Some of these new transition classes may need parameters which are not covered by the current set of attributes listed above. The purpose of the param element is to provide a generic means of supplying parameters to these extended transition types and subtypes.

The transition element can take the param element, defined in the SMIL MediaParam Module, as a child element. This element can be included from HTML or from some other module, depending upon the profile of the host language.

[redacted] needs a parameter called "flameLength". The example below shows how this implementation could use the param element to provide a value for "flameLength".

```
<transition id="myfire" type="superCool" subtype="fire">
   <param name="flameLength" value="20" />
</transition>
```

Note that the meaning of the additional parameters provided to the transition element depends upon the implementation of the specific transition.

## 12.4.3 The transIn and transOut attributes

Once a transition class has been defined in the head of a document, then a transition instance can be created by applying the transition class to the active duration of a media object element or other element with "renderable content". We do this by specifying a transIn or transOut attribute on the media object element. Transitions specified with a transIn attribute will begin at the beginning of the media element's active duration. Transitions specified with a transOut attribute will end at the end of the media element's active duration or end at the end of the element's fill state if a non-default fill value is applied.

The transIn and transOut attributes are added to all media object elements listed in the SMIL Media Object Module. The default value of both attributes is an empty string, which indicates that no transition should be performed.

The value of these attributes is a semicolon-separated list of transition id's. Each of the id's should correspond to the value of the XML identifier of one of the transition elements previously defined in the document. The purpose of the semicolon-separated list is to allow authors to specify a set of fallback transitions if the preferred transition is not available. The first transition in the list should be performed if the user-agent has implemented this transition. If this transition is not available, then the second transition in the list should be performed, and so on. If the value of the transIn or transOut attribute does not correspond to the value of the XML identifier of any one of the transition elements previously defined, then this is an error. In the case of this error, the value of the attribute should be considered to be the empty string and therefore no transition should be performed. For more detailed parsing rules, see the Transition Parsing Rules section.

**Rules For Applying Transitions to Media Elements**

1. Since the purpose of transitions is to pass from one media object to another, then transitions must be applied so that they either begin or end (or both) with some media object.

   [redacted] However, the visual effect may appear to be applying this transition in the middle of an element's active duration. Consider the following example:

   ```
   ...
   <par>
       <img src="butterfly.jpg" dur="10s" />
       <img src="eagle.jpg" begin="3s" dur="4s" />
   </par>
   ...
   ```

   Assuming that eagle.jpg is z-ordered on top of butterfly.jpg, then transitions applied to both the beginning and end of eagle.jpg would have the visual appearance of being applied during the active duration of butterfly.jpg. However, from the authoring perspective, they are still applied at the beginning and end of eagle.jpg.

2. Transitions happen by default during the active duration plus any fill period of the element to which they are applied. See the next rule for the effect of the fill value on the begin time of an out transition (transOut). Applying a transition to an element does not affect duration of the element. For instance, in the example below, applying a 1-second transition at the beginning of eagle.jpg does not add or subtract from the active duration of eagle.jpg - it is still displayed from 5-10 seconds in the presentation. Applying a 1-second transition at the beginning of eagle.jpg makes the transition take place from 5-6 seconds and applying a 2-second transition at the end of eagle.jpg would make the transition happen from 8-10 seconds.

   ```
   ...
   <seq>
       <img src="butterfly.jpg" dur="5s" ... />
       <img src="eagle.jpg"     dur="5s" ... />
   </seq>
   ```

3. Out transitions (transOut) end at the end of the active duration of an element plus any fill period. For elements with the default fill behavior of remove, out transitions end at the end of the active duration of the element. For elements with other values of fill, the transition ends at the end of the frozen period of the element.

   For instance, in the following presentation the fill behavior of the image element is "freeze", which keeps the image frozen until its parent ends. The parent ends when all of its children end, which is the end of the video at 30 seconds. In order to end at the end of the frozen duration (30 seconds) the fade-to-black transition begins at 29 seconds. Therefore both elements fade to black together at 29 seconds.

```
...
<transition id="toblack1s" type="fade" subtype="fadeToColor"
            fadeColor="#000000" dur="1s"/>
...
<par>
    <img  ... dur="10s" transOut="toblack1s" fill="freeze"/>
    ...
</par>
```

   However, in the following example the fill behavior of the image element is "remove". Therefore, the transition ends at the end of the active duration of the element. The image element fades to black starting at 9 seconds and the video element fades to black starting at 29 seconds.

```
...
<transition id="toblack1s"   type="fade" subtype="fadeToColor"
            color="#000000" dur="1s"/>
...
<par>
    <img   ... dur="10s" transOut="toblack1s" fill="remove"/>
    <video ... dur="30s" transOut="toblack1s"/>
</par>
```

4. The active duration for the media element to be transitioned *to* (the destination media) must either overlap the active duration or the fill state for the media element to be transitioned *from* (the background). In the Introductory example, the active duration for each destination media object immediately follows the end of the active duration of each background media object. In these cases (where the active durations immediately follow but do not overlap), the fill="transition" value must be used to enable the transition between the frozen last frame of the previous (background) media and active frames of the current (destination) media. See fill="transition" for more information. In cases where the active durations overlap (and hence the media being played to have different z-orders), the transition is between active frames of both media.

   In the following example the active durations do not overlap but the fill="transition" freezes the last frame of the first video. The result is a crossfade between the last frame of foo1.mpg and active frames of foo2.mpg.

```
...
<seq>
    <video src="foo1.mpg" fill="transition"... />
    <video src="foo2.mpg" transIn="xfade1s" ... />
</seq>
...
```

   In the following presentation, however, the crossfades both at the beginning and end of foo2.mpg are between active frames of both foo1.mpg and foo2.mpg since their active durations overlap. The example assumes the videos are at different z-orders.

```
...
<transition id="xfade" type="fade" subtype="crossfade" dur="1s" />
...
<par>
    <video src="foo1.mpg" dur="30s" />
    <video src="foo2.mpg" begin="10s" dur="10s"
           transIn="xfade" transOut="xfade" />
</par>
...
```

5. If the active durations for the media objects involved in the transition do not overlap, then the background for the area behind the media should be used. For example, the active durations for img1.jpg and img2.jpg do not overlap in the following example. Therefore, img1.jpg will transition to whatever is behind it.

```
...
<transition id="awipe" type="barWipe" dur="1s" ... />
...
<par>
    <img src="img1.jpg" dur="2s" transOut="awipe" .../>
    <img src="img2.jpg" begin="5s" dur="2s"       .../>
</par>
...
```

6. If both an in and out transition are specified on a media element, and the times for those transitions overlap, then the in transition takes precedence, and the out transition should be ignored and no out transition should be performed.

   For instance, in the following example, the "barWipe" in transition will take precedence over the "fadeToColor" out transition. The in transition will fully take place for the first 2 seconds of img1.jpg, and the out transition is ignored and no out transition is performed.

```
...
<transition id="awipe"   type="barWipe"    dur="2s" ... />
<transition id="toblack" type="fadeToColor dur="2s" ... />
...
<img src="img1.jpg" dur="3s" transIn="awipe" transOut="toblack" .../>
...
```

7. Since transitions imply passing from the beginning or end of display of one media to another, transitions do not repeat.

   Consider the following example. The img2.jpg has a simple duration of 5 seconds, but an active duration of 15 seconds, since it plays a total of three times. However, the in transition only plays once at the beginning of the active duration of img2.jpg, which is at 5 seconds into the active duration of the sequence time container. The out transition also plays only once, starting at 19 seconds into the active duration of the sequence time container.

```
...
<transition id="awipe"   type="barWipe"     dur="1s" ... />
<transition id="toblack" type="fadeToColor" dur="1s" ... />
...
<seq>
    <img src="img1.jpg" dur="5s" fill="transition" .../>
    <img src="img2.jpg" dur="5s" repeatCount="3"
```

```
        transIn="awipe" transOut="toblack" ... />
    <img src="img3.jpg" dur="5s" .../>
</seq>
...
```

**Use of fill="transition"**

The fill attribute, defined in the SMIL Timing and Synchronization Modules, allows an author to specify that an element should be extended beyond its active duration by *freezing* the final state of the element. A new fill value, "transition", is required to enable transitions between elements that would not normally be displayed at the same time. This fill attribute value can be applied only to elements with renderable content and is not applicable to pure time container elements such as par, seq, and excl. If fill=`transition` is applied to a pure time container element, then the value is ignored and reverts to its default value.

The >="`transition` fill value indicates that after its active duration ends the element will be frozen and it will remain frozen until the end of the next transition on an element with which it overlaps in the layout. The element containing the fill="transition" will be removed when the transition ends. The timing rules defined in the SMIL Timing and Synchronization Modules still apply: the element is subject to the constraints of its parent time container and can be removed by its parent regardless of whether or not a transition is declared. Each profile must define the meaning of overlapping in the layout.

In the following example *not* using transitions, the default behavior is to remove the object representing img1.jpg after 10 seconds.

```
...
<seq>
    <img src="img1.jpg" dur="10s" ... />
    <img src="img2.jpg" dur="10s" ... />
</seq>
...
```

Adding a transition between img1.jpg and img2.jpg requires that img1.jpg remains displayed after its active duration ends so that it can be used by the transition to img2.jpg. The first image is removed as soon as the transition ends. The fill=>="`transition` enables this behavior as illustrated by the

```
...
<transition id="awipe" type="barWipe" dur="1s" ... />
...
<seq>
    <img src="img1.jpg" dur="10s" fill="transition" ... />
    <img src="img2.jpg" dur="10s" transIn="awipe"   ... />
</seq>
...
```

**Slideshow example with transitions**

After adding the fill and transIn attributes, our example slideshow from the Introduction section now looks like the following:

```
...
<transition id="wipe1" type="barWipe" subtype="leftToRight" dur="1s"/>
...
<seq>
    <img src="butterfly.jpg" dur="5s" fill="transition" ... />
    <img src="eagle.jpg"     dur="5s" fill="transition" transIn="wipe1" ... />
    <img src="wolf.jpg"      dur="5s" fill="transition" transIn="wipe1" ... />
    <img src="seal.jpg"      dur="5s"                   transIn="wipe1" ... />
</seq>
```

Now the presentation plays as follows, as illustrated by this animated image.

- At 0 seconds, cuts directly to butterfly.jpg.
- At 5 seconds begins a 1-second left-to-right wipe from butterfly.jpg to eagle.jpg.
- At 6 seconds, eagle.jpg is fully displayed and remains displayed for 4 more seconds until 10 seconds.
- At 10 seconds, begins a 1-second left-to-right wipe from eagle.jpg to wolf.jpg.
- At 11 seconds, wolf.jpg is fully displayed for 4 more seconds until 15 seconds.
- At 15 seconds, begins a 1-second left-to-right wipe to from wolf.jpg to seal.jpg.
- At 16 seconds, seal.jpg is fully displayed for 4 more seconds until 20 seconds.
- At 20 seconds the presentation ends.

Notice that these transitions occur *during* the active duration of each of the images which reference the transition and do not add or subtract from their host element's active duration. In this case, the transition occurs at the beginning of each media element's active duration.

Notice the importance of fill="transition". If we had not specified fill="transition" on butterfly.jpg, eagle.jpg, and wolf.jpg, then the transitions at 5, 10, and 15 seconds would have taken place between the background of the playback area (or the default background color, depending on how the layout language is specified) instead of the previous image in the sequence.

**Exclusive children and fill="transition"**

The fill="transition" also enables transitions from one excl child to another when the previously active child would normally be removed from the display. In the following example the first image transitions in from the background, displays for 5 seconds and then freezes because of the fill="transition". The next child activated by a button click will transition in from butterfly.jpg. When that child completes it will also freeze due to the fill="transition", remaining available for use in the next transition. It will transition in to the next image activated by a button click, and so on.

```
...
<transition id="wipe1" type="barWipe" subtype="leftToRight" dur="1s"/>
...
<excl>
    <img src="butterfly.jpg" begin="0" dur="5s" fill="transition" transIn="wipe1" ... />
    <img src="eagle.jpg"     begin="button1.click" dur="5s" fill="transition" transIn="wipe1" ... />
    <img src="wolf.jpg"      begin="button2.click" dur="5s" fill="transition" transIn="wipe1" ... />
    <img src="seal.jpg"      begin="button3.click" dur="5s" fill="transition" transIn="wipe1" ... />
</excl>
```

Note that fill takes effect after the active duration of an element ends. In the above example, if button2 is clicked at 3 seconds, then butterfly.jpg will end, and the fill="transition" value for butterfly.jpg will be in effect through the end of the next transition. Therefore the transition will occur from butterfly.jpg to wolf.jpg and the frozen butterfly.jpg will disappear when the transition completes.

The pauseDisplay attribute of the priorityClass element, defined in the SMIL Timing and Synchronization Modules can also be used to control the

display of children of an exclusive element. In the example above, pauseDisplay could be used to keep butterfly.jpg displayed when paused so the transition would occur between butterfly.jpg to the next media activated, and butterfly.jpg would continue to be displayed after the transition (assuming that it is not completely covered by the other media).

### 12.4.4 Handling Parameter Errors

Transitions parameters can be specified incorrectly in many different ways with varying levels of severity. Therefore, the following errors should be handled with the specified action:

1. *Transition type is not valid.* If the implementation does not recognize the value of the type attribute, or if that transition type is not implemented, then this transition class is invalid. However, this does not necessarily mean that no transition will be performed, as specified in the Transition Parsing Rules section.
2. *Transitions subtype is not valid for specified transition type.* The specified transition subtype should be ignored and the default subtype for the specified transition type should be performed.
3. *Transition duration is not specified.* The default duration of 1 second should be assumed.
4. *Transition parameter besides type or subtype is outside the legal range.* If a transition parameter is specified outside of the legal range, then the default value of the parameter should be assumed.
5. *Transition parameter does not apply to this transition type.* Since not all transition parameters apply to all transition types, then a common error could be to specify a transition parameter which does not apply to the specified transition type. These irrelevant parameters should be ignored. For instance, the "borderWidth" parameter does not apply to the "fade" transition type. If "borderWidth" were to be specified for the "fade" transition type, then it should be ignored.
6. *Transition duration is longer than the duration of the media object itself.* In this case, the entire transition should be ignored and not performed.

### 12.4.5 Transition Parsing Rules

As stated earlier, each **transition** can have a default transition subtype. Also, the transIn or transOut attributes on media elements take a semicolon-separated list of transition id's to indicate a list of fallback transitions. To eliminate ambiguity between the default subtype and the fallback list, this section defines an algorithm that must be followed to determine the transition to perform. The general procedure is that the first resolved transition from the list of fallback transitions is the one that should be performed.

Given one or more previously declared **transition** elements and a list of fallback transition id's (specified on the transIn or transOut attributes), an implementation must use the following algorithm to determine the transition to perform.

1. Set `current-id` to the first id in the list.
2. If `current-id` is empty (we have no more id's in the list), then exit this algorithm. The implementation must not consider this an error and must not perform any transition.
3. If `current-id` is the id of some previously defined **transition** element then go to Step 4. If not, then set `current-id` to the next id in the list and go to Step 2.
4. If the value of the "type" attribute on the **transition** element identified by `current-id` is known to the implementation then go to Step 5. If not, then set `current-id` to the next id in the list and go to Step 2.
5. If the "subtype" attribute is specified on the **transition** element identified by `current-id` then go to Step 6. If it is not specified, then the implementation must exit this algorithm and perform the **default** transition subtype for the specified transition type.
6. If the value of the "subtype" attribute on the **transition** element identified by `current-id` is known to the implementation then the implementation [illegible] to Step 2.

### 12.4.6 Extending The Set Of Transitions

In the algorithm specified earlier for determining which transition to perform, there is an implicit method for extending the set of transitions. If the new transition does not fall into any of the general descriptions of transition families in the Transition Taxonomy section, implementations may create a new transition type (a new family of transitions) and then create new transition subtypes under that newly-defined type. However, it is recommended that if the new transition falls into one of the existing families of transitions, implementations should simply extend the set of subtypes for that existing type. Implementations may use whatever type and subtype names they choose for these extended transitions. However, when these new transitions are used within a document, they must be namespace-qualified.

## 12.5 InlineTransitions Module

As mentioned in the Transition Model section, SMIL 2.0 Transitions allow two methods of specifying transitions: a shorthand method and an inline method. The BasicTransitions module specifies the shorthand method while this module specifies the inline method. Inline transitions provide additional timing and progress control compared to the shorthand transitions. The transitionFilter element provides the inline transition support.

### 12.5.1 The **transitionFilter** element

The **transitionFilter** element is an animation element, similar to the animateMotion element defined in the SMIL 2.0 BasicAnimation Module. The animateMotion element animates the position of an element. In contrast, the transitionFilter element animates the progress of a filter behavior (transition) on a media element or elements with renderable content. The filter behavior temporarily alters the visual or aural rendering of the media. The transitionFilter element can target any element with "renderable content", not necessarily a media element. The host language determines which elements to which transitionFilter can be applied. For instance, in HTML, a span or a div may represent "renderable content". The transitionFilter element may target a renderable content element in two ways: it may be the child of that element, or with the targetElement attribute.

The transitionFilter element shares many of the attributes from the **transition** element. It integrates timing support from the SMIL 2.0 BasicInlineTiming Module, and animation support from the SMIL 2.0 BasicAnimation module. This module can also be combined with other SMIL 2.0 Modules such as TimeManipulations, depending on the modules implemented by the host language.

This element must be the child of a media element or other element with renderable content, as defined by the host language. This is in contrast to BasicTransitions that are declared in the "transition" element and then specified in the transIn or transOut attributes that are applied to media elements.

Similar to how transIn and transOut are *attributes* of the media object to which the transition is applied, the transitionFilter element is a *child* of the media object to which the transition is applied. However, even though the transitionFilter element is a child of a media object, it is not a time container, and cannot extend the active duration of the media object. Therefore, if transitionFilter is a child of a media element, it can only apply a transition to that media element during that media element's active duration. If it is desired to apply a transition during an element's frozen period, then transitionFilter should not be a child of the media element. Rather, the targetElement attribute should be used to target that media element.

Note that the transitionFilter element represents an "in" transition in the sense that the destination media (the media that is fully visible when progress is 1.0) is the media to which the transition is applied (the parent media, in this case). However, since transitionFilter gives full control over the timing of the progress, an "in" transition may be made to look like an "out" transition by simply running the transition from a progress of 1.0 and ending the transition at a progress of 0.0.

**transitionFilter Element attributes**

type
: This is the same attribute as in the transition element.

subtype
: This is the same attribute as in the transition element.

mode
: Indicates whether the transitionFilter's parent element will transition in or out. Legal values are "in" indicating that the parent media will become more visible as the transition progress increases and "out" indicating that the parent media will become less visible as the transition progress increases. The default value is "in". Unlike the transIn and transOut attributes on media elements, the mode attribute does not automatically tie the transitionFilter to the begin or end of the media. Authors can use the begin attribute on the transitionFilter to indicate the begin time for the transitionFilter.

fadeColor
: This is the same attribute as in the transition element.

begin
: Defined in the SMIL Timing and Synchronization Module. This attribute is optional and the default is 0 seconds.

dur
: Defined in the SMIL Timing and Synchronization Module. The default duration is the intrinsic duration built into the transition. All of the transitions defined in the Appendix have a default duration of 1 second.

end
: Defined in the SMIL Timing and Synchronization Module.

: Defined in the SMIL Timing and Synchronization Module.

repeatDur
: Defined in the SMIL Timing and Synchronization Module.

from
: Amount of progress through the transitionFilter from which to begin execution of the transitionFilter. Legal values are real numbers in the range 0.0-1.0. For instance, this attribute would equal "0.3" to begin a cross-fade with the destination image faded in by 30%. This attribute is defined in the SMIL 2.0 BasicAnimation Module and is similar to the startProgress attribute on the transition element. The default value is 0.0. Ignored if the values attribute is specified.

to
: Amount of progress through the transitionFilter at which to end execution of the transitionFilter. Legal values are real numbers in the range 0.0-1.0. This attribute is defined in the SMIL 2.0 BasicAnimation Module and is similar to the endProgress attribute on the transition element. The default value is 1.0. Ignored if the values attribute is specified.

by
: Specifies a relative offset value for the progress of the transitionFilter. Legal values are real numbers in the range 0.0-1.0. Defined in the SMIL 2.0 BasicAnimation Module. Ignored if the values attribute is specified.

values
: A semicolon-separated list of one or more values specifying the progress of the transitionFilter. This attribute can provide more precise control over the progress than a combination of the from, to, and by attributes and overrides those attributes if specified. Legal values are real numbers in the range 0.0-1.0. Defined in the SMIL 2.0 BasicAnimation Module. Use the calcMode attribute to determine how the values are interpreted.

calcMode
: Specifies the interpolation mode of the progress of the transitionFilter. Defined in SMIL 2.0 BasicAnimation Module. The calcMode attribute can take any of the following values:

  `discrete`
  : This specifies that the transitionFilter progress will jump from one value to the next without any interpolation.

  `linear`
  : Simple linear interpolation between values is used to calculate the progress of the transitionFilter. This is the default.

  `paced`
  : This value will be ignored if specified for a transitionFilter element. The default value ("linear") will be used instead.

targetElement
: This attribute specifies the target element to be animated. The attribute value must be the value of an XML identifier attribute of an element (i.e. an "IDREF") within the host document. For a formal definition of IDREF, refer to XML 1.0 [XML10].

: This attribute specifies the target element to be animated. The attribute value must be an XLink locator, referring to the target element to be animated.

When integrating transitionFilter elements into the host language, the language designer should avoid including both of these attributes. If however, the host language designer chooses to include both attributes in the host language, then when both are specified for a given animation element the XLink href attribute takes precedence over the targetElement attribute.

The advantage of using the targetElement attribute is the simpler syntax of the attribute value compared to the href attribute. The advantage of using the XLink href attribute is that it is extensible to a full linking mechanism in future versions of SMIL Transitions, and the animation element can be processed by generic XLink processors. The XLink form is also provided for host languages that are designed to use XLink for all such references. The following two examples illustrate the two approaches.

This example uses the simpler targetElement syntax:

```
<transitionFilter targetElement="foo" .../>
```

This example uses the more flexible XLink locater syntax, with the equivalent target:

```
<foo xmlns:xlink="http://www.w3.org/1999/xlink">
  ...
  <transitionFilter xlink:href="#foo" .../>
  ...
</foo>
```

When using an XLink href attribute on a transitionFilter element, the following additional XLink attributes need to be defined in the host language. These may be defined in a DTD, or the host language may require these in the document syntax to support generic XLink processors. For more information, refer to [XLINK].

The following XLink attributes are required by the XLink specification. The values are fixed, and so may be specified as such in a DTD. All other XLink attributes are optional, and do not affect SMIL Transitions semantics.

**XLink attributes for href**

type
: Must be `simple`. Identifies the type of XLink being used.

actuate
: Must be `onLoad`. Indicates that the link to the target element is followed automatically (i.e., without user action).

show
: Must be `embed`. Indicates that the reference does not include additional content in the file.

Additional details on the target element specification as relates to the host document and language are described in the Integration section.

**Media Element fill="transition"**

fill
: This module adds the "transition" value to the possible values of the fill attribute defined on media in the SMIL Timing and Synchronization Module. This is the same attribute as specified in the BasicTransitions module.

**Element content**

The transitionFilter element can have the param element as a child.

**Examples of the transitionFilter element**

Example 1: transitionFilter slide show

The following example uses inline transitions to provide a slideshow that includes transitions between the images, similar to the example discussed in the introduction. The presentation plays as follows.

- Beginning at 0 seconds, butterfly displays for 5 seconds.
- At 5 seconds butterfly freezes due to the fill="transition" specified on it and the 1-second left-to-right wipe from butterfly into eagle occurs.
- At 6 seconds, eagle is fully displayed and remains displayed for 4 more seconds.
- At 10 seconds eagle freezes due to the fill="transition" and the 1-second left-to-right wipe from eagle to wolf occurs.
- At 11 seconds, wolf is fully displayed and remains displayed for 4 more seconds.
- At 15 seconds wolf freezes due to the fill="transition" and the 1-second left-to-right wipe from wolf to seal occurs.
- At 16 seconds, seal is fully displayed and remains displayed for 4 more seconds.
- At 20 seconds the presentation ends.

```
...
<seq>
    <img id="butterfly" src="butterfly.jpg" dur="5s" fill="transition" />
    <img id="eagle" src="eagle.jpg" dur="5s" fill="transition" >
        <transitionFilter type="barWipe" subtype="leftToRight" dur="1s" />
    <img id="wolf" src="wolf.jpg" dur="5s" fill="transition" >
        <transitionFilter type="barWipe" subtype="leftToRight" dur="1s" />
    </img>
    <img id="seal" src="seal.jpg" dur="5s" >
        <transitionFilter type="barWipe" subtype="leftToRight" dur="1s" />
    </img>
</seq>
...
```

Example 2: transitionFilter discrete clock transition

The following example uses a values list and discrete calcMode to specify the progress of the transition in 12 steps. The transition begins 2 seconds after the video begins and continues for 12 seconds. Since the transition is circular, the effect is that of a clock-wipe that reveals one hour on the clock face at a time.

```
<video id="video1" src="car.avi">
    <transitionfilter id="trans1"
        type="ellipseWipe" subtype="circle"
        begin="2" dur="12" calcMode="discrete"
        values="0.083; 0.166; 0.250; 0.333; 0.416; 0.500;
            0.583; 0.666; 0.750; 0.833; 0.916; 1.000;" />
</video>
```

Example 3: transitionFilter from and to

The following example uses a partial transition that progresses from 0 to 50% (0.5) complete. It assumes that the video is positioned directly on top of the image in the layout. The presentation plays as follows.

- Beginning at 0 seconds the car displays. The image begins as well, but is not visible because it is behind the video.
- At 1 second the transition begins to wipe away the car and reveal the background image. Over the duration of the transition (2 seconds) the wipe proceeds clockwise from the 12 o'clock position revealing 50% of the image behind the car, up through the 6 o'clock position.
- At 3 seconds the car and transition both end, revealing all of racing.jpg.
- At 5 seconds the image ends.

```
<par>
    <img src="racing.jpg" begin="0s" dur="5s" />
    <video id="car" src="car.avi" begin="0s" dur="3s"
        <transitionfilter type="clockWipe" subtype="clockwiseTwelve"
            begin="1s" dur="2s" from="0.0" to="0.5" />
    </video>
</par>
```

## 12.5.2 The param element

The transitionFilter element can take the param element, defined in the SMIL MediaParam Module, as a child element. This element can be included from HTML or from some other module, depending upon the profile of the host language. The param element defines parameter information specific to the individual transitionFilter. For example, the implementation of a windshieldWipe could take a parameter that defines the length of the radius for the wipe as follows:

```
<transitionfilter type="windshieldWipe"
    begin="4" dur="3" from="0.5" to="1.0" >
    <param name="radius" value="3in" >
```

```
</transitionFilter>
```

Support of the param element is implementation-dependant. The meaning of the parameters depends upon the implementation of the specific transition.

## 12.6 TransitionModifiers Module

The TransitionModifiers module gives additional control over the visual effect of the transition: controlling the horizontal and vertical repeat pattern, and controlling the visual effect along the pattern border. The SMPTE standard also allows for this type of geometric control.

This module requires either the BasicTransitions Module or the InlineTransitions Module.

horzRepeat
: This attribute specifies how many times to perform the transition pattern along the horizontal axis.
The default value is 1 (the pattern occurs once horizontally).

vertRepeat
: This attribute specifies how many times to perform the transition pattern along the vertical axis.
The default value is 1 (the pattern occurs once vertically).

borderWidth
: This attribute specifies the width of a generated border along a wipe edge. Legal values are integers greater than or equal to 0. If borderWidth is equal to 0, then no border should be generated along the wipe edge.
The default value is 0.

borderColor
: values are CSS2 system color values *or* the string "blend". If the value of this attribute is a color, then the generated border along the wipe or warp edge is filled with this color. If the value of this attribute is "blend", then the generated border along the wipe blend is an additive blend (or blur) of the media sources. The default value is "black".

### 12.6.1 Horizontal and Vertical Pattern Repeat

Using the horzRepeat and vertRepeat attributes, the geometric pattern which makes up the transition can be repeated in both the horizontal and vertical directions over the area occupied by the media. To achieve the repeat, the area occupied by the destination media is divided into equal sections horizontally and/or vertically according to the values of horzRepeat and vertRepeat. Identical transitions are then performed, one in each of the resulting sections, at the same time.

The following diagrams illustrate the difference between the behavior provided by the default horzRepeat and vertRepeat attributes and each attribute with two copies of the transition applied to an image.

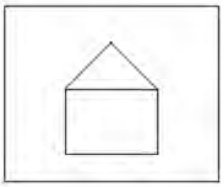


Figure 1. An image that does not have any transitions applied to it.

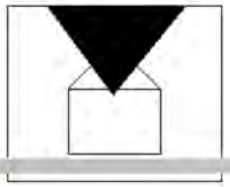


Figure 2. The image from Figure 1 with a fan transition in progress. The current area of the transition is illustrated by the black triangle. This example uses the default value of one for both horzRepeat and vertRepeat, which yields one occurrence of the transition. Therefore, the fan pattern is not repeated in either direction.

This animated image illustrates the single fan transition from Figure 2. The fan transition could be declared as follows:

```
<transition ... type="fanWipe" subtype="centerTop" dur="1s"/>
```

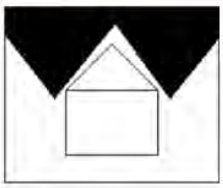


Figure 3. The same fan transition from Figure 2 in progress, but with two horizontal repetitions (horzRepeat="2"). The repeat yields two smaller, but identical copies of the transition, one in the left half of the image and one in the right half of the image. The number of patterns in the horizontal direction equals horzRepeat.





Figure 4: The same fan transition from Figure 3 in progress, but with two vertical repetitions (vertRepeat="2"). The repeat yields two smaller, but identical copies of the transition, one in the top half of the image and one in the bottom half of the image. The number of patterns in the vertical direction equals vertRepeat.

The following example shows the declaration of the transition from Figure 4. It specifies a vertRepeat value of 2 which indicates that the fan transition will occur in two locations on the media at once.

```
<transition ... type="fanWipe" subtype="centerTop" dur="1s"
        horzRepeat="1" vertRepeat="2"/>
```

Note that we didn't have set to horzRepeat to one, since one is the default value, but we explicitly set it here for clarity. This transition is illustrated by this animated image.

In a more complex example, if horzRepeat was set to 3 and vertRepeat was set to 2 then 6 small fan transitions would occur at once over the media, in a pattern of 3 wide (horzRepeat) and 2 high (vertRepeat).

Note that the horzRepeat and vertRepeat attributes might not produce a visual change depending on the type of transition. Specifically, these attributes have no visual affect if replicating the transition pattern produces identical results. For example, a vertRepeat attribute set equal to two would have no visual impact on a left-to-right push- or slideWipe because the transition would still occur from the left edge all the way to the right edge of the media. In contrast, the same vertRepeat attribute would affect a top-to-bottom push- or slideWipe because one transition would occur from the top to the middle of the media and the other transition would occur from the middle to the bottom of the media at the same time. Neither horzRepeat nor vertRepeat affect a fade transition because the fade applies uniformly regardless of how many times it is replicated.

Implementations can choose to optimize by ignoring the horzRepeat and vertRepeat attributes in cases where they would have no effect.

## 12.7 Integration

The purpose of this section is to specify requirements and recommendations on the host language or profile in order to integrate SMIL Transitions.

1. Profiles that include the TransitionModifiers Module must also include either the BasicTransitions Module or the InlineTransitions Module.
2. Since transitions are applied to media and are not specific to the layout of that media, then SMIL Transitions are layout agnostic. Any type of layout language may be used.
3. Profiles that include the BasicTransitions Module must have some method of specifying the **transition** element. It is recommended that all **transition** elements be specified in the `<head>` of the document (if one exists) and also that there be some sort of container element which groups all the **transition** elements together (similar to the `<layout>` element in the `<head>` of SMIL 1.0 documents).
4. A profile integrating the BasicTransitions Module must provide a means of declaring an XML identifier on **transition** elements.
5. A profile integrating the InlineTransitions Module module must provide a means of declaring an XML identifier on **transitionFilter** elements.
6. The profile must define what overlapping in the layout means for fill="transition" (required in the BasicTransitions Module and the InlineTransitions Module).
7. If the profile includes the InlineTranstions Module, then the host language designer must choose whether to support the targetElement attribute or the XLink attributes for specifying the target element. Note that if the XLink syntax is used, the host language designer must decide how to denote the XLink namespace for the associated attributes. The namespace can be fixed in a DTD, or the language designer can require colonized attribute names (*qnames*) to denote the XLink namespace for the attributes. The required XLink attributes have fixed values, and so may also be specified in a DTD, or can be required on the animation elements. Host language designers may require that the optional XLink attributes be specified. These decisions are left to the host language designer - the syntax details for XLink attributes do not affect the semantics of SMIL Transitions.

## 12.8 Appendix: Taxonomy Tables

Table 1: The Taxonomy Table contains a detailed list of transition type and subtype names. The names of the types and subtypes have been chosen so that the name provides some hint of the visual effect of the transition. However, in some cases, the name alone is not enough to visually describe these transitions. For a better understanding of these transitions, please see pages 11-16 of SMPTE 258M-1993 [SMPTE-EDL].

As an assistance to the reader in identifying the patterns of the SMPTE transitions this Appendix also provides illustrations of the corresponding SMPTE wipes in the following tables.

Table 2: SMPTE Edge Wipes

Table 3: SMPTE Iris Wipes

Table 4: SMPTE Clock Wipes

Table 5: SMPTE Matrix Wipes

In the case of any discrepancies between type and subtype names in the Taxonomy Table and in the illustrated tables, the Taxonomy Table takes precedence. The SMPTE specification [SMPTE-EDL] takes precedence over the illustrated tables in this appendix. The illustrations are provided for convenience only.

### 12.8.1 Table 1: The Taxonomy Table

The SMPTE Wipe Codes (where appropriate) are provided in parentheses after the subtype name and are for reference only. The Wipe Codes are not part of the transition subtype name. The default transition subtype for each type is indicated by the word [default].

| **Transition type** | **Transition subtypes** (SMPTE Wipe Codes in parentheses) |
|---|---|
| Edge Wipes - wipes occur along an edge | |
| "barWipe" | "leftToRight" (1) [default], "topToBottom" (2) |

| | |
|---|---|
| "boxWipe" | "topLeft" (3) [default], "topRight" (4), "bottomRight" (5), "bottomLeft" (6), "topCenter" (23), "rightCenter" (24), "bottomCenter" (25), "leftCenter" (26) |
| "fourBoxWipe" | "cornersIn" (7) [default], "cornersOut" (8) |
| "barnDoorWipe" | "vertical" (21) [default], "horizontal" (22), "diagonalBottomLeft" (45), "diagonalTopLeft" (46) |
| "diagonalWipe" | "topLeft" (41) [default], "topRight" (42) |
| "bowTieWipe" | "vertical" (43) [default], "horizontal" (44) |
| "miscDiagonalWipe" | "doubleBarnDoor" (47) [default], "doubleDiamond" (48) |
| "veeWipe" | "down" (61) [default], "left" (62), "up" (63), "right" (64) |
| "barnVeeWipe" | "down" (65) [default], "left" (66), "up" (67), "right" (68) |
| "zigZagWipe" | "leftToRight" (71) [default], "topToBottom" (72) |
| "barnZigZagWipe" | "vertical" (73) [default], "horizontal" (74) |
| Iris Wipes - shapes expand from the center of the media | |
| "irisWipe" | "rectangle" (101) [default], "diamond" (102) |
| "triangleWipe" | "up" (103) [default], "right" (104), "down" (105), "left" (106) |
| "arrowHeadWipe" | "up" (107) [default], "right" (108), "down" (109), "left" (110) |
| "pentagonWipe" | "up" (111) [default], "down" (112) |
| "hexagonWipe" | "horizontal" (113) [default], "vertical" (114) |
| "ellipseWipe" | "circle" (119) [default], "horizontal" (120), "vertical" (121) |
| "eyeWipe" | "horizontal" (122) [default], "vertical" (123) |
| "roundRectWipe" | "horizontal" (124) [default], "vertical" (125) |
| "starWipe" | "fourPoint" (127) [default], "fivePoint" (128), "sixPoint" (129) |
| "miscShapeWipe" | "heart" (130) [default], "keyhole" (131) |
| Clock Wipes - rotate around a center point | |
| "clockWipe" | "clockwiseTwelve" (201) [default], "clockwiseThree" (202), "clockwiseSix" (203), "clockwiseNine" (204) |
| "pinWheelWipe" | "twoBladeVertical" (205) [default], "twoBladeHorizontal" (206), "fourBlade" (207) |
| "singleSweepWipe" | "clockwiseTop" (221) [default], "clockwiseRight" (222), "clockwiseBottom" (223), "clockwiseLeft" (224), "clockwiseTopLeft" (241), "counterClockwiseBottomLeft" (242), "clockwiseBottomRight" (243), "counterClockwiseTopRight" (244) |
| "fanWipe" | "centerTop" (211) [default], "centerRight" (212), "top" (231), "right" (232), "bottom" (233), "left" (234) |
| "doubleFanWipe" | "fanOutVertical" (213) [default], "fanOutHorizontal" (214), "fanInVertical" (235), "fanInHorizontal" (236) |
| "doubleSweepWipe" | "parallelVertical" (225) [default], "parallelDiagonal" (226), "oppositeVertical" (227), "oppositeHorizontal" (228), "parallelDiagonalTopLeft" (245), "parallelDiagonalBottomLeft" (246) |
| "saloonDoorWipe" | "top" (251) [default], "left" (252), "bottom" (253), "right" (254) |
| "windshieldWipe" | "right" (261) [default], "up" (262), "vertical" (263), "horizontal" (264) |
| Matrix Wipes - media is revealed in squares following a pattern | |
| "snakeWipe" | "topLeftHorizontal" (301) [default], "topLeftVertical" (302), "topLeftDiagonal" (303), "topRightDiagonal" (304), "bottomRightDiagonal" (305), "bottomLeftDiagonal" (306) |
| "spiralWipe" | "topLeftClockwise" (310) [default], "topRightClockwise" (311), "bottomRightClockwise" (312), "bottomLeftClockwise" (313), "topLeftCounterClockwise" (314), "topRightCounterClockwise" (315), "bottomRightCounterClockwise" (316), "bottomLeftCounterClockwise" (317) |
| "parallelSnakesWipe" | "verticalTopSame" (320), [default] "verticalBottomSame" (321), "verticalTopLeftOpposite" (322), "verticalBottomLeftOpposite" (323), "horizontalLeftSame" (324), "horizontalRightSame" (325), "horizontalTopLeftOpposite" (326), "horizontalTopRightOpposite" (327), "diagonalBottomLeftOpposite" (328), "diagonalTopLeftOpposite" (329) |
| "boxSnakesWipe" | "twoBoxTop" (340) [default], "twoBoxBottom" (341), "twoBoxLeft" (342), "twoBoxRight" (343), "fourBoxVertical (344), "fourBoxHorizontal" (345) |
| "waterfallWipe" | "verticalLeft" (350) [default], "verticalRight" (351), "horizontalLeft" (352), "horizontalRight" (353) |
| Non-SMPTE Wipes | |
| "pushWipe" | "fromLeft" [default], "fromTop", "fromRight", "fromBottom" |
| "slideWipe" | "fromLeft" [default], "fromTop", "fromRight", "fromBottom" |
| "fade" | "crossfade" [default], "fadeToColor", "fadeFromColor" |

**Descriptions of non-SMPTE Transitions**

The "pushWipe" transitions looks as if the destination media "pushes" the background media away. In other words, both the background media and the destination media are moving.

In the "slideWipe" transitions, the destination media moves, but the background media does not. The visual effect of "slideWipe" transitions is that the destination media is "sliding" in across the background media.

The "fade" transitions are pixel-by-pixel blends between the destination media and either the background media or a specified color.

### 12.8.2 Table 2: SMPTE Edge Wipes

Edge wipes start from a horizontal, vertical, or diagonal edge and expand in a given shape. The direction of change is to increase the white area.

**"barWipe"**


"leftToRight" (1) [default]


"topToBottom" (2)

**"boxWipe"**


"topLeft" (3) [default]


"topRight" (4)


"bottomRight" (5)


"bottomLeft" (6)


"topCenter" (23)

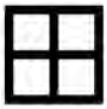

"rightCenter" (24)


"bottomCenter" (25)


"leftCenter" (26)

**"fourBoxWipe"**


"cornersIn" (7) [default]


"cornersOut" (8)

**"barnDoorWipe"**


"vertical" (21) [default]


"horizontal" (22)


"diagonalBottomLeft" (45)


"diagonalTopLeft" (46)

**"diagonalWipe"**



"topLeft" (41) [default]



"topRight" (42)

**"bowTieWipe"**



"vertical" (43) [default]



"horizontal" (44)

**"miscDiagonalWipe"**



"doubleBarnDoor" (47) [default]



"doubleDiamond" (48)

**"veeWipe"**



"down" (61) [default]



"left" (62)





**"barnVeeWipe"**



"down" (65) [default]



"left" (66)



"up" (67)



"right" (68)

**"zigZagWipe"**



"leftToRight" (71) [default]



"topToBottom" (72)

**"barnZigZagWipe"**



"vertical" (73) [default]



"horizontal" (74)

### 12.8.3 Table 3: SMPTE Iris Wipes

Iris wipes expand in a given shape from the center of the media. The direction of change is to increase the white area.

**"irisWipe"**



"rectangle" (101) [default]



"diamond" (102)

**"triangleWipe"**



"up" (103) [default]



"right" (104)



"down" (105)



"left" (106)

**"arrowHeadWipe"**



"up" (107) [default]



"right" (108)



"down" (109)



"left" (110)

**"pentagonWipe"**



"up" (111) [default]



"down" (112)

**"hexagonalWipe"**

"hexagonWipe"


"horizontal" (113) [default]


"vertical" (114)

"ellipseWipe"


"circle" (119) [default]


"horizontal" (120)


"vertical" (121)

"eyeWipe"


"horizontal" (122) [default]


"vertical" (123)

"roundRectWipe"


"horizontal" (124) [default]


"vertical" (125)

"starWipe"


"fourPoint" (127) [default]


"fivePoint" (128)


"sixPoint" (129)

"miscShapeWipe"


"heart" (130) [default]


"keyhole" (131)

### 12.8.4 Table 4: SMPTE Clock Wipes

Clock wipes rotate around a center point. The center of rotation is indicated in the following illustrations by the ⊙ symbol. The arrow ↷ shows the direction of rotation. The direction of change is to increase the white area.

**"clockWipe"**



"clockwiseTwelve" (201) [default]



"clockwiseThree" (202)



"clockwiseSix" (203)



"clockwiseNine" (204)

**"pinWheelWipe"**



"twoBladeVertical" (205) [default]



"twoBladeHorizontal" (206)



"fourBlade" (207)

**"singleSweepWipe"**



"clockwiseTop" (221) [default]



"clockwiseRight" (222)



"clockwiseBottom" (223)



"clockwiseLeft" (224)



"clockwiseTopLeft" (241)



"counterClockwiseBottomLeft" (242)

SMPTE Wipe #243
"clockwiseBottomRight" (243)



"counterClockwiseTopRight" (244)

**"fanWipe"**






"centerTop" (211) [default]


"centerRight" (212)


"top" (231)


"right" (232)


"bottom" (233)


"left" (234)

**"doubleFanWipe"**


"fanOutVertical" (213) [default]


"fanOutHorizontal" (214)


"fanInVertical" (235)


"fanInHorizontal" (236)

**"doubleSweepWipe"**


"parallelVertical" (225) [default]


"parallelDiagonal" (226)


"oppositeVertical" (227)


"oppositeHorizontal" (228)


"parallelDiagonalTopLeft" (245)


"parallelDiagonalBottomLeft" (246)

**"saloonDoorWipe"**


"top" (251) [default]


"left" (252)


"bottom" (253)


"right" (254)

**"windshieldWipe"**







"right" (261) [default]



"up" (262)



"vertical" (263)

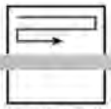


"horizontal" (264)

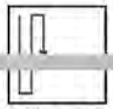
### 12.8.5 Table 5: SMPTE Matrix Wipes

Matrix wipes reveal media in squares following a pattern. The arrow ➙ shows the pattern.

**"snakeWipe"**



"topLeftHorizontal" (301) [default]



"topLeftVertical" (302)



"topLeftDiagonal" (303)



"topRightDiagonal" (304)

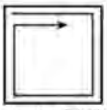


"bottomRightDiagonal" (305)

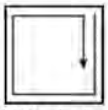


"bottomLeftDiagonal" (306)

**"spiralWipe"**



"topLeftClockwise" (310) [default]



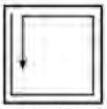
"topRightClockwise" (311)



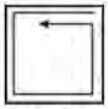
"bottomRightClockwise" (312)



"bottomLeftClockwise" (313)



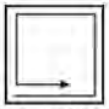
"topLeftCounterClockwise" (314)



"topRightCounterClockwise" (315)

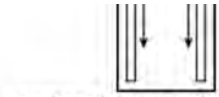


"bottomRightCounterClockwise" (316)

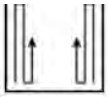


"bottomLeftCounterClockwise" (317)

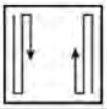
**"parallelSnakesWipe"**

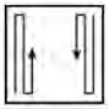


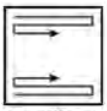
"verticalTopSame" (320) [default]



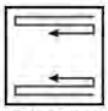
"verticalBottomSame" (321)



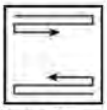




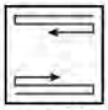
"horizontalLeftSame" (324)



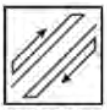
"horizontalRightSame" (325)



"horizontalTopLeftOpposite" (326)



"horizontalTopRightOpposite" (327)

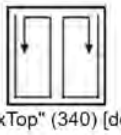


"diagonalBottomLeftOpposite" (328)

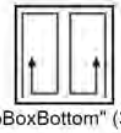


"diagonalTopLeftOpposite" (329)

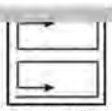
**"boxSnakesWipe"**



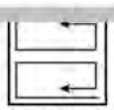
"twoBoxTop" (340) [default]



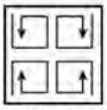
"twoBoxBottom" (341)



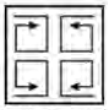
"twoBoxLeft" (342)



"twoBoxRight" (343)



"fourBoxVertical (344)



"fourBoxHorizontal" (345)

**"waterfallWipe"**



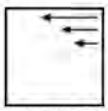
"verticalLeft" (350) [default]



"verticalRight" (351)



"horizontalLeft" (352)



"horizontalRight" (353)

## 13. SMIL 2.0 Language Profile

***Editors:***
Nabil Layaïda (Nabil.Layaida@inrialpes.fr), INRIA
Jacco Van Ossenbruggen (Jacco.van.Ossenbruggen@cwi.nl), CWI

## 13.1 Abstract

The SMIL 2.0 profile describes the SMIL 2.0 modules that are included in the SMIL 2.0 Language and details how these modules are integrated. It contains support for all of the major SMIL 2.0 features including animation, content control, layout, linking, media object, meta-information, structure, timing and transition effects. It is designed for Web clients that support direct playback from SMIL 2.0 markup.

## 13.2 SMIL 2.0 Profile

This section is *informative*.

The SMIL 2.0 Profile is defined as a markup language. The syntax of this language is formally described with a document type definition (DTD) or an XML Schema which is based on the SMIL modules as defined in "The SMIL 2.0 Modules".

The SMIL 2.0 Profile design requirements are:

1. Ensure that the profile is completely backward compatible with SMIL 1.0.
2. Ensure that all the modules' semantics maintain compatibility with SMIL semantics (this includes content and timing).
3. Adopt new W3C Recommendations when appropriate without conflicting with other requirements.

## 13.3 Normative Definition of the SMIL 2.0 Language Profile

This section is *normative*.

### 13.3.1 Document Conformance

This version of SMIL provides a definition of strictly conforming SMIL 2.0 documents, which are restricted to tags and attributes from the SMIL 2.0 namespace. The Section "Extending SMIL 2.0 Language" provides information on using SMIL 2.0 with other namespaces, for instance, on including new tags within SMIL 2.0 documents.

### 13.3.2 SMIL 2.0 Language Conformance

**Conforming SMIL 2.0 Documents**

A SMIL 2.0 document is a *conforming* SMIL 2.0 document if it adheres to the specification described in this document (Synchronized Multimedia Integration Language (SMIL) 2.0 Profile Specification) including SMIL 2.0's DTD (see Document Type Definition). A conforming SMIL 2.0 document must meet all of the following criteria:

1. The root element of the document must be the **smil** element.
2. The document must be well-formed XML.
3. It must conform to the following W3C Recommendations:
   - The XML 1.0 specification (Extensible Markup Language (XML) 1.0) [XML10].
   - Namespaces in XML [XML-NS]. Full XML Namespaces must be supported.
   - Any use of CSS styles and properties shall conform to Cascading Style Sheets, level 2 CSS2 Specification [CSS2].
4. A SMIL 2.0 document can contain the following DOCTYPE declaration:

   The SMIL 2.0 Language DOCTYPE is:

   ```
   <!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
                         "http://www.w3.org/2001/SMIL20/SMIL20.dtd">
   ```

   If a document contains this declaration, it must be a valid XML document. Note that this implies that extensions to the syntax defined in the DTD are not allowed. If the document is invalid, the user agent should issue an error.

5. A document must declare a default namespace for its elements with an xmlns attribute on the **smil** root element with its identifier URI:

   ```
   <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
     ...
   </smil>
   ```

   The default namespace declaration must be xmlns="http://www.w3.org/2001/SMIL20/Language".

6. A document may declare both an XML DOCTYPE declaration (as defined in [XML10]) and one or more XML namespace declarations (as defined in [XML-NS]). To be recognized as a SMIL 2.0 document by a conforming SMIL 2.0 user agent, the document must include the SMIL 2.0 namespace identifier as the default namespace on the **<smil>** tag. For example:

   Declare a SMIL 2.0 document with custom extensions conforming to a custom DTD:

   ```
   <!DOCTYPE smil SYSTEM "http://www.example.org/myveryownSMIL.dtd">
   <smil xmlns="http://www.w3.org/2001/SMIL20/Language"
         xmlns:mysmil="http://www.example.org/2001/SMIL30/Language">
         <mysmil:foo>
         ...
         </mysmil:foo>
   </smil>
   ```

7. A document that declares neither a DOCTYPE nor a default namespace declaration will be processed as a SMIL 1.0 document. Extension elements or attributes not defined by the SMIL 1.0 namespace should be declared using the XML namespace mechanism.

8. Given that, as of this writing, DTDs have no way to describe the allowability of namespace-qualified extensions, and that extensions to SMIL 2.0 conformant documents must be namespace-qualified, here is the algorithm to be used to validate documents with extensions:

   If all non-SMIL 2.0 namespace elements and attributes and all xmlns attributes which refer to non-SMIL 2.0 namespace elements are removed from the given document and if the appropriate <!DOCTYPE ... > statement which points to the SMIL 2.0 DTD is included, the result is a valid XML document.

9. Many SMIL 2.0 attributes are associated with several SMIL 2.0 namespaces including the module namespaces, module collection namespaces, the SMIL 2.0 language namespace, and the overall SMIL 2.0 namespace. Attributes in the SMIL 2.0 namespaces having the same local part are considered the same attribute as far as SMIL 2.0 Language document syntax. It is illegal to use a SMIL 2.0 attribute several times on an element, even if each occurrence is qualified by a different SMIL 2.0 namespace, and SMIL user agents shall treat this as a syntax error. For example, the following document fragment is an error, because the begin attribute appears twice on the ref element:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language"
      xmlns:BasicInlineTiming="http://www.w3.org/2001/SMIL20/BasicInlineTiming">
  ...
  <ref ... begin="..." ... BasicInlineTiming:begin="..." ... />
  ...
</smil>
```

The SMIL 2.0 language or these conformance criteria provide no designated size limits on any aspect of SMIL 2.0 content. There are no maximum values on the number of elements, the amount of character data, or the number of characters in attribute values.

SMIL 2.0 deprecates **base** as a property value for the content attribute of the meta element of SMIL 1.0 in favor of the more general XML Base URI mechanisms.

The SMIL 2.0 Language profile supports the XML Base Recommendation [XMLBase]. XML Base is supported on all elements, and affects the interpretation of URIs as specified in the individual modules defining the URI attributes. Specifically, any applicable XML Base base URI must be applied to the interpretation of the href attribute of the link elements a, area and anchor, as well as the src attribute of the media elements audio, video, img, animation, textstream, text, and ref. XML Base must also be applied on longdesc attributes of all of the SMIL 2.0 Language elements.

The rules above should be revised once a normative XML Schema for SMIL 2.0 is available. This revision will take into account XML Schema validation.

**Conforming SMIL 2.0 Language User Agents**

A SMIL 2.0 user agent is a program which can parse and process a SMIL 2.0 document and render the contents of the document onto output mediums. A conforming SMIL 2.0 user agent must meet all of the following criteria:

1. In order to be consistent with the XML 1.0 Recommendation [XML10], the user agent must parse and evaluate a SMIL 2.0 document for well-formedness. If the user agent claims to be a validating user agent, it must also validate documents against their referenced DTDs according to [XML10].
2. When the user agent claims to support the functionality of SMIL 2.0 through SMIL 2.0 elements and attributes and the semantics associated with these elements and attributes, it must do so in ways consistent with this specification.
3. The user agent must be able to successfully parse and process any conforming SMIL 2.0 documents, and support and correctly implement the semantics of all SMIL 2.0 Language Profile features.
4. The XML parser of the user agent must be able to parse and process XML constructs defined in [XML10] and [XML-NS].
5. The default namespace on the smil root element recognized by the user agent will fall into one of three types:
    1. Default namespace on the smil root element recognized in its entirety by the user agent. The user agent should process the document as the version identified by the recognized namespace. Any elements, attributes, or other syntax not defined by the default namespace on the smil root element must be fully namespace qualified using the standard XML mechanisms for declaring namespaces for elements and attributes described in [XML-NS]. The "skip-content" mechanism (defined in the SkipContent module) will be applied to extension elements unrecognized by the user agent. Unqualified elements not part of the default namespace are illegal and must result in an error.

       Examples:

       1) A pure SMIL 1.0 document:

       ```
       <smil xmlns="http://www.w3.org/TR/REC-smil">
         ...
       </smil>
       ```

       2) A pure SMIL 2.0 document:

       ```
       <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
         ...
       </smil>
       ```

       3) A SMIL 1.0 document that has been extended to use the excl element:

       ```
       <smil xmlns="http://www.w3.org/TR/REC-smil"
             xmlns:smil20="http://www.w3.org/2001/SMIL20/" >
         <smil20:excl>
           ...
         </smil20:excl>
       </smil>
       ```

       4) A SMIL 2.0 document that has been extended to use the 'foo' element from a fictitious SMIL 3.0 version of SMIL:

       ```
       <smil xmlns="http://www.w3.org/2001/SMIL20/Language"
             xmlns:smil30="http://www.example.org/2001/SMIL30/" >
         <smil30:foo>
           ...
         </smil30:foo>
       </smil>
       ```

    2. No default namespace declaration is present on the smil root element in the document. The document will be processed as a SMIL 1.0 document.
    3. Default namespace declaration on the smil root element unrecognized. A SMIL user agent will not recognize the document as any version of SMIL and must issue an error.
6. The user agent must support the following W3C Recommendations with regard to SMIL 2.0 content:
    - Complete support for the XML 1.0 specification (Extensible Markup Language (XML) 1.0) [XML10].
    - Complete support for inclusion of non-SMIL 2.0 namespaces within SMIL 2.0 content via Namespaces in XML [XML-NS]. A xmlns declaration or xmlns prefix declaration may be used on any element in the SMIL 2.0 language profile.
7. The user agent ignores namespace prefix qualified elements from unrecognized namespaces and must support the skip-content attributes. If no skip-content attributes are declared, the value of true is assumed.
8. The user agent ignores elements with unrecognized default namespace declarations and must support the skip-content attribute. If no skip-content attributes are declared, the value of true is assumed.
9. The user agent must issue an error for an attribute value which does not conform to the syntax specified for that attribute.
10. The detection of a syntax error in a SMIL 2.0 language profile document must result in the user agent issuing an error and not playing the document.
11. When the deprecated (hyphenated) and the new (camelCase) version of an attribute syntax are used in the same document, SMIL 2.0 user

agents should take into account the camelCase version only.
12. Conforming SMIL 2.0 Language user agents must play SMIL 1.0 documents as specified in the SMIL 1.0 Recommendation [SMIL10].

The Web Accessibility Initiative is defining "User Agent Accessibility Guidelines 1.0" [UAAG]. Developers are encouraged to design user agents that satisfy at least the Level A requirements of that document. Should UAAG 1.0 become a W3C Recommendation, a future version of SMIL is likely to require Level A conformance to UAAG 1.0 as part of SMIL user agent conformance.

### 13.3.3 The SMIL 2.0 Language Profile

The SMIL 2.0 Language Profile supports the timeline-centric multimedia features found in the SMIL 2.0 modules. It uses only modules from the SMIL 2.0 recommendation. As the language profile includes the mandatory modules, it is a SMIL Host Language conforming language profile. This language profile includes the following SMIL 2.0 modules:

- Structure functionality
  - Structure Module
- Metainformation functionality
  - Metainformation Module
- Timing functionality
  - BasicInlineTiming Module
  - SyncbaseTiming Module
  - EventTiming Module
  - MinMaxTiming Module
  - RepeatTiming Module
  - RepeatValueTiming Module
  - WallclockTiming Module
  - MultiArcTiming Module
  - AccessKeyTiming Module
  - BasicTimeContainers Module
  - ExclTimeContainers Module
  - RestartTiming Module
  - SyncBehavior Module
  - SyncBehaviorDefault Module
  - RestartDefault Module
  - FillDefault Module
  - MediaMarkerTiming Module
- Animation functionality
  - BasicAnimation Module
- Content Control functionality
  - BasicContentControl Module
  - CustomTestAttributes Module
  - PrefetchControl Module
  - SkipContentControl Module
- Layout functionality
  - BasicLayout Module
  - AudioLayout Module
  - MultiWindowLayout Module
  - HierarchicalLayout Module
- Linking functionality
  - BasicLinking Module
  - LinkingAttributes Module
  - ObjectLinking Module
- Transition Effects functionality
  - BasicTransitions Module
  - TransitionModifiers Module
- Media functionality
  - BasicMedia Module
  - MediaClipping Module
  - MediaClipMarkers Module
  - MediaParam Module
  - BrushMedia Module
  - MediaAccessibility Module

The collection names contained in the following table define the SMIL 2.0 Profile vocabulary.

| SMIL 2.0 Profile | |
|---|---|
| **Collection Name** | **Elements in Collection** |
| Animation | animate, set, animateMotion, animateColor |
| ContentControl | switch, prefetch |
| Layout | region, root-layout, layout, regPoint, topLayout |
| LinkAnchor | a, area (anchor) |
| MediaContent | text, img, audio, video, ref, animation, textstream, brush, param |
| Metainformation | meta, metadata |
| Structure | smil, head, body |
| Schedule | par, seq, excl |
| Transition | transition |

In the following sections, we define the set of elements and attributes used in each of the modules included in the SMIL 2.0 Profile. The content model for each element is described. The content model of an element is a description of elements which can appear as its direct children. The special content model "EMPTY" means that a given element may not have children.

| Collection Name | Attributes in Collection |
|---|---|
| Core | id (ID), class (NMTOKEN), title (CDATA), alt (CDATA), longdesc (CDATA), xml:base (CDATA) |
| I18n | xml:lang (NMTOKEN) |

The id, class and title attributes in the collection Core are defined for all the elements of the SMIL 2.0 Profile. The id attribute is used in the SMIL 2.0 Language Profile to assign a unique XML identifier to every element in a SMIL document. In this document, equivalent but deprecated attributes and elements in parentheses.

### 13.3.4 Animation Module

The Animation Module provides a framework for incorporating animation into a timing framework, and a mechanism for composing the effects of multiple animations. The Animation Module uses the timing modules included in this profile for the underlying model of time. The SMIL 2.0 profile includes the animation functionality of the BasicAnimation module. The BasicAnimation Module defines the semantics for the animate, set, animateMotion and animateColor elements.

In the SMIL 2.0 Language Profile, Animation elements can have the following attributes and content model :

| Animation Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| animate | Core, I18n, basicTiming, Test, attributeName, attributeType, targetElement, from, to, by, values, calcMode, accumulate, additive, skip-content, customTest, fill (freeze \| remove \| hold \| auto \| default), fillDefault ( remove \| freeze \| hold \| transition \| auto \| inherit ) | EMPTY |
| set | Core, I18n, basicTiming, Test, attributeName, attributeType, targetElement, to, skip-content, customTest, fill (freeze \| remove \| hold \| auto \| default), fillDefault ( remove \| freeze \| hold \| transition \| auto \| inherit ) | EMPTY |
| animateMotion | Core, I18n, basicTiming, Test, targetElement, origin, from, to, by, values, calcMode, accumulate, additive, skip-content, customTest, fill (freeze \| remove \| hold \| auto \| default), fillDefault ( remove \| freeze \| hold \| transition \| auto \| inherit ) | EMPTY |
| animateColor | Core, I18n, basicTiming, Test, attributeName, attributeType, targetElement, from, to, by, values, calcMode, accumulate, additive, skip-content, customTest, fill (freeze \| remove \| hold \| auto \| default), fillDefault ( remove \| freeze \| hold \| transition \| auto \| inherit ) | EMPTY |

This profile adds the animate, set, animateMotion and animateColor elements to the content model of the par, seq, excl and priorityClass elements of the Timing and Synchronization Modules. It also adds these elements to the content model of the body element of the Structure Module.

**Specifying the target element of the animation**

The animation target elements supported in the SMIL 2.0 Profile are the region element defined in the Layout Modules, the area (anchor) element defined in the Linking Modules and the text, img, audio, animation, video, ref, textstream and the brush elements defined in the Media Objects modules.

The SMIL 2.0 Language Profile uses the targetElement attribute to identify the element to be affected by animation elements. As recommended in the BasicAnimation Module when the targetElement attribute is supported, this profile excludes the XLink attributes href, type, actuate and show from the animate, set, animateMotion and animateColor elements.

**Specifying the target attribute of the animation**

The target attributes of the animations are a subset of those of the region, area (anchor), and media elements. The animatable attributes of the region, area (anchor), and media elements are listed in the table below.

The area (anchor) element has the coords attribute which can be subject to animation. The attribute coords is considered of type string in this profile. This means that only discrete non-additive animation is supported on this attribute.

The media elements have the following sub-region attributes which can be subject to animation: left, right, top, bottom, width, height, z-index and backgroundColor.

| Elements | Target Element | Target Attributes |
|---|---|---|
| animate | region | soundLevel, left, right, top, bottom, width, height, z-index, backgroundColor (background-color), regionName |
| | area (anchor) | coords(string) |
| | text, img, audio, animation, video, ref, textstream | left, right, top, bottom, width, height, z-index, backgroundColor |
| | brush | left, right, top, bottom, width, height, z-index, backgroundColor (background-color), color |
| set | region | soundLevel, left, right, top, bottom, width, height, z-index, backgroundColor, regionName |
| | area (anchor) | coords(string) |
| | text, img, audio, animation, video, ref, textstream | left, right, top, bottom, width, height, z-index, backgroundColor |
| | brush | left, right, top, bottom, width, height, z-index, color |
| animateMotion | region | Animates the top and left attributes of the region. |
| | text, img, audio, animation, video, ref, textstream | Animates the top and left attributes of the sub-region associated with the media element. |
| animateColor | region | backgroundColor (background-color) |
| | text, img, audio, animation, video, ref, textstream | backgroundColor |
| | brush | color |

**Integration definitions**

The SMIL 2.0 Language profile defines a set of integration definitions as required by the Animation modules. These definitions are:

- Animation values in the SMIL 2.0 Language profile are subject to the same syntax requirements as the attribute values on the animated elements.
- Percentage values on the attributes have the same semantics as they do on the attribute. If percentage values are not supported on an attribute then percentage values are not supported for animation in the SMIL 2.0 Language profile.
- The SMIL 2.0 Language profile supports CSS2 [CSS2] color names as described in the Layout modules. The lower clamping bound color value is the black color or rgb(0%, 0%, 0%) and the higher clamping color value is the white color or rgb(100%, 100%, 100%). Values on an animated color lower than the lower clamping bound are clamped to the black color; values greater than the higher clamping color are clamped to the white color.
- The semantics of clamping values for attributes should be performed in floating point with a precision of at least that given by a 4-byte IEEE-format real number [IEEE-Arithmetic]. Cumulative and additive presentation values should be computed without regard to the range of the target attribute. Prior to display, the presentation value should be reduced or increased to the nearest value which is within the range of the target attribute. For integer attributes the computed value should then be rounded to the nearest integer (coerced-integer-value). The mathematical definition of rounding is:

```
coerced-integer-value = Math.floor( interpolated-value + 0.5 )
```

- The position of the region element to which the animateMotion element applies is the top left corner, in the coordinate system within which the
- The origin for the animateMotion when applied to the sub-region on media elements is the top left of the containing region as defined in the Layout modules.

### 13.3.5 Content Control Modules

The Content Control Modules provide a framework for selecting content based on a set of test attributes. The Content Control Modules define semantics for the switch, prefetch, customAttributes and customTest elements. The SMIL 2.0 profile includes the Content Control functionality of the BasicContentControl, CustomTestAttributes, PrefetchControl and SkipContentControl modules.

In the SMIL 2.0 Language Profile, Content Control elements can have the following attributes and content model :

| Content Control Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| switch | Core, I18n, Test, customTest | (Schedule \| priorityClass \| MediaContent \| ContentControl \| LinkAnchor \| Animation \| layout )* |
| prefetch | Core, I18n, Test, Timing, mediaSize, mediaTime, bandwidth, src, clipBegin (clip-begin), clipEnd (clip-end), skip-content, customTest | EMPTY |
| customAttributes | Core, I18n, Test, skip-content | customTest+ |
| customTest | Core, I18n, skip-content, defaultState (true\|false) 'false', override (visible \| hidden) 'hidden', uid (URI) | EMPTY |

This profile adds the switch element to the content model of the par, seq and excl elements of the Timing and Synchronization Modules, of the body and the head elements of the Structure Module, of the content model of the a element of the Linking Modules. The profile adds the customAttributes element to the content model of the head and the customTest element to the content model of the customAttributes element.

The Content Control functionality is used to define the Attribute set "Test":

| Collection Name | Attributes in Collection |
|---|---|
| | systemBitrate (system-bitrate), systemCaptions (system-captions), systemLanguage (system-language), system-overdub-or- |
| | systemOverdubOrSubtitle, systemAudioDesc, systemOperatingSystem, systemCPU, systemComponent |

The collection of Attributes Test is added to all the elements defined in the SMIL 2.0 profile, except customTest and customAttributes. A SMIL 2.0 user agent must support all of the values for the systemOperatingSystem and systemCPU attributes listed in the Content Control Modules. In addition, the user agent should accept namespaced values as future extensions, and not declare a syntax error. The user agent should return false for unrecognized values of the systemOperatingSystem and systemCPU attributes.

### 13.3.6 Layout Modules

The Layout Modules provide a framework for spatial layout of visual components. The Layout Modules define semantics for the region, root-layout, topLayout, layout and the regPoint elements. The SMIL 2.0 profile includes the Layout functionality of the BasicLayout, AudioLayout, MultiWindowLayout and HierarchicalLayout modules.

In the SMIL 2.0 Language Profile, Layout elements can have the following attributes and content model :

| Layout Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| region | Core, I18n, Test, backgroundColor (background-color), showBackground (always \| whenActive), bottom, fit (fill \| hidden \| meet \| scroll \| slice), width, height, left, right, top, soundLevel, z-index, skip-content, customTest, regionName | region* |
| root-layout | Core, I18n, Test, backgroundColor (background-color), width, height, skip-content, customTest | EMPTY |
| topLayout | Core, I18n, Test, backgroundColor (background-color), width, height, open, close, skip-content, customTest | region* |
| layout | Core, I18n, Test, type, customTest | (root-layout \| region \| topLayout \| regPoint)* |
| regPoint | Core, I18n, Test, top, bottom, left, right, regAlign ( topLeft\|topMid \| topRight \| midLeft \| center \| midRight \| bottomLeft \| bottomMid \| bottomRight ), skip-content, customTest | EMPTY |

(**) The "background-color" attribute of SMIL1.0 is deprecated in favor of "backgroundColor", but both are supported.

| Collection Name | Attributes in Collection |
|---|---|
| SubregionAttributes | top, left, bottom, right, width, height, z-index, fit, backgroundColor (background-color), regPoint, regAlign |

This profile adds the layout element to the content model of the head element of the Structure Module. It also adds this element to the content model of the switch element of the Content Control Modules, when the switch element is a child of the head element.

### 13.3.7 Linking Modules

The Linking Modules provide a framework for relating documents to content, documents and document fragments. The Linking Modules define semantics for the a and area (anchor) elements. They define also the semantics of a set of attributes defined for these elements. The SMIL 2.0 profile includes the Linking functionality of the BasicLinking, LinkingAttributes and ObjectLinking modules.

Both the a and area elements have an href attribute, whose value must be a valid URI.

Support for URIs with XPointer fragment identifier syntax is not required.

In the SMIL 2.0 Language Profile, Linking elements can have the following attributes and content model :

| Liking Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| a | Core, I18n, basicTiming, Test, href, sourceLevel, destinationLevel, sourcePlaystate (play \| pause \| stop) 'pause', destinationPlaystate (play \| pause \| stop) 'play', show (new \| replace \| pause) 'replace', accesskey, tabindex, target, external, actuate, customTest | (Schedule \| MediaContent \| ContentControl \| Animation )* |
| area (anchor) | Core, I18n, basicTiming, Test, shape, coords, href, nohref, sourceLevel, destinationLevel, sourcePlaystate, destinationPlaystate, show, accesskey, tabindex, target, external, actuate, shape, fragment, skip-content, customTest | (animate \| set)* |

This profile adds the area and a elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Modules. It also adds these elements to the content model of the body element of the Structure Module.

In the SMIL 2.0 language profile, a value of `onLoad` set on the attribute actuate indicates that the link is automatically traversed when the linking element becomes active. For linking elements containing SMIL timing, this is when the active duration of the linking element begins.

The attribute tabindex specifies the position of the element in the tabbing order at a particular instant for the current document. The tabbing order defines the order in which elements will receive focus when navigated by the user via an input device such as a keyboard. At any particular point in time, only active elements are taken into account for the tabbing order; inactive elements are ignored.

When a media object element has a tabindex attribute and becomes active, then its ordered tab index is inserted in the SMIL tab index at the location specified by the media object's tabindex attribute value. This assumes that the media object itself has tab indices, such as embedded HTML with tabindex attributes. This enables all link starting points in a SMIL presentation to have a place on the ordered list to be tab-keyed through, including those in embedded presentations.

For SMIL 1.0 backward compatibility, the anchor element is available but deprecated in favor of area. The anchor element supports the same attributes as area, both the new SMIL 2.0 attributes and the SMIL 1.0 attributes as defined in [SMIL10].

SMIL 1.0 backward compatibility: The show attribute value `pause` is deprecated in favor of setting the show attribute to `new` and the sourcePlaystate attribute to `pause`.

### 13.3.8 Media Object Modules

The Media Object Modules provide a framework for declaring media. The Media Object Modules define semantics for the ref, animation, audio, img, video, text, textstream and brush elements. The SMIL 2.0 profile includes the Media functionality of the BasicMedia, MediaClipping, MediaClipMarkers, MediaParam, BrushMedia and MediaAccessibility modules.

In the SMIL 2.0 Language Profile, media elements can have the following attributes and content model:

| Media Object Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| text, img, audio, animation, video, ref, textstream | Core, I18n, Timing, Test, SubregionAttributes, region, fill (freeze \| remove \| hold \| transition \| auto \| default), author, copyright, abstract, src, type, erase, mediaRepeat, sensitivity, tabindex, customTest, transIn, transOut, clipBegin (clip-begin), clipEnd (clip-end), readIndex | (param \| area (anchor) \| switch \| Animation)* |
| | Core, I18n, Timing, Test, SubregionAttributes, abstract, region, fill (freeze \| remove \| hold \| transition | (param \| area |
| | transIn, transOut, readIndex | Animation)* |
| param | Core, I18n, Test, name, value, valuetype (data \| ref \| object), type, skip-content | EMPTY |

This profile adds the ref, animation, audio, img, video, text, textstream and brush elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Modules. It also adds these elements to the content model of the body element of the Structure Module. It also adds these elements to the content model of the a element of the Linking Modules.

SMIL 1.0 only allowed anchor as a child element of a media element. In addition to anchor, the following elements are now allowed as children of a SMIL media object: area, param, animate, set, animateColor, animateMotion (note that the a element is not included). The switch element is allowed, with the restriction that in this case the content of the switch may only be from the same set of elements.

**Widely Supported MIME Types**

This section is *informative*.

The members of the W3C SYMM Working Group believe that the following MIME types will be widely supported by SMIL user agents:

- audio/basic [MIME-2]
- image/png ([PNG-MIME], [PNG-REC])
- image/jpeg ([MIME-2], [JFIF])

Implementers of SMIL user agents should thus strive to provide support for each of these types. Note, however, that this section is non-normative, and that support for these MIME types is not a precondition for conformance to this specification.

Authors are encouraged to encode media objects using one of the widely supported MIME types whenever possible. This will ensure that their SMIL documents can be played back by a wide range of SMIL user agents.

If authors use a MIME type that is not in the list of widely supported types, they should provide an alternative version encoded using a baseline format. This can be achieved by using a switch element as shown in the following example:

```
<switch>
  <audio src="non-baseline-format-object" />
  <audio src="baseline-format-object" />
</switch>
```

non-baseline format will play the second media object.

**Media Object Integration Requirements**

This section is *normative*.

The MediaParam module defines the erase attribute, and defers definition of the "display area" to the language profile. "Display area" for the purposes of the SMIL 2.0 Language corresponds to a SMIL BasicLayout region. The effects of erase="never" apply after the active duration of the media object and any fill period (defined by SMIL Timing and Synchronization), and only until other media plays to the region targeted by the media object, or until the same media object restarts.

### 13.3.9 Metainformation Module

The Metainformation Module provides a framework for describing a document, either to inform the human user or to assist in automation. The Metainformation Module defines semantics for the meta and metadata elements. The SMIL 2.0 profile includes the Metainformation functionality of the Metainformation module.

In the SMIL 2.0 Language Profile, Metainformation elements can have the following attributes and content model :

| Metainformation Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| meta | Core, I18n, skip-content, content (CDATA), name (CDATA) | EMPTY |
| metadata | Core, I18n, skip-content | EMPTY |

This profile adds the meta element to the content model of the head element of the Structure Module.

The content model of metadata is empty. Profiles that extend the SMIL 2.0 language profile can define the RDF (Resource Description Framework) schema to be used in extending the content model of the metadata element. The Resource Description Framework is defined in the W3C RDF Recommendation [RDFsyntax].

### 13.3.10 Structure Module

The Structure Module provides a framework for structuring a SMIL document. The Structure Module defines semantics for the smil, head, and body elements. The SMIL 2.0 profile includes the Structure functionality of the Structure module.

In the SMIL 2.0 Language Profile, the Structure elements can have the following attributes and content model :

| Structure Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| smil | Core, I18n, Test, xmlns | (head?,body?) |
| head | Core, I18n | (meta*, (customAttributes, meta*)?,(metadata, meta*)?,((layout\|switch),meta*)?, (transition+, meta*)?) |
| body | Core, I18n, Timing, fill, abstract, author, copyright | (Schedule \| MediaContent \| ContentControl \| a )* |

The body element acts as the root element to span the timing tree. The body element has the behavior of a seq element. Timing on the body element is supported. The syncbase of the body element is the application begin time, which is implementation dependent, as is the application end time. Note that the effect of fill on the body element is between the end of the presentation and the application end time, and therefore the effect of fill is implementation dependent.

### 13.3.11 Timing and Synchronization Modules

The Timing and Synchronization Modules provide a framework for describing timing structure, timing control properties and temporal relationships between elements. The Timing and Synchronization Modules define semantics for par, seq, excl and priorityClass elements. In addition, these modules define semantics for attributes including begin, dur, end, repeat (deprecated), repeatCount, repeatDur, syncBehavior, syncTolerance, syncBehaviorDefault, syncToleranceDefault, restartDefault, fillDefault, restart, min, max. The SMIL 2.0 profile includes the Timing functionality of the BasicInlineTiming, SyncbaseTiming, EventTiming, MinMaxTiming, RepeatTiming, RepeatValueTiming, WallclockTiming, MultiArcTiming, AccessKeyTiming, BasicTimeContainers, ExclTimeContainers, RestartTiming, SyncBehavior, SyncBehaviorDefault, RestartDefault and the FillDefault modules.

In the SMIL 2.0 Language Profile, Timing and Synchronization elements can have the following attributes and content model :

| Elements | Attributes | Content model |
|---|---|---|
| par | Core, I18n, Timing, Test, endsync, customTest, fill (freeze \| remove \| hold \| auto \| default), abstract, author, copyright, region | (Schedule \| MediaContent \| ContentControl \| a \| Animation)* |
| seq | Core, I18n, Timing, Test, customTest, fill (freeze \| remove \| hold \| auto \| default), abstract, author, copyright, region | (Schedule \| MediaContent \| ContentControl \| a \| Animation * |
| excl | Core, I18n, Timing, Test, endsync, skip-content, customTest, fill (freeze \| remove \| hold \| auto \| default ), abstract, author, copyright, region | ((Schedule \| MediaContent \| ContentControl \| a \| Animation)* \| priorityClass+) |
| priorityClass | Core, I18n, Test, peers ( stop \| pause\| defer \| never ) 'stop', higher ( stop \| pause ) 'pause', lower ( defer \| never ) 'defer', skip-content, pauseDisplay, customTest, abstract, author, copyright | ((Schedule \| MediaContent \| ContentControl \| a \| Animation)*) |

The Attribute collections Timing and basicTiming are defined as follows:

| Collection Name | Attributes in Collection |
|---|---|
| Timing | begin, dur, end, repeat (deprecated), repeatCount, repeatDur, syncBehavior ( canSlip \| locked \| independent \| default), syncTolerance, syncBehaviorDefault ( canSlip \| locked \| independent \| inherit ) 'inherit', syncToleranceDefault, restartDefault (always \| whenNotActive \| never), fillDefault ( remove \| freeze \| hold \| transition \| auto \| inherit ), restart (always \| whenNotActive \| never \| default), min, max |
| basicTiming | begin, dur, end, repeat (deprecated), repeatCount, repeatDur, min, max |

This profile adds the par, seq, and excl elements to the content model of the body element of the Structure Module and adds these elements to the content model of the a and area elements of the Linking Modules.

Elements of the Media Object Modules have the attributes describing timing and properties of contents.

**Supported Event Symbols**

The SMIL 2.0 Language profile specifies which types of events can be used as part of the begin and end attribute values. The supported events are described as Event-symbols according to the syntax introduced in the SMIL Timing and Synchronization module.

The supported event symbols in the SMIL 2.0 Language Profile are:

| Event | example |
|---|---|
| focusInEvent (In DOM Level 2: "DOMFocusIn") | end="foo.focusInEvent + 3s" |
| focusOutEvent (In DOM Level 2: "DOMFocusOut") | begin="foo.focusOutEvent" |
| activateEvent (In DOM Level 2: "DOMActivate") | begin="foo.activateEvent" |
| beginEvent | begin="foo.beginEvent + 2s" |
| endEvent | end="foo.endEvent + 2s" |
| repeatEvent | end="foo.repeatEvent" |
| inBoundsEvent | end="foo.inBoundsEvent" |
| outOfBoundsEvent | begin="foo.outOfBoundsEvent + 5s" |
| topLayoutCloseEvent | end="toplayout1.topLayoutCloseEvent" |
| topLayoutOpenEvent | end="toplayout2.topLayoutOpenEvent+5s" |

**Event semantics**

**focusInEvent:**
Raised when a media element gets the keyboard focus in its rendering space, i.e., when it becomes the media element to which all subsequent keystroke-event information is passed. Once an element has the keyboard focus, it continues to have it until a user action or DOM method call either removes the focus from it or gives the focus to another media element, or until its rendering space is removed. Only one media element can have the focus at any particular time. The focusInEvent is delivered to media elements only, and does not bubble.

**focusOutEvent:**
Raised when a media element loses the keyboard focus from its rendering space, i.e., when it stops being the media element to which all subsequent keystroke-event information is passed. The focusOutEvent is delivered to media elements only, and does not bubble.

**activateEvent:**
Raised when a media element is activated by user input such as by a mouse click within its visible rendering space or by specific keystrokes when the element has the keyboard focus. The activateEvent is delivered to media elements only, and does not bubble.

**beginEvent:**
Raised when the element actually begins playback of its active duration. If an element does not ever begin playing, this event is never raised. If an element has a repeat count, beginEvent only is raised at the beginning of the first iteration. The beginEvent is delivered to elements which support timing, such as media elements and time containers, and does not bubble.

**endEvent:**
Raised when an element actually ends playback; this is when its active duration is reached or whenever a playing element is stopped. In the following example,

```
<ref id="x" end="30s" src="15s.mpg" />
<ref id="y" end="10s" src="20s.mpg" />
<ref id="z" repeatCount="4" src="5s.mpg" />
```

x.endEvent occurs at roughly 30s when the active duration is reached, y.endEvent occurs at roughly 10s when the playback of the continuous media is ended early by the active duration being reached, and z.endEvent occurs at roughly 20s when the fourth and final repeat has completed, thus reaching the end of its active duration. The endEvent is delivered to elements which support timing, such as media elements and time containers, and does not bubble.

**repeatEvent:**
Fired when the second and subsequent iterations of a repeated element begin playback. An element that has no repeatDur, repeatCount, or repeat attribute but that plays two or more times due to multiple begin times will not raise a repeatEvent when it restarts. Also, children of a time container that repeats will not raise their own repeatEvents when their parent repeats and they begin playing again. The beginEvent is delivered to elements which support timing, such as media elements and time containers, and does not bubble.

**inBoundsEvent:**
Raised when one of the following happens:

- by any input from the mouse or other input device that brings the mouse cursor from outside to within the bounds of a media element's rendering space, regardless of what part, if any, of that rendering space is visible at the time, i.e., z-order is not a factor.
- by any other action that moves the "cursor" or "pointer", as defined by the implementation, from outside to within the bounds of a media element's rendering space, regardless of what part, if any, of that rendering space is visible at the time, i.e., z-order is not a factor. An implementation may decide, for instance, to raise an inBoundsEvent on an element whenever it gets the focus, including when keystrokes give it the focus.

A media element's bounds are restrained by the bounds of the region in which it is contained., i.e., a media element's bounds do not extend

Note that, unlike with keyboard focus which can only be active on one object at a time, the state of being within an object's bounds can be true for multiple objects simultaneously. For instance, if one object is on top of another and the cursor is placed on top of both objects, both would have raised an inBoundsEvent more recently than the raising of any respective outOfBoundsEvent.

**outOfBoundsEvent:**
Raised when one of the following happens:

- by any input from the mouse or other input device that brings the mouse cursor from within to outside the bounds of a media element's rendering space, regardless of what part, if any, of that rendering space is visible at the time,
- by any other action that moves the "cursor" or "pointer", as defined by the implementation, from within to outside the bounds of a media element's rendering space, regardless of what part, if any, of that rendering space is visible at the time.

A media element's bounds are restrained by its region's bounds, i.e., a media element's bounds do not extend beyond its region's bounds. The outOfBoundsEvent is delivered to media elements only, and does not bubble.

**topLayoutCloseEvent**
Raised when a topLayout closes for any reason. This event is delivered to that topLayout. If a topLayout reopens when additional media becomes active in its region(s), this event will be raised again if and when the topLayout closes again, and will be raised every subsequent time it closes. This event is delivered to topLayout elements only and does not bubble.

**topLayoutOpenEvent**
Raised when a topLayout window opens. This event is delivered to that topLayout. If a topLayout closes and then reopens when additional media becomes active in its region(s), this event will be raised again, and will be raised every subsequent time it reopens. This event is delivered to topLayout elements only and does not bubble.

**Order of raising of simultaneous events:**

There will be cases where events occur simultaneously. To ensure that each SMIL 2.0 Language implementation handles them in the same order, the following order must be used to resolve ties:

1. InBoundsEvent
2. focusInEvent (should follow 1)
3. OutOfBoundsEvent
5. focusOutEvent (should follow 3)
6. endEvent
7. beginEvent (must follow 6)
8. repeatEvent
9. topLayoutCloseEvent
10. topLayoutOpenEvent (must follow 9)

Events are listed in order of precedence, e.g., if event #6 in this list occurs at the same time as event #7, then #6 must be raised prior to #7.

The InBoundsEvent, focusInEvent, OutOfBoundsEvent, activateEvent, and focusOutEvent events do not bubble and are delivered to the target media element.

The beginEvent, endEvent and repeatEvent events do not bubble and are delivered to the timed element on which the event occurs.

The topLayoutOpenEvent and topLayoutCloseEvent events do not bubble and are delivered to the topLayout element on which the event occurs.

**Extending the set of supported events**

The SMIL 2.0 Language profile supports an extensible set of events. In order to resolve possible name conflicts with the events that are supported in this profile qualified event names are supported. Namespace prefixes are used to qualify the event names. As a result, the colon is reserved in begin and end attributes for qualifying event names.

For example:

```
<smil ... xmlns:example="http://www.example.com">
  <img id="foo" .../>
  <audio begin="foo.example:focusInEvent".../>
    ...
</smil>
```

**Integration definitions**

A SMIL document's begin time is defined as the moment a user agent begins the timeline for the overall document. A SMIL document's end time is defined as equal to the end time of the body element.

## 13.3.12 Transition Effects Modules

The Transition Modules provide a framework for describing transitions such as fades and wipes. The Transition Modules define semantics for the transition element. The SMIL 2.0 profile includes the functionality of the BasicTransitions and TransitionModifiers modules.

In the SMIL 2.0 Language Profile, Transition Effects elements have the following attributes and content model :

| Transition Effects Module | | |
|---|---|---|
| **Elements** | **Attributes** | **Content model** |
| transition | Core, I18n, Test, dur, type, subtype, startProgress, endProgress, direction, fadeColor, horzRepeat, vertRepeat, borderWidth, borderColor, skip-content, customTest | EMPTY |

This profile adds the transition element to the content model of the head element of the Structure Module.

The Transition Effects Modules add transIn and transOut attributes to ref, animation, audio, img, video, text, textstream and brush elements of the Media Object Modules.

The Transition Effects Modules add the `transition` value to the fill attribute for all elements on which this value of the fill attribute is supported.

## 13.4 Extending the SMIL 2.0 Language

*This section is normative*

In the future, SMIL 2.0 Language may be extended by other W3C Recommendations, or by private extensions. For these extensions, the following rules must be obeyed:

- All elements introduced in extensions must have a skip-content attribute if it should be possible that their content is processed by SMIL 2.0 user agents.
- Private extensions must be introduced by defining a new XML namespace.

Conformant SMIL 2.0 user agents are prepared to handle documents containing extensions that obey these two rules.

## 13.5 Appendix A: SMIL 20 Document Type Definition

This section is *normative*.

The SMIL 2.0 Language Profile Document Type Definition is defined as a set of SMIL 2.0 modules. All SMIL 2.0 modules are integrated according to the guidelines in the W3C Note "Synchronized Multimedia Modules based upon SMIL 1.0" [SMIL-MOD], and defined within their respective module sections.

## 13.6 Appendix B: SMIL 20 XML Schema

This section is *informative*.

Refer to the XML Schema for the SMIL 2.0 language profile.

# 14. SMIL 2.0 Basic Profile and Scalability Framework

***Editors***
Kenichi Kubota (kuboken@isl.mei.co.jp), Panasonic
Aaron Cohen (aaron.m.cohen@intel.com), Intel
Michelle Kim (mykim@us.ibm.com), IBM

## 14.1 Abstract

SMIL 2.0 provides a scalability framework, where a family of scalable SMIL profiles can be defined using subsets of the SMIL 2.0 language profile. A SMIL document can be authored conforming to a scalable SMIL profile such that it provides limited functionality on a resource-constrained device while allowing richer capabilities on a more capable device. SMIL 2.0 Basic (or SMIL Basic) is a profile that meets the needs of resource-constrained devices such as mobile phones and portable disc players. The SMIL Basic profile provides the basis for defining scalable SMIL profiles. SMIL Basic is SMIL host language conformant. It consists of precisely those modules that are required for SMIL host language conformance. This section defines the SMIL guidelines for defining them, and their conformance requirements.

---

## 14.2 Introduction

**This section is *informative*.**

The Synchronized Multimedia Integration Language (SMIL) includes powerful functionality for various multimedia services for customers of various/differing preferences, ranging from desktops to ubiquitous information appliances such as mobile phones, portable disk players, car navigation systems, television sets, and voice user agents. Each of these platforms has its specific capabilities and requirements. It is clear that not all of the SMIL 2.0 features will be required on all platforms. SMIL 2.0 modularization groups semantically related SMIL elements, attributes, and attribute values into a disjoint set of SMIL modules. These modules can then be recombined to produce a SMIL profile that meets the needs of different communities. For example a hand held device or a cell-phone may only support a small subset of SMIL 2.0 modules in its own profile.

As such a SMIL profile allows a SMIL user agent to implement only the subset of the SMIL 2.0 standard it needs, while maintaining document interoperability between devices profiles built for different needs. SMIL 2.0 provides a framework for defining a family of scalable profiles. A scalable profile enables user agents of a wide range of complexity to render from a single, scalable, SMIL document intelligent presentations tailored to the capabilities of the target devices. Conformance to a SMIL Basic profile provides a basis for interoperability guarantees.

The advantages of scalable profiles are:

- Authors can re-purpose SMIL content targeting a wide range of devices that implement SMIL semantics.
- The rendering of the same content can be improved automatically as devices get more powerful.
- All SMIL 2.0 documents can share a document type, a schema, and a set of defined namespaces, and the required default xmlns declaration.
- Any future SMIL 2.0 extensions can easily be incorporated into SMIL documents and user agents.

A scalable profile is defined by extending the SMIL Basic profile, using the content control facilities to support application/device specific features via a namespace mechanism. SMIL Basic is SMIL 2.0 host language conformant. It contains all the SMIL modules required for host language conformance. In fact, it contains only those modules required for host language conformance. A user agent is said to be SMIL Basic conformant if it can render documents at least as complex as those allowed by SMIL Basic. It may render significantly more complex documents to the extent it supports extensions. User agents conforming to the SMIL 2.0 language profile automatically conform to SMIL 2.0 Basic.

## 14.3 SMIL 2.0 Basic Profile

**This section is *normative*.**

### 14.3.1 Definition

SMIL 2.0 Basic Profile consists of the same set of modules that are required for SMIL Host Language Conformance in resource constrained devices. It includes the following SMIL 2.0 Modules:

- SMIL 2.0 Content Control Modules -- BasicContentControl and SkipContentControl
- SMIL 2.0 Layout Modules -- BasicLayout
- SMIL 2.0 Linking Modules -- BasicLinking
- SMIL 2.0 Media Object Modules-- BasicMedia
- SMIL 2.0 Structure Module -- Structure
- SMIL 2.0 Timing and Synchronization Modules -- BasicInlineTiming, MinMaxTiming, BasicTimeContainers, and RepeatTiming

While not strictly required, we strongly encourage languages based upon SMIL Basic to include the SyncbaseTiming module from SMIL 2.0 Timing and Synchronization using the scalable profiles mechanism, and for SMIL Basic User agents to support the module when device constraints are not prohibitive.

**XML Namespace Declarations and Internationalization**

XML namespace declaration using the xmlns and XML internationalization xml:lang attributes are supported on all elements.

### 14.3.2 Conformance of SMIL 2.0 Basic Documents

A SMIL Basic document is a "conforming" SMIL Basic document if it is a conforming SMIL 2.0 document.

### 14.3.3 Conformance of SMIL 2.0 Basic User Agents

A SMIL Basic user agent is a program which can parse SMIL 2.0 documents, process the subset of SMIL 2.0 functionality defined, and render the contents of the document onto output mediums. A conforming SMIL Basic user agent must meet all of the following criteria:

1. In order to be consistent with the XML 1.0 Recommendation [XML10], the user agent must parse and evaluate a SMIL 2.0 document for well-formedness. If the user agent claims to be a validating user agent, it must also validate documents against their referenced DTDs according to [XML10].
2. When the user agent claims to support the functionality of SMIL 2.0 through SMIL 2.0 elements and attributes and the semantics associated with these elements and attributes, it must do so in ways consistent with this specification.
3. The user agent must be able to successfully parse any conforming SMIL 2.0 documents, and correctly implement the semantics of all supported SMIL 2.0 modules.
4. The user agent must be able to successfully parse conforming SMIL 1.0 Documents and correctly implement the semantics of all supported SMIL 2.0 modules in the context of SMIL 1.0 compatible syntax.
5. The user agent must completely support all SMIL 2.0 modules listed as required for host language conformance.
6. The XML parser of the user agent must be able to parse and process XML constructs defined in [XML10] and Namespaces in XML [XML-NS].
7. The user agent must ignore unimplemented attributes and elements. Unimplemented attributes should be treated as if they were not specified. Unimplemented elements should have the skip-content mechanism applied. If no skip-content attribute is declared, the value of `true` is assumed.
8. The user agent must correctly evaluate all SMIL 2.0 namespace URIs when used with the systemRequired and system-required attributes.
9. The default namespace recognized by the user agent will fall into one of three types:
    1. Default namespace recognized in its entirety by the user agent. The user agent should process the document as the recognized version. The skip-content mechanism (defined in the SkipContentControl module) will be applied to extension elements unrecognized by the user agent. Unqualified elements not part of the default namespace are illegal and must result in an error.

       Examples:

       - A pure SMIL 1.0 document:

         ```
         <smil xmlns="http://www.w3.org/TR/REC-smil">
           . . .
         </smil>
         ```

       - A pure SMIL 2.0 document:

         ```
         <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
           . . .
         </smil>
         ```

       - A SMIL 1.0 document that has been extended to use the excl element:

         ```
         <smil xmlns="http://www.w3.org/TR/REC-smil"
               xmlns:smil20="http://www.w3.org/2001/SMIL20">
           <smil20:excl>
             ...
           </smil20:excl>
         </smil>
         ```

       - A SMIL 2.0 document that has been extended to use the 'foo' element from a fictitious SMIL 3.0 version of SMIL:

         ```
         <smil xmlns="http://www.w3.org/2001/SMIL20/Language"
               xmlns:smil30="http://www.example.org/2002/SMIL30/">
         ```

```
<smil30:foo>
  ...
</smil30:foo>
</smil>
```

2. No default namespace declaration is present in the document. The document will be processed as a SMIL 1.0 document.
3. Default namespace unrecognized. A SMIL user agent will not recognize the document as any version of SMIL and must issue an error.

10. The user agent must support the following W3C Recommendations with regard to SMIL 2.0 content:
    - Complete support for the Extensible Markup Language (XML) 1.0 (Second Edition) [XML10].
    - Complete support for inclusion of non-SMIL 2.0 namespaces within SMIL 2.0 content via Namespaces in XML [XML-NS]. An xmlns prefix declaration may be used on any element in the SMIL 2.0 language profile.
    - The user agent must issue an error for an attribute value that does not conform to the syntax specified for that attribute. The detection of a syntax error in a SMIL 2.0 language profile document must result in the user agent issuing an error and not playing the document.

## 14.4 Scalable Profiles

**This section is *normative*.**

A scalable profile enables user agents of a wide range of complexity to render from a single, scalable, SMIL document intelligent presentations tailored to the capabilities of the target devices. A family of scalable SMIL profiles can be built using the Basic Profile plus additional sets of modules geared to the particular needs each profile addresses. A profile specifies scalability using the content control facilities to support application/device specific features via a namespace mechanism. This mechanism will enable the extension and sub-setting of SMIL 2.0 in a uniform way. The SMIL 2.0 namespace may be used with other XML namespaces as per [XML-NS], although such documents are not strictly conforming SMIL 2.0 documents as defined elsewhere. It is expected that future work by W3C may address ways to specify conformance for documents involving multiple namespaces.

### 14.4.1 Definition

A scalable SMIL 2.0 profile contains all the modules in the SMIL Basic Profile as defined above.

Additionally it may contain any one or more modules from the SMIL 2.0 Modules which are not part of SMIL 2.0 Basic.

XML namespace declaration using the xmlns attribute and XML internationalization xml:lang attributes are supported on all elements.

A scalable profile is specified by using the content control facilities via a namespace mechanism as follows:

1. The systemRequired attribute specifies a set of modules that are to be added to SMIL Basic to support the profile. The profile is defined as the union of the modules in the SMIL 2.0 Basic profile and those specified in the systemRequired attribute.
2. The systemRequired attribute can be specified either on the root **smil** element, or specified inline on elements other than the root element.

### 14.4.2 SMIL 2.0 Document Scalability Guidelines

**This section is *informative*.**

Here we present some scalability guidelines for SMIL Basic content authors that will enable their content to be played on the widest range of SMIL Basic and SMIL 2.0 Language conformant devices.

A SMIL 2.0 document must declare a namespace for its elements with its xmlns attribute at the **smil** root element with its identifier URI:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  ...
</smil>
```

The default namespace must be xmlns="**http://www.w3.org/2001/SMIL20/Language**".

With the systemRequired attribute the author may specify what components of SMIL 2.0 are required to render the document.

Examples:

- A document that requires full support of the SMIL 2.0 language profile:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language"
      xmlns:smil20lang="http://www.w3.org/2001/SMIL20/Language"
      systemRequired="smil20lang" >
  ...
</smil>
```

- A document with specific requirements for support of the BasicTransitions and HierarchicalLayout modules:

```
      xmlns:transition="http://www.w3.org/2001/SMIL20/BasicTransitions"
      xmlns:layout="http://www.w3.org/2001/SMIL20/HierarchicalLayout"
      systemRequired="transition+layout" >
  ...
</smil>
```

- A document that explicitly requires support for the HostLanguage module collection. Note that SMIL Basic documents are required to be Host Language conformant, so listing this requirement explicitly is not strictly necessary:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language"
      xmlns:HostLanguage="http://www.w3.org/2001/SMIL20/HostLanguage"
      systemRequired="HostLanguage" >
  ...
</smil>
```

The SMIL namespace URI and module identifiers URIs are defined and reserved in SMIL 2.0 Modules. Namespace prefixes are set by the document author, e.g., "transition", and "layout". This provides authors with a means to express explicitly what additional modules are required to support the document.

The author may choose to explicitly qualify blocks of content with the systemRequired or system-required attributes. The following example contains the systemRequired attribute on the **seq** container within a **switch**, allowing the inclusion of the **brush** element when the "BrushMedia" test succeeds, and providing an image based alternative when the BrushMedia module is not supported:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language"
      xmlns:BrushMedia="http://www.w3.org/2001/SMIL20/BrushMedia" >
  <head>
    <layout>
      <region id="colorbox" top="0px" left="0px" height="50px" width="50px" />
```

```
  </layout>
 </head>
 <body>
  <switch>
   <seq systemRequired="BrushMedia">
    <brush dur="5s" color="#0000FF" region="colorbox"/>
    <brush dur="5s" color="#00FF00" region="colorbox"/>
    <brush dur="5s" color="#FF0000" region="colorbox"/>
   </seq>
   <seq>
    <img dur="5s" src="blue.jpg"  region="colorbox"/>
    <img dur="5s" src="green.jpg" region="colorbox"/>
    <img dur="5s" src="red.jpg"   region="colorbox"/>
   </seq>
  </switch>
 </body>
```

The value of systemRequired attribute refers to the module identifier URI using the xmlns: name prefix, e.g. "BrushMedia". It is preferable that this name, as an identifier of the required module(s), be the module name defined in the SMIL 2.0 specification. The list of the module names is summarized in SMIL 2.0 Modules.

Note that there is an implied difference between the systemRequired on the **smil** element and an "inline" systemRequired on the other SMIL elements. The former is a hard requirement for rendering the document. For example, if the systemRequired on the **smil** element lists a module that the user agent does not support even though the module is not actually used in the document, the document is still prohibited from presentation by that user agent.

Conversely, the use of the systemRequired attribute on other elements only specifies a requirement for rendering a sub-tree of the document. If some of the content of a presentation requires support beyond SMIL Basic and that specified on the systemRequired attribute on the **smil** element, the additional features should be wrapped with the **switch** element and system test attributes, which can then be evaluated by a user agent and be processed accordingly.

### 14.4.3 Conformance of Scalable SMIL 2.0 Documents

A scalable SMIL document must be a conforming SMIL 2.0 document.

### 14.4.4 Conformance of Scalable User Agents

A conforming scalable user agent must conform to the requirements for SMIL Basic User Agents above.

A conforming user agent must implement as a minimum all the modules defined by the SMIL 2.0 Basic Profile.

The **switch** element and the system test attributes can be processed either by the client in the terminal, or by the server on the server side before delivery (perhaps after a CC/PP [CC/PP] negotiation).

# Appendix A. SMIL 2.0 DTDs

A zip archive of all the SMIL 2.0 DTDs is also available

## A.1 SMIL 2.0 Module DTDs:

### A.1.1 The SMIL Animation Module

```
<!-- ================================================================== -->
<!-- SMIL Animation Module  ============================================ -->
<!-- file: SMIL-anim.mod

  This is SMIL 2.0.

       Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
       See http://www.w3.org/Consortium/Legal/.

       Author:     Jacco van Ossenbruggen
       Revision:   2001/07/31  Thierry Michel

    This DTD module is identified by the PUBLIC and SYSTEM identifiers:

    PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Animation//EN"
    SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-anim.mod"

    ================================================================== -->


<!-- ============================= Dependencies ======================== -->
<!-- The integrating profile is expected to define the following entities,
     Unless the defaults provided are sufficient.
 -->

<!-- SMIL.SplineAnimation.module entity:  Define as "INCLUDE" if the integrating
     profile includes the SMIL 2.0 SplineAnimation Module, "IGNORE" if not.
     The default is "IGNORE", i.e. by default SplineAnimation is not included
     in the integrating language profile.
 -->
<!ENTITY % SMIL.SplineAnimation.module "IGNORE">

<!-- Animation depends on SMIL Timing, importing the attributes listed
     in the SMIL.AnimationTime.attrib entity.  If the integrating profile does
     include the SMIL.MinMaxTiming.module, its default value includes the
     attributes defined in SMIL.BasicInlineTiming.attrib and
         SMIL.MinMaxTiming.attrib.  Otherwise, it is defaulted to
         SMIL.BasicInlineTiming.attrib, which is the minimum requirement.

     Note that the profile can override these defaults by redefining
     SMIL.AnimationTime.attrib.  The profile is also expected to define
     SMIL.fill.attrib and SMIL.fillDefault.attrib.
 -->
<!ENTITY % SMIL.MinMaxTiming.module "IGNORE">
<![%SMIL.MinMaxTiming.module;[
 <!ENTITY % SMIL.AnimationTime.attrib "
      %SMIL.BasicInlineTiming.attrib;
      %SMIL.BasicInlineTiming-deprecated.attrib;
      %SMIL.MinMaxTiming.attrib;
```

```
  ">
]]>
<!ENTITY % SMIL.AnimationTime.attrib "%SMIL.BasicInlineTiming.attrib;">
<!ENTITY % SMIL.fill.attrib "">

<!ENTITY % SMIL.animTimingAttrs "
  %SMIL.AnimationTime.attrib;
  %SMIL.fill.attrib;
  %SMIL.fillDefault.attrib;
">

<!-- Language Designer chooses to integrate targetElement or xlink attributes.
     To integrate the targetElement attribute, define the entity
     animation-targetElement as "INCLUDE"; to integrate the XLink attributes,
     define animation-XLinkTarget as "INCLUDE".

     One or the other MUST be defined.  It is strongly recommended that only one
     of the two be defined.
-->

<!ENTITY % SMIL.animation-targetElement "IGNORE">
<![%SMIL.animation-targetElement;[
  <!ENTITY % SMIL.animTargetElementAttr
   "targetElement  IDREF  #IMPLIED"
  >
]]>
<!ENTITY % SMIL.animTargetElementAttr "">

<!ENTITY % SMIL.animation-XLinkTarget "IGNORE">
<![%SMIL.animation-XLinkTarget;[
  <!ENTITY % SMIL.animTargetElementXLink "
    actuate        (onRequest|onLoad)                'onLoad'
    href           %URI.datatype;                    #IMPLIED
    show           (new | embed | replace)           #FIXED 'embed'
    type           (simple | extended | locator | arc) #FIXED 'simple'
">
]]>
<!ENTITY % SMIL.animTargetElementXLink "">

<!-- ========================== Attribute Groups ========================== -->

<!-- All animation elements include these attributes -->
<!ENTITY % SMIL.animAttrsCommon
 "%Core.attrib;
  %I18n.attrib;
  %SMIL.Test.attrib;
  %SMIL.animTimingAttrs;
  %SMIL.animTargetElementAttr;
  %SMIL.animTargetElementXLink;"
>

<!-- All except animateMotion need an identified target attribute -->
<!ENTITY % SMIL.animAttrsNamedTarget
 "%SMIL.animAttrsCommon;
  attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED"
>

<!-- All except set support the full animation-function specification,
     additive and cumulative animation.
     SplineAnimation adds the attributes keyTimes, keySplines and path,
         and the calcMode value "spline", to those of BasicAnimation.
 -->
<![%SMIL.SplineAnimation.module;[
  <!ENTITY % SMIL.splineAnimCalcModeValues "| spline">
  <!ENTITY % SMIL.splineAnimValueAttrs
   "keyTimes CDATA #IMPLIED
    keySplines CDATA #IMPLIED"
  >
  <!ENTITY % SMIL.splineAnimPathAttr
   "path CDATA #IMPLIED"
  >
]]>
<!ENTITY % SMIL.splineAnimCalcModeValues "">
<!ENTITY % SMIL.splineAnimValueAttrs "">
<!ENTITY % SMIL.splineAnimPathAttr "">

<!ENTITY % SMIL.animValueAttrs "
  %SMIL.BasicAnimation.attrib;
  calcMode   (discrete|linear|paced %SMIL.splineAnimCalcModeValues;) 'linear'
  %SMIL.splineAnimValueAttrs;
  additive   (replace | sum) 'replace'
  accumulate (none | sum) 'none'"
>


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

<!ENTITY % SMIL.animate.attrib  "">
<!ENTITY % SMIL.animate.content "EMPTY">
<!ENTITY % SMIL.animate.qname   "animate">
<!ELEMENT %SMIL.animate.qname; %SMIL.animate.content;>
<!ATTLIST %SMIL.animate.qname; %SMIL.animate.attrib;
  %SMIL.animAttrsNamedTarget;
  %SMIL.animValueAttrs;
>

<!ENTITY % SMIL.set.attrib  "">
<!ENTITY % SMIL.set.content "EMPTY">
<!ENTITY % SMIL.set.qname   "set">
<!ELEMENT %SMIL.set.qname; %SMIL.set.content;>
<!ATTLIST %SMIL.set.qname; %SMIL.set.attrib;
  %SMIL.animAttrsNamedTarget;
  to  CDATA  #IMPLIED
>

<!ENTITY % SMIL.animateMotion.attrib  "">
<!ENTITY % SMIL.animateMotion.content "EMPTY">
<!ENTITY % SMIL.animateMotion.qname   "animateMotion">
<!ELEMENT %SMIL.animateMotion.qname; %SMIL.animateMotion.content;>
<!ATTLIST %SMIL.animateMotion.qname; %SMIL.animateMotion.attrib;
  %SMIL.animAttrsCommon;
  %SMIL.animValueAttrs;
  %SMIL.splineAnimPathAttr;
  origin  (default)  "default"
>


<!ENTITY % SMIL.animateColor.attrib  "">
<!ENTITY % SMIL.animateColor.content "EMPTY">
<!ENTITY % SMIL.animateColor.qname   "animateColor">
<!ELEMENT %SMIL.animateColor.qname; %SMIL.animateColor.content;>
<!ATTLIST %SMIL.animateColor.qname; %SMIL.animateColor.attrib;
  %SMIL.animAttrsNamedTarget;
```

```
%SMIL.animValueAttrs;
>

<!-- ========================== End Animation ============================== -->
<!-- end of SMIL-anim.mod -->
```

## A.1.2 The SMIL Content Control Module

```
<!-- ================================================================== -->
<!-- SMIL Content Control Module  ===================================== -->
<!-- file: SMIL-control.mod

     This is SMIL 2.0.

       Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
       See http://www.w3.org/Consortium/Legal/.

       Author:     Jacco van Ossenbruggen, Aaron Cohen
       Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Content Control//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-control.mod"

     ================================================================== -->

<!ENTITY % SMIL.BasicContentControl.module "INCLUDE">
<![%SMIL.BasicContentControl.module;[
  <!ENTITY % SMIL.switch.attrib "">
  <!ENTITY % SMIL.switch.content "EMPTY">
  <!ENTITY % SMIL.switch.qname "switch">

  <!ELEMENT %SMIL.switch.qname; %SMIL.switch.content;>
  <!ATTLIST %SMIL.switch.qname; %SMIL.switch.attrib;
        %Core.attrib;
        %I18n.attrib;
  >
]]>

<!-- ========================== CustomTest Elements ========================== -->
<!ENTITY % SMIL.CustomTestAttributes.module "IGNORE">
<![%SMIL.CustomTestAttributes.module;[

  <!ENTITY % SMIL.customTest.attrib "">
  <!ENTITY % SMIL.customTest.qname "customTest">
  <!ENTITY % SMIL.customTest.content "EMPTY">
  <!ELEMENT %SMIL.customTest.qname; %SMIL.customTest.content;>
  <!ATTLIST %SMIL.customTest.qname; %SMIL.customTest.attrib;
        defaultState (true|false)                   'false'
        override     (visible|hidden)               'hidden'
        uid          %URI.datatype;                 #IMPLIED
        %Core.attrib;
        %I18n.attrib;
  >
  <!ENTITY % SMIL.customAttributes.attrib "">
  <!ENTITY % SMIL.customAttributes.qname "customAttributes">
  <!ENTITY % SMIL.customAttributes.content "(customTest+)">
  <!ELEMENT %SMIL.customAttributes.qname; %SMIL.customAttributes.content;>
  <!ATTLIST %SMIL.customAttributes.qname; %SMIL.customAttributes.attrib;
        %I18n.attrib;
  >

]]> <!-- end of CustomTestAttributes -->

<!-- ========================== PrefetchControl Elements ===================== -->
<!ENTITY % SMIL.PrefetchControl.module "IGNORE">
<![%SMIL.PrefetchControl.module;[
  <!ENTITY % SMIL.prefetch.attrib "">
  <!ENTITY % SMIL.prefetch.qname "prefetch">
  <!ENTITY % SMIL.prefetch.content "EMPTY">
  <!ELEMENT %SMIL.prefetch.qname; %SMIL.prefetch.content;>
  <!ATTLIST %SMIL.prefetch.qname; %SMIL.prefetch.attrib;
        src           %URI.datatype;    #IMPLIED
        mediaSize     CDATA             #IMPLIED
        mediaTime     CDATA             #IMPLIED
        bandwidth     CDATA             #IMPLIED
        %Core.attrib;
        %I18n.attrib;
  >
]]>

<!-- end of SMIL-control.mod -->
```

## A.1.3 The SMIL Layout Module

```
<!-- ================================================================== -->
<!-- SMIL 2.0 Layout Modules ========================================== -->
<!-- file: SMIL-layout.mod

       This is SMIL 2.0.

       Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
       See http://www.w3.org/Consortium/Legal/.

       Author:     Jacco van Ossenbruggen, Aaron Cohen
       Revision:   2001/07/31  Thierry Michel

       This DTD module is identified by the PUBLIC and SYSTEM identifiers:

       PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Layout//EN"
       SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-layout.mod"

       ================================================================== -->

<!-- ================== BasicLayout ================================= -->
<!-- ================== BasicLayout Profiling Entities ============== -->
<!ENTITY % SMIL.layout.attrib       "">
<!ENTITY % SMIL.region.attrib       "">
<!ENTITY % SMIL.rootlayout.attrib   "">
<!ENTITY % SMIL.layout.content     "EMPTY">
<!ENTITY % SMIL.region.content     "EMPTY">
<!ENTITY % SMIL.rootlayout.content "EMPTY">

<!-- ================== BasicLayout Entities ======================== -->
<!ENTITY % SMIL.common-layout-attrs "
```

```
        height              CDATA    'auto'
        width               CDATA    'auto'
        %SMIL.backgroundColor.attrib;
">

<!ENTITY % SMIL.region-attrs "
        bottom              CDATA    'auto'
        left                CDATA    'auto'
        right               CDATA    'auto'
        top                 CDATA    'auto'
        z-index             CDATA    #IMPLIED
        showBackground      (always|whenActive) 'always'
        %SMIL.fit.attrib;
">

<!-- ================== BasicLayout Elements ============================== -->
<!--
     Layout contains the region and root-layout elements defined by
     smil-basic-layout or other elements defined by an external layout
     mechanism.
-->

<!ENTITY % SMIL.layout.qname "layout">
<!ELEMENT %SMIL.layout.qname; %SMIL.layout.content;>
<!ATTLIST %SMIL.layout.qname; %SMIL.layout.attrib;
        %Core.attrib;
        %I18n.attrib;
        type CDATA 'text/smil-basic-layout'
>

<!-- ================== Region Element ===================================-->
<!ENTITY % SMIL.region.qname "region">
<!ELEMENT %SMIL.region.qname; %SMIL.region.content;>
<!ATTLIST %SMIL.region.qname; %SMIL.region.attrib;
        %Core.attrib;
        %I18n.attrib;
        %SMIL.backgroundColor-deprecated.attrib;
        %SMIL.common-layout-attrs;
        %SMIL.region-attrs;
        regionName CDATA #IMPLIED
>

<!-- ================== Root-layout Element ==============================-->
<!ENTITY % SMIL.root-layout.qname "root-layout">
<!ELEMENT %SMIL.root-layout.qname; %SMIL.rootlayout.content; >
<!ATTLIST %SMIL.root-layout.qname; %SMIL.rootlayout.attrib;
        %Core.attrib;
        %I18n.attrib;
        %SMIL.backgroundColor-deprecated.attrib;
        %SMIL.common-layout-attrs;
>


<!-- ================== AudioLayout ======================================== -->
<!ENTITY % SMIL.AudioLayout.module "IGNORE">
<![%SMIL.AudioLayout.module;[
  <!-- ================== AudioLayout Entities ============================ -->
  <!ENTITY % SMIL.audio-attrs "
        soundLevel                      CDATA    '100&#37;'
  ">

  <!-- ================ AudioLayout Elements ============================== -->
  <!-- ================ Add soundLevel to region element ================== -->
  <!ATTLIST %SMIL.region.qname; %SMIL.audio-attrs;>
]]> <!-- end AudioLayout.module -->


<!-- ================ MultiWindowLayout ==================================== -->
<!ENTITY % SMIL.MultiWindowLayout.module "IGNORE">
<![%SMIL.MultiWindowLayout.module;[
  <!-- ============== MultiWindowLayout Profiling Entities ================ -->
  <!ENTITY % SMIL.topLayout.attrib    "">
  <!ENTITY % SMIL.topLayout.content   "EMPTY">

  <!-- ============== MultiWindowLayout Elements ========================== -->
  <!--================ topLayout element ================================== -->
  <!ENTITY % SMIL.topLayout.qname "topLayout">
  <!ELEMENT %SMIL.topLayout.qname; %SMIL.topLayout.content;>
  <!ATTLIST %SMIL.topLayout.qname; %SMIL.topLayout.attrib;
        %Core.attrib;
        %I18n.attrib;
        %SMIL.common-layout-attrs;
        close               (onRequest|whenNotActive) 'onRequest'
        open                (onStart|whenActive)      'onStart'
  >


<!-- ===================== HierarchicalLayout ============================ -->
<!ENTITY % SMIL.HierarchicalLayout.module "IGNORE">
<![%SMIL.HierarchicalLayout.module;[
  <!-- ========== HierarchicalLayout Profiling Entities =================== -->
  <!ENTITY % SMIL.regPoint.attrib     "">
  <!ENTITY % SMIL.regPoint.content   "EMPTY">

  <!-- ============ HierarchicalLayout Elements ========================== -->
  <!ENTITY % SMIL.regPoint.qname "regPoint">
  <!ELEMENT %SMIL.regPoint.qname; %SMIL.regPoint.content;>
  <!ATTLIST %SMIL.regPoint.qname; %SMIL.regPoint.attrib;
        %Core.attrib;
        %I18n.attrib;
        %SMIL.regAlign.attrib;
        bottom              CDATA    'auto'
        left                CDATA    'auto'
        right               CDATA    'auto'
        top                 CDATA    'auto'
  >
]]> <!-- end HierarchicalLayout.module -->


<!-- end of SMIL-layout.mod -->
```

## A.1.4 The SMIL Linking Module

```
<!-- ================================================================== -->
<!-- SMIL Linking Module  ============================================== -->
<!-- file: SMIL-link.mod

     This is SMIL 2.0.
```

```
        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Author:     Jacco van Ossenbruggen, Lloyd Rutledge, Aaron Cohen
        Revision:   2001/07/31  Thierry Michel

    This DTD module is identified by the PUBLIC and SYSTEM identifiers:

    PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Linking//EN"
    SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-linking.mod"

    ========================================================================= -->

<!-- ======================== LinkingAttributes Entities ==================== -->
<!ENTITY % SMIL.linking-attrs "
        sourceLevel             CDATA               '100&#37;'
        destinationLevel        CDATA               '100&#37;'
        sourcePlaystate         (play|pause|stop)   #IMPLIED
        destinationPlaystate    (play|pause|stop)   'play'
        show                    (new|pause|replace) 'replace'
        accesskey               %Character.datatype; #IMPLIED
        target                  CDATA                #IMPLIED
        external                (true|false)        'false'
        actuate                 (onRequest|onLoad)  'onRequest'
        %SMIL.tabindex.attrib;
">


<!-- ======================== BasicLinking Elements ========================= -->
<!ENTITY % SMIL.BasicLinking.module "IGNORE">
<![%SMIL.BasicLinking.module;[

  <!-- ======================== BasicLinking Entities ======================== -->
  <!ENTITY % SMIL.Shape "(rect|circle|poly|default)">
  <!ENTITY % SMIL.Coords "CDATA">
    <!-- comma separated list of lengths -->

  <!ENTITY % SMIL.a.attrib  "">
  <!ENTITY % SMIL.a.content "EMPTY">
  <!ENTITY % SMIL.a.qname   "a">
  <!ELEMENT %SMIL.a.qname; %SMIL.a.content;>
  <!ATTLIST %SMIL.a.qname; %SMIL.a.attrib;
    %SMIL.linking-attrs;
    href                    %URI.datatype;      #IMPLIED
    %Core.attrib;
    %I18n.attrib;
  >

  <!ENTITY % SMIL.area.attrib  "">
  <!ENTITY % SMIL.area.content "EMPTY">
  <!ENTITY % SMIL.area.qname   "area">
  <!ELEMENT %SMIL.area.qname; %SMIL.area.content;>
  <!ATTLIST %SMIL.area.qname; %SMIL.area.attrib;
    %SMIL.linking-attrs;
    shape                   %SMIL.Shape;            'rect'
    coords                  %SMIL.Coords;           #IMPLIED
    href                    %URI.datatype;      #IMPLIED
    %Core.attrib;
    %I18n.attrib;
  >

  <!ENTITY % SMIL.anchor.attrib  "">
  <!ENTITY % SMIL.anchor.content "EMPTY">
  <!ENTITY % SMIL.anchor.qname  "anchor">
  <!ELEMENT %SMIL.anchor.qname; %SMIL.anchor.content;>
  <!ATTLIST %SMIL.anchor.qname; %SMIL.anchor.attrib;
    %SMIL.linking-attrs;
    shape                   %SMIL.Shape;            'rect'
    coords                  %SMIL.Coords;           #IMPLIED
    href                    %URI.datatype;              #IMPLIED
    nohref                  (nohref)            #IMPLIED
    %Core.attrib;
    %I18n.attrib;
  >
]]> <!-- end of BasicLinking -->

<!-- ======================== ObjectLinking ================================= -->
<!ENTITY % SMIL.ObjectLinking.module "IGNORE">
<![%SMIL.ObjectLinking.module;[

  <!ENTITY % SMIL.Fragment "
    fragment                CDATA               #IMPLIED
  ">

  <!-- ======================== ObjectLinking Elements ======================= -->
  <!-- add fragment attribute to area, and anchor elements -->
  <!ATTLIST %SMIL.area.qname;
     %SMIL.Fragment;
  >

  <!ATTLIST %SMIL.anchor.qname;
     %SMIL.Fragment;
  >
]]>
<!-- ======================== End ObjectLinking ============================ -->

<!-- end of SMIL-link.mod -->
```

### A.1.5 The SMIL Media Object Module

```
<!-- SMIL 2.0 Media Objects Modules ======================================== -->
<!-- file: SMIL-media.mod

    This is SMIL 2.0.

       Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
       See http://www.w3.org/Consortium/Legal/.

       Author:     Rob Lanphier, Jacco van Ossenbruggen
       Revision:   2001/07/31  Thierry Michel


    This DTD module is identified by the PUBLIC and SYSTEM identifiers:

    PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Media Objects//EN"
    SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-media.mod"
```

```
    ============================================================================ -->

<!-- ================== Profiling Entities ================================== -->

<!ENTITY % SMIL.MediaClipping.module "IGNORE">
<![%SMIL.MediaClipping.module;[
  <!ENTITY % SMIL.mo-attributes-MediaClipping "
        %SMIL.MediaClip.attrib;
  ">
]]>
<!ENTITY % SMIL.mo-attributes-MediaClipping "">

<!ENTITY % SMIL.MediaClipping.deprecated.module "IGNORE">
<![%SMIL.MediaClipping.module;[
  <!ENTITY % SMIL.mo-attributes-MediaClipping-deprecated "
        %SMIL.MediaClip.attrib.deprecated;
  ">
  ]]>
<!ENTITY % SMIL.mo-attributes-MediaClipping-deprecated "">

<!ENTITY % SMIL.MediaParam.module "IGNORE">
<![%SMIL.MediaParam.module;[
  <!ENTITY % SMIL.mo-attributes-MediaParam "
        erase        (whenDone|never)   'whenDone'
        mediaRepeat  (preserve|strip)   'preserve'
  ">
  <!ENTITY % SMIL.param.qname "param">
  <!ELEMENT %SMIL.param.qname; EMPTY>

  <!ATTLIST %SMIL.param.qname; %SMIL.param.attrib;
    %Core.attrib;
    name        CDATA          #IMPLIED
    value       CDATA          #IMPLIED
    valuetype   (data|ref|object) "data"
    type        %ContentType.datatype;  #IMPLIED
  >
]]>
<!ENTITY % SMIL.mo-attributes-MediaParam "">

<!ENTITY % SMIL.MediaAccessibility.module "IGNORE">
<![%SMIL.MediaAccessibility.module;[
  <!ENTITY % SMIL.mo-attributes-MediaAccessibility "
        readIndex    CDATA           #IMPLIED
  ">
]]>
<!ENTITY % SMIL.mo-attributes-MediaAccessibility "">

<!ENTITY % SMIL.BasicMedia.module "INCLUDE">
<![%SMIL.BasicMedia.module;[
  <!ENTITY % SMIL.media-object.content "EMPTY">
  <!ENTITY % SMIL.media-object.attrib "">

  <!-- ================ Media Objects Entities ============================== -->

  <!ENTITY % SMIL.mo-attributes-BasicMedia "
        src             CDATA   #IMPLIED
        type            CDATA   #IMPLIED
  ">

  <!ENTITY % SMIL.mo-attributes "
        %Core.attrib;
        %I18n.attrib;
        %SMIL.Description.attrib;
        %SMIL.mo-attributes-BasicMedia;
        %SMIL.mo-attributes-MediaParam;
        %SMIL.mo-attributes-MediaAccessibility;
        %SMIL.media-object.attrib;
  ">

  <!--
     Most info is in the attributes, media objects are empty or
     have children defined at the language integration level:
  -->

  <!ENTITY % SMIL.mo-content "%SMIL.media-object.content;">

  <!-- ================ Media Objects Elements ============================== -->
  <!ENTITY % SMIL.ref.qname        "ref">
  <!ENTITY % SMIL.audio.qname      "audio">
  <!ENTITY % SMIL.img.qname        "img">
  <!ENTITY % SMIL.text.qname       "text">
  <!ENTITY % SMIL.textstream.qname "textstream">
  <!ENTITY % SMIL.animation.qname  "animation">

  <!ENTITY % SMIL.ref.content        "%SMIL.mo-content;">
  <!ENTITY % SMIL.audio.content      "%SMIL.mo-content;">
  <!ENTITY % SMIL.img.content        "%SMIL.mo-content;">
  <!ENTITY % SMIL.video.content      "%SMIL.mo-content;">
  <!ENTITY % SMIL.text.content       "%SMIL.mo-content;">
  <!ENTITY % SMIL.textstream.content "%SMIL.mo-content;">
  <!ENTITY % SMIL.animation.content  "%SMIL.mo-content;">

  <!ELEMENT %SMIL.ref.qname;          %SMIL.ref.content;>
  <!ELEMENT %SMIL.audio.qname;        %SMIL.audio.content;>
  <!ELEMENT %SMIL.img.qname;          %SMIL.img.content;>
  <!ELEMENT %SMIL.video.qname;        %SMIL.video.content;>
  <!ELEMENT %SMIL.text.qname;         %SMIL.text.content;>
  <!ELEMENT %SMIL.textstream.qname;   %SMIL.textstream.content;>
  <!ELEMENT %SMIL.animation.qname;    %SMIL.animation.content;>

  <!ATTLIST %SMIL.img.qname;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.text.qname;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.ref.qname;
        %SMIL.mo-attributes-MediaClipping;
        %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.audio.qname;
        %SMIL.mo-attributes-MediaClipping;
        %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.video.qname;
        %SMIL.mo-attributes-MediaClipping;
        %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.textstream.qname;
        %SMIL.mo-attributes-MediaClipping;
        %SMIL.mo-attributes-MediaClipping-deprecated;
```

```
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.animation.qname;
        %SMIL.mo-attributes-MediaClipping;
        %SMIL.mo-attributes;
  >
]]>
<!ENTITY % SMIL.mo-attributes-BasicMedia "">

<!-- BrushMedia -->
<!ENTITY % SMIL.BrushMedia.module "IGNORE">
<![%SMIL.BrushMedia.module;[
  <!ENTITY % SMIL.brush.attrib "">
  <!ENTITY % SMIL.brush.content "%SMIL.mo-content;">
  <!ENTITY % SMIL.brush.qname "brush">
  <!ELEMENT %SMIL.brush.qname; %SMIL.brush.content;>
  <!ATTLIST %SMIL.brush.qname; %SMIL.brush.attrib;
        %Core.attrib;
        %I18n.attrib;
        %SMIL.Description.attrib;
        %SMIL.mo-attributes-MediaAccessibility;
        %SMIL.mo-attributes-MediaParam;
        %SMIL.media-object.attrib;
        color        CDATA           #IMPLIED
  >
]]>

<!-- end of SMIL-media.mod -->
```

## A.1.6 The SMIL Metainformation Module

```
<!-- ============================================================ -->
<!-- SMIL Metainformation Module  ================================ -->
<!-- file: SMIL-metainformation.mod

     This is SMIL 2.0.

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Author: Thierry Michel, Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

     This module declares the meta and metadata elements types and
     its attributes, used to provide declarative document metainformation.

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Metadata//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-metainformation.mod"

     ============================================================ -->


<!-- ================== Profiling Entities ======================= -->

<!ENTITY % SMIL.meta.content     "EMPTY">
<!ENTITY % SMIL.meta.attrib      "">
<!ENTITY % SMIL.meta.qname       "meta">

<!ENTITY % SMIL.metadata.content "EMPTY">
<!ENTITY % SMIL.metadata.attrib  "">
<!ENTITY % SMIL.metadata.qname   "metadata">

<!-- ================== meta element ============================== -->

<!ELEMENT %SMIL.meta.qname; %SMIL.meta.content;>
<!ATTLIST %SMIL.meta.qname; %SMIL.meta.attrib;
  %Core.attrib;
  %I18n.attrib;
  content CDATA #IMPLIED
  name CDATA #REQUIRED
  >

<!-- ================== metadata element ========================== -->

<!ELEMENT %SMIL.metadata.qname; %SMIL.metadata.content;>
<!ATTLIST %SMIL.metadata.qname; %SMIL.metadata.attrib;
  %Core.attrib;
  %I18n.attrib;
>

<!-- end of SMIL-metadata.mod -->
```

## A.1.7 The SMIL Structure Module

```
<!-- ============================================================================ -->
<!-- SMIL Structure Module  ===================================================== -->
<!-- file: SMIL-struct.mod

     This is SMIL 2.0.

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Structure//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-struct.mod"

     ============================================================================ -->

<!-- ================== SMIL Document Root ======================== -->
<!ENTITY % SMIL.smil.attrib  "" >
<!ENTITY % SMIL.smil.content "EMPTY" >
<!ENTITY % SMIL.smil.qname   "smil" >

<!ELEMENT %SMIL.smil.qname; %SMIL.smil.content;>
<!ATTLIST %SMIL.smil.qname; %SMIL.smil.attrib;
        %Core.attrib;
        %I18n.attrib;
        xmlns %URI.datatype; #REQUIRED
>
```

```
<!-- ================== The Document Head ================================ -->
<!ENTITY % SMIL.head.content "EMPTY" >
<!ENTITY % SMIL.head.attrib  "" >
<!ENTITY % SMIL.head.qname   "head" >

<!ELEMENT %SMIL.head.qname; %SMIL.head.content;>
<!ATTLIST %SMIL.head.qname; %SMIL.head.attrib;
       %Core.attrib;
       %I18n.attrib;
>

<!--================== The Document Body - Timing Root ================== -->
<!ENTITY % SMIL.body.content "EMPTY" >
<!ENTITY % SMIL.body.attrib  "" >
<!ENTITY % SMIL.body.qname   "body" >

<!ELEMENT %SMIL.body.qname; %SMIL.body.content;>
<!ATTLIST %SMIL.body.qname; %SMIL.body.attrib;
       %Core.attrib;
       %I18n.attrib;
>
<!-- end of SMIL-struct.mod -->
```

### A.1.8 The SMIL Timing Module

```
<!-- ============================================================= -->
<!-- SMIL Timing and Synchronization Modules ======================= -->
<!-- file: SMIL-timing.mod

     This is SMIL 2.0.

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Author:     Jacco van Ossenbruggen.
        Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Timing//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-timing.mod"

     ============================================================= -->


<!-- ================== Timing Elements ============================== -->

<!ENTITY % SMIL.BasicTimeContainers.module "IGNORE">
<![%SMIL.BasicTimeContainers.module;[
  <!ENTITY % SMIL.par.content "EMPTY">
  <!ENTITY % SMIL.seq.content "EMPTY">
  <!ENTITY % SMIL.par.attrib  "">
  <!ENTITY % SMIL.seq.attrib  "">
  <!ENTITY % SMIL.seq.qname   "seq">
  <!ENTITY % SMIL.par.qname   "par">

  <!ELEMENT %SMIL.seq.qname; %SMIL.seq.content;>
  <!ATTLIST %SMIL.seq.qname; %SMIL.seq.attrib;
   %Core.attrib;
   %I18n.attrib;
   %SMIL.Description.attrib;
  >

  <!ELEMENT %SMIL.par.qname; %SMIL.par.content;>
  <!ATTLIST %SMIL.par.qname; %SMIL.par.attrib;
   %Core.attrib;
   %I18n.attrib;
   %SMIL.Description.attrib;
  >
]]>  <!-- End of BasicTimeContainers.module -->
```

```
<!ENTITY % SMIL.ExclTimeContainers.module "IGNORE">
<![%SMIL.ExclTimeContainers.module;[
  <!ENTITY % SMIL.excl.content          "EMPTY">
  <!ENTITY % SMIL.priorityClass.content "EMPTY">
  <!ENTITY % SMIL.excl.attrib           "">
  <!ENTITY % SMIL.priorityClass.attrib  "">
  <!ENTITY % SMIL.excl.qname            "excl">
  <!ENTITY % SMIL.priorityClass.qname   "priorityClass">

  <!ELEMENT %SMIL.excl.qname; %SMIL.excl.content;>
  <!ATTLIST %SMIL.excl.qname; %SMIL.excl.attrib;
   %Core.attrib;
   %I18n.attrib;
   %SMIL.Description.attrib;
  >

  <!ELEMENT %SMIL.priorityClass.qname; %SMIL.priorityClass.content;>
  <!ATTLIST %SMIL.priorityClass.qname; %SMIL.priorityClass.attrib;
    peers       (stop|pause|defer|never) "stop"
    higher      (stop|pause)             "pause"
    lower       (defer|never)            "defer"
    pauseDisplay (disable|hide|show )    "show"
    %SMIL.Description.attrib;
    %Core.attrib;
    %I18n.attrib;
  >
]]>  <!-- End of ExclTimeContainers.module -->

<!-- end of SMIL-timing.mod -->
```

### A.1.9 The SMIL Transition Module

```
<!-- ============================================================= -->
<!-- SMIL Transition Module  ======================================= -->
<!-- file: SMIL-transition.mod

     This is SMIL 2.0.

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.
```

```
    Author:     Jacco van Ossenbruggen
    Revision:   2001/07/31  Thierry Michel

    This DTD module is identified by the PUBLIC and SYSTEM identifiers:

    PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Transition//EN"
    SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-transition.mod"

    ========================================================================= -->

<!ENTITY % SMIL.TransitionModifiers.module "IGNORE">
<![%SMIL.TransitionModifiers.module;[
 <!ENTITY % SMIL.transition-modifiers-attrs '
    horzRepeat   CDATA                "0"
    vertRepeat   CDATA                "0"
    borderWidth  CDATA                "0"
    borderColor  CDATA                "black"
 '>
]]> <!-- End of TransitionModifiers.module -->
<!ENTITY % SMIL.transition-modifiers-attrs "">

<!ENTITY % SMIL.BasicTransitions.module "INCLUDE">
<![%SMIL.BasicTransitions.module;[

 <!ENTITY % SMIL.transition-types "(barWipe|boxWipe|fourBoxWipe|barnDoorWipe|
  diagonalWipe|bowTieWipe|miscDiagonalWipe|veeWipe|barnVeeWipe|zigZagWipe|
  barnZigZagWipe|irisWipe|triangleWipe|arrowHeadWipe|pentagonWipe|
  hexagonWipe|ellipseWipe|eyeWipe|roundRectWipe|starWipe|miscShapeWipe|clockWipe|
  pinWheelWipe|singleSweepWipe|fanWipe|doubleFanWipe|doubleSweepWipe|
  saloonDoorWipe|windshieldWipe|snakeWipe|spiralWipe|parallelSnakesWipe|
  boxSnakesWipe|waterfallWipe|pushWipe|slideWipe|fade)"
 >

 <!ENTITY % SMIL.transition-subtypes "(bottom
  |bottomCenter|bottomLeft|bottomLeftClockwise|bottomLeftCounterClockwise|
  bottomLeftDiagonal|bottomRight|bottomRightClockwise|
  bottomRightCounterClockwise|bottomRightDiagonal|centerRight|centerTop|
  circle|clockwiseBottom|clockwiseBottomRight|clockwiseLeft|clockwiseNine|
  clockwiseRight|clockwiseSix|clockwiseThree|clockwiseTop|clockwiseTopLeft|
  clockwiseTwelve|cornersIn|cornersOut|counterClockwiseBottomLeft|
  counterClockwiseTopRight|crossfade|diagonalBottomLeft|
  diagonalBottomLeftOpposite|diagonalTopLeft|diagonalTopLeftOpposite|
  diamond|doubleBarnDoor|doubleDiamond|down|fadeFromColor|fadeToColor|
  fanInHorizontal|fanInVertical|fanOutHorizontal|fanOutVertical|fivePoint|
  fourBlade|fourBoxHorizontal|fourBoxVertical|fourPoint|fromBottom|fromLeft|
  fromRight|fromTop|heart|horizontal|horizontalLeft|horizontalLeftSame|
  horizontalRight|horizontalRightSame|horizontalTopLeftOpposite|
  horizontalTopRightOpposite|keyhole|left|leftCenter|leftToRight|
  oppositeHorizontal|oppositeVertical|parallelDiagonal|
  parallelDiagonalBottomLeft|parallelDiagonalTopLeft|
  topLeft|topLeftClockwise|topLeftCounterClockwise|topLeftDiagonal|
  topLeftHorizontal|topLeftVertical|topRight|topRightClockwise|
  topRightCounterClockwise|topRightDiagonal|topToBottom|twoBladeHorizontal|
  twoBladeVertical|twoBoxBottom|twoBoxLeft|twoBoxRight|twoBoxTop|up|
  vertical|verticalBottomLeftOpposite|verticalBottomSame|verticalLeft|
  verticalRight|verticalTopLeftOpposite|verticalTopSame)"
 >

 <!ENTITY  % SMIL.transition-attrs '
    type         %SMIL.transition-types;     #IMPLIED
    subtype      %SMIL.transition-subtypes;  #IMPLIED
    fadeColor    CDATA                  "black"
    %SMIL.transition-modifiers-attrs;
 '>

 <!ENTITY % SMIL.transition.attrib  "">
 <!ENTITY % SMIL.transition.content "EMPTY">
 <!ENTITY % SMIL.transition.qname   "transition">
 <!ELEMENT %SMIL.transition.qname; %SMIL.transition.content;>
 <!ATTLIST %SMIL.transition.qname; %SMIL.transition.attrib;
    %Core.attrib;
    %I18n.attrib;
    %SMIL.transition-attrs;
    dur           %TimeValue.datatype; #IMPLIED
    startProgress CDATA             "0.0"
    endProgress   CDATA             "1.0"
    direction     (forward|reverse)  "forward"
 >
]]> <!-- End of BasicTransitions.module -->

<!ENTITY % SMIL.InlineTransitions.module "IGNORE">
<![%SMIL.InlineTransitions.module;[

 <!ENTITY % SMIL.transitionFilter.attrib  "">
 <!ENTITY % SMIL.transitionFilter.content "EMPTY">
 <!ENTITY % SMIL.transitionFilter.qname   "transitionFilter">
 <!ELEMENT %SMIL.transitionFilter.qname; %SMIL.transitionFilter.content;>
 <!ATTLIST %SMIL.transitionFilter.qname; %SMIL.transitionFilter.attrib;
    %Core.attrib;
    %I18n.attrib;
    %SMIL.transition-attrs;
    %SMIL.BasicInlineTiming.attrib;
    %SMIL.BasicAnimation.attrib;
    calcMode   (discrete|linear|paced) 'linear'
 >
]]> <!-- End of InlineTransitions.module -->

<!-- end of SMIL-transition.mod -->
```

## A.2 SMIL 2.0 Language Profile:

### A.2.1 SMIL 2.0 Language Profile document model

```
<!-- ============================================================================  -->
<!-- SMIL 2.0 Document Model Module ============================================  -->
<!-- file: smil-model-1.mod

    This is SMIL 2.0.

       Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
       See http://www.w3.org/Consortium/Legal/.

       Author: Warner ten Kate, Jacco van Ossenbruggen, Aaron Cohen
       Revision:   2001/07/31  Thierry Michel

    This DTD module is identified by the PUBLIC and SYSTEM identifiers:
```

```
    PUBLIC "-//W3C//ENTITIES SMIL 2.0 Document Model 1.0//EN"
    SYSTEM "http://www.w3.org/2001/SMIL20/smil-model-1.mod"

    ============================================================== -->

<!--
       This file defines the SMIL 2.0 Language Document Model.
       All attributes and content models are defined in the second
       half of this file.  We first start with some utility definitions.
       These are mainly used to simplify the use of Modules in the
       second part of the file.

-->

<!-- ================= Util: Head ======================================= -->
<!ENTITY % SMIL.head-meta.content       "%SMIL.metadata.qname;">
<!ENTITY % SMIL.head-layout.content     "%SMIL.layout.qname;
                                       | %SMIL.switch.qname;">
<!ENTITY % SMIL.head-control.content    "%SMIL.customAttributes.qname;">
<!ENTITY % SMIL.head-transition.content "%SMIL.transition.qname;+">

<!--=================== Util: Body - Content Control ====================== -->
<!ENTITY % SMIL.content-control "%SMIL.switch.qname; | %SMIL.prefetch.qname;">
<!ENTITY % SMIL.content-control-attrs "%SMIL.Test.attrib;
                                       %SMIL.skip-content.attrib;">

<!--=================== Util: Body - Animation ======================== -->
<!ENTITY % SMIL.animation.elements "%SMIL.animate.qname;
                                  | %SMIL.set.qname;
                                  | %SMIL.animateMotion.qname;
                                  | %SMIL.animateColor.qname;">

<!--=================== Util: Body - Media ======================== -->

<!ENTITY % SMIL.media-object "%SMIL.audio.qname;
                            | %SMIL.video.qname;
                            | %SMIL.animation.qname;
                            | %SMIL.text.qname;
                            | %SMIL.img.qname;
                            | %SMIL.textstream.qname;
                            | %SMIL.ref.qname;
                            | %SMIL.brush.qname;
                            | %SMIL.animation.elements;">

<!--=================== Util: Body - Timing ============================== -->
<!ENTITY % SMIL.BasicTimeContainers.class "%SMIL.par.qname;
                                         | %SMIL.seq.qname;">

<!ENTITY % SMIL.ExclTimeContainers.class "%SMIL.excl.qname;">

<!ENTITY % SMIL.timecontainer.class   "%SMIL.BasicTimeContainers.class;
                                      |%SMIL.ExclTimeContainers.class;">

<!ENTITY % SMIL.timecontainer.content "%SMIL.timecontainer.class;
                                    | %SMIL.media-object;
                                    | %SMIL.content-control;
                                    | %SMIL.a.qname;">

<!ENTITY % SMIL.smil-basictime.attrib "
 %SMIL.BasicInlineTiming.attrib;
 %SMIL.BasicInlineTiming-deprecated.attrib;
 %SMIL.MinMaxTiming.attrib;
">

<!ENTITY % SMIL.timecontainer.attrib "
 %SMIL.BasicInlineTiming.attrib;
 %SMIL.BasicInlineTiming-deprecated.attrib;
 %SMIL.MinMaxTiming.attrib;
 %SMIL.RestartTiming.attrib;
 %SMIL.RestartDefaultTiming.attrib;
 %SMIL.SyncBehavior.attrib;
 %SMIL.SyncBehaviorDefault.attrib;
">

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

<!--
     The actual content model and attribute definitions for each module
     sections follow below.
-->

<!-- ================= Content Control ================================== -->
<!ENTITY % SMIL.BasicContentControl.module  "INCLUDE">
<!ENTITY % SMIL.CustomTestAttributes.module "INCLUDE">
<!ENTITY % SMIL.PrefetchControl.module      "INCLUDE">
<!ENTITY % SMIL.skip-contentControl.module   "INCLUDE">

<!ENTITY % SMIL.switch.content "((%SMIL.timecontainer.class;
                               | %SMIL.media-object;
                               | %SMIL.content-control;
                               | %SMIL.a.qname;
                               | %SMIL.area.qname;
                               | %SMIL.anchor.qname;)*
                               | %SMIL.layout.qname;*)">

<!ENTITY % SMIL.switch.attrib "%SMIL.Test.attrib; %SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.prefetch.attrib "
 %SMIL.timecontainer.attrib;
 %SMIL.MediaClip.attrib;
 %SMIL.MediaClip.attrib.deprecated;
 %SMIL.Test.attrib;
 %SMIL.customTestAttr.attrib;
 %SMIL.skip-content.attrib;
">

<!ENTITY % SMIL.customAttributes.attrib  "%SMIL.Test.attrib; %SMIL.skip-content.attrib;">
<!ENTITY % SMIL.customTest.attrib    "%SMIL.skip-content.attrib;">

<!-- ================= Animation ======================================== -->
<!ENTITY % SMIL.BasicAnimation.module "INCLUDE">

<!-- choose targetElement or XLink: -->
<!ENTITY % SMIL.animation-targetElement "INCLUDE">
<!ENTITY % SMIL.animation-XLinkTarget   "IGNORE">

<!ENTITY % SMIL.animate.content "EMPTY">
<!ENTITY % SMIL.animateColor.content "EMPTY">
<!ENTITY % SMIL.animateMotion.content "EMPTY">
<!ENTITY % SMIL.set.content "EMPTY">

<!ENTITY % SMIL.animate.attrib        "%SMIL.skip-content.attrib; %SMIL.customTestAttr.attrib;">
```

```
<!ENTITY % SMIL.animateMotion.attrib  "%SMIL.skip-content.attrib; %SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.set.attrib            "%SMIL.skip-content.attrib; %SMIL.customTestAttr.attrib;">

<!-- ================== Layout ======================================= -->
<!ENTITY % SMIL.BasicLayout.module        "INCLUDE">
<!ENTITY % SMIL.AudioLayout.module        "INCLUDE">
<!ENTITY % SMIL.MultiWindowLayout.module  "INCLUDE">
<!ENTITY % SMIL.HierarchicalLayout.module "INCLUDE">

<!ENTITY % SMIL.layout.content "(%SMIL.region.qname;
                               | %SMIL.topLayout.qname;
                               | %SMIL.root-layout.qname;
                               | %SMIL.regPoint.qname;)*">
<!ENTITY % SMIL.region.content "(%SMIL.region.qname;)*">
<!ENTITY % SMIL.topLayout.content "(%SMIL.region.qname;)*">
<!ENTITY % SMIL.rootlayout.content "EMPTY">
<!ENTITY % SMIL.regPoint.content "EMPTY">

<!ENTITY % SMIL.layout.attrib          "%SMIL.Test.attrib; %SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.rootlayout.attrib      "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.topLayout.attrib       "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.region.attrib          "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.regPoint.attrib        "%SMIL.content-control-attrs;">

<!-- ================== Linking ====================================== -->
<!ENTITY % SMIL.LinkingAttributes.module "INCLUDE">
<!ENTITY % SMIL.BasicLinking.module      "INCLUDE">
<!ENTITY % SMIL.ObjectLinking.module   "INCLUDE">

<!ENTITY % SMIL.a.content      "(%SMIL.timecontainer.class;|%SMIL.media-object;|
                                %SMIL.content-control;)*">
<!ENTITY % SMIL.area.content   "(%SMIL.animate.qname;| %SMIL.set.qname;)*">
<!ENTITY % SMIL.anchor.content "(%SMIL.animate.qname; | %SMIL.set.qname;)*">

<!ENTITY % SMIL.a.attrib      "%SMIL.smil-basictime.attrib; %SMIL.Test.attrib; %SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.area.attrib   "%SMIL.smil-basictime.attrib; %SMIL.content-control-attrs;">
<!ENTITY % SMIL.anchor.attrib "%SMIL.smil-basictime.attrib; %SMIL.content-control-attrs;">

<!-- ================== Media  ======================================= -->
<!ENTITY % SMIL.BasicMedia.module                    "INCLUDE">
<!ENTITY % SMIL.MediaClipping.module                 "INCLUDE">
<!ENTITY % SMIL.MediaClipping.deprecated.module      "INCLUDE">
<!ENTITY % SMIL.MediaClipMarkers.module              "INCLUDE">
<!ENTITY % SMIL.MediaParam.module                    "INCLUDE">
<!ENTITY % SMIL.BrushMedia.module                    "INCLUDE">
<!ENTITY % SMIL.MediaAccessibility.module            "INCLUDE">

<!ENTITY % SMIL.media-object.content "(%SMIL.animation.elements;
                                     | %SMIL.switch.qname;
                                     | %SMIL.anchor.qname;
                                     | %SMIL.area.qname;
                                     | %SMIL.param.qname;)*">
<!ENTITY % SMIL.media-object.attrib "
  %SMIL.BasicInlineTiming.attrib;
  %SMIL.BasicInlineTiming-deprecated.attrib;
  %SMIL.MinMaxTiming.attrib;
  %SMIL.RestartTiming.attrib;
  %SMIL.RestartDefaultTiming.attrib;
  %SMIL.SyncBehavior.attrib;
  %SMIL.SyncBehaviorDefault.attrib;
  %SMIL.endsync.media.attrib;
  %SMIL.fill.attrib;
  %SMIL.fillDefault.attrib;
  %SMIL.Test.attrib;
  %SMIL.customTestAttr.attrib;
  %SMIL.regionAttr.attrib;
  %SMIL.Transition.attrib;
  %SMIL.backgroundColor.attrib;
  %SMIL.backgroundColor-deprecated.attrib;
  %SMIL.Sub-region.attrib;
  %SMIL.RegistrationPoint.attrib;
  %SMIL.fit.attrib;
  %SMIL.tabindex.attrib;
">

<!ENTITY % SMIL.brush.attrib        "%SMIL.skip-content.attrib;">
<!ENTITY % SMIL.param.attrib        "%SMIL.content-control-attrs;">

<!-- ================== Metadata ===================================== -->
<!ENTITY % SMIL.meta.content     "EMPTY">
<!ENTITY % SMIL.meta.attrib      "%SMIL.skip-content.attrib;">

<!ENTITY % SMIL.metadata.content "EMPTY">
<!ENTITY % SMIL.metadata.attrib  "%SMIL.skip-content.attrib;">

<!-- ================== Structure ==================================== -->
<!ENTITY % SMIL.Structure.module "INCLUDE">
<!ENTITY % SMIL.smil.content "(%SMIL.head.qname;?,%SMIL.body.qname;?)">
<!ENTITY % SMIL.head.content "(
        %SMIL.meta.qname;*,
        ((%SMIL.head-control.content;),   %SMIL.meta.qname;*)?,
        ((%SMIL.head-meta.content;),      %SMIL.meta.qname;*)?,
        ((%SMIL.head-layout.content;),    %SMIL.meta.qname;*)?,
        ((%SMIL.head-transition.content;),%SMIL.meta.qname;*)?
)">
<!ENTITY % SMIL.body.content "(%SMIL.timecontainer.class;|%SMIL.media-object;|

<!ENTITY % SMIL.smil.attrib "%SMIL.Test.attrib;">
<!ENTITY % SMIL.body.attrib "
       %SMIL.timecontainer.attrib;
       %SMIL.Description.attrib;
       %SMIL.fill.attrib;
">

<!-- ================== Transitions ================================== -->
<!ENTITY % SMIL.BasicTransitions.module        "INCLUDE">
<!ENTITY % SMIL.TransitionModifiers.module     "INCLUDE">
<!ENTITY % SMIL.InlineTransitions.module       "IGNORE">

<!ENTITY % SMIL.transition.content "EMPTY">
<!ENTITY % SMIL.transition.attrib "%SMIL.content-control-attrs;">

<!-- ================== Timing ======================================= -->
<!ENTITY % SMIL.BasicInlineTiming.module      "INCLUDE">
<!ENTITY % SMIL.SyncbaseTiming.module         "INCLUDE">
<!ENTITY % SMIL.EventTiming.module            "INCLUDE">
<!ENTITY % SMIL.WallclockTiming.module        "INCLUDE">
<!ENTITY % SMIL.MultiSyncArcTiming.module     "INCLUDE">
<!ENTITY % SMIL.MediaMarkerTiming.module      "INCLUDE">
<!ENTITY % SMIL.MinMaxTiming.module           "INCLUDE">
<!ENTITY % SMIL.BasicTimeContainers.module    "INCLUDE">
<!ENTITY % SMIL.ExclTimeContainers.module     "INCLUDE">
<!ENTITY % SMIL.PrevTiming.module             "INCLUDE">
<!ENTITY % SMIL.RestartTiming.module          "INCLUDE">
```

```
<!ENTITY % SMIL.SyncBehavior.module          "INCLUDE">
<!ENTITY % SMIL.SyncBehaviorDefault.module   "INCLUDE">
<!ENTITY % SMIL.RestartDefault.module        "INCLUDE">
<!ENTITY % SMIL.fillDefault.module           "INCLUDE">

<!ENTITY % SMIL.par.attrib "
       %SMIL.endsync.attrib;
       %SMIL.fill.attrib;
       %SMIL.timecontainer.attrib;
       %SMIL.Test.attrib;
       %SMIL.customTestAttr.attrib;
       %SMIL.regionAttr.attrib;
">
<!ENTITY % SMIL.seq.attrib "
       %SMIL.fill.attrib;
       %SMIL.timecontainer.attrib;
       %SMIL.Test.attrib;
       %SMIL.customTestAttr.attrib;
       %SMIL.regionAttr.attrib;
">
<!ENTITY % SMIL.excl.attrib "
       %SMIL.endsync.attrib;
       %SMIL.fill.attrib;
       %SMIL.timecontainer.attrib;
       %SMIL.Test.attrib;
       %SMIL.customTestAttr.attrib;
       %SMIL.regionAttr.attrib;
       %SMIL.skip-content.attrib;
">
<!ENTITY % SMIL.par.content "(%SMIL.timecontainer.content;)*">
<!ENTITY % SMIL.seq.content "(%SMIL.timecontainer.content;)*">
<!ENTITY % SMIL.excl.content "((%SMIL.timecontainer.content;)*
                             | %SMIL.priorityClass.qname;+)">

<!ENTITY % SMIL.priorityClass.attrib  "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.priorityClass.content "(%SMIL.timecontainer.content;)*">

<!-- end of smil-model-1.mod -->
```

## A.2.2 SMIL 2.0 Language Profile DTD driver

```
<!-- .......................................................................... -->
<!-- SMIL 2.0 DTD  ........................................................... -->
<!-- file: SMIL20.dtd
-->
<!-- SMIL 2.0 DTD

       This is SMIL 2.0.

       Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
       See http://www.w3.org/Consortium/Legal/.

       Author:     Jacco van Ossenbruggen
       Revision:   2001/07/31  Thierry Michel

 This is the driver file for the SMIL 2.0 DTD.

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//DTD SMIL 2.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL20.dtd"

-->

<!ENTITY % NS.prefixed "IGNORE" >
<!ENTITY % SMIL.prefix "" >

<!-- Define the Content Model -->
<!ENTITY % smil-model.mod
    PUBLIC "-//W3C//ENTITIES SMIL 2.0 Document Model 1.0//EN"
           "smil-model-1.mod" >

<!-- Modular Framework Module  ................................... -->
<!ENTITY % smil-framework.module "INCLUDE" >
<![%smil-framework.module;[
<!ENTITY % smil-framework.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN"
            "smil-framework-1.mod" >
%smil-framework.mod;]]>

<!-- The SMIL 2.0 Profile includes the following sections:
            C. The SMIL Animation Module
            D. The SMIL Content Control Module
            G. The SMIL Layout Module
            H. The SMIL Linking Module
            I. The SMIL Media Object Module
            J. The SMIL Metainformation Module
            K. The SMIL Structure Module
            L. The SMIL Timing and Synchronization Module
            M. Integrating SMIL Timing into other XML-Based Languages
            P. The SMIL Transition effects Module

            The SMIL Streaming Media Object Module is optional.
-->

<!--
<!ENTITY % smil-streamingmedia.model "IGNORE">
<![%smil-streamingmedia.model;[
  <!ENTITY % smil-streaming-mod
    PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Streaming Media Objects//EN"
    "SMIL-streamingmedia.mod">
  %smil-streaming-mod;
]]>
-->

<!ENTITY % SMIL.anim-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Animation//EN"
  "SMIL-anim.mod">
<!ENTITY % SMIL.control-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Content Control//EN"
  "SMIL-control.mod">
<!ENTITY % SMIL.layout-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Layout//EN"
  "SMIL-layout.mod" >
<!ENTITY % SMIL.link-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Linking//EN"
  "SMIL-link.mod">
<!ENTITY % SMIL.media-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Media Objects//EN"
```

```
  "SMIL-media.mod">
<!ENTITY % SMIL.meta-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Metainformation//EN"
  "SMIL-metainformation.mod">
<!ENTITY % SMIL.struct-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Structure//EN"
  "SMIL-struct.mod">
<!ENTITY % SMIL.timing-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Timing//EN"
  "SMIL-timing.mod">
<!ENTITY % SMIL.transition-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Transition//EN"
  "SMIL-transition.mod">

%SMIL.struct-mod;
%SMIL.anim-mod;
%SMIL.control-mod;
%SMIL.meta-mod;
%SMIL.layout-mod;
%SMIL.link-mod;
%SMIL.media-mod;
%SMIL.timing-mod;
%SMIL.transition-mod;

<!-- end of SMIL20.dtd -->
```

## A.3 General modularization framework:

### A.3.1 SMIL 2.0 common datatypes

```
<!-- ..................................................................... -->
<!-- SMIL 2.0 Datatypes Module  ........................................... -->
<!-- file: smil-datatypes-1.mod

     This is SMIL 2.0.

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Author:     Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Datatypes 1.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/smil-datatypes-1.mod"

     ..................................................................... -->

<!-- Datatypes

     defines containers for the following datatypes, many of
     these imported from other specifications and standards.
-->

<!ENTITY % Character.datatype "CDATA">
    <!-- a single character from [ISO10646] -->
<!ENTITY % ContentType.datatype "CDATA">
    <!-- media type, as per [RFC2045] -->
<!ENTITY % LanguageCode.datatype "NMTOKEN">
    <!-- a language code, as per [RFC1766] -->
<!ENTITY % LanguageCodes.datatype "CDATA">
    <!-- comma-separated list of language codes, as per [RFC1766] -->
<!ENTITY % Number.datatype "CDATA">
    <!-- one or more digits -->
<!ENTITY % Script.datatype "CDATA">
    <!-- script expression -->
<!ENTITY % Text.datatype "CDATA">
    <!-- used for titles etc. -->
<!ENTITY % TimeValue.datatype "CDATA">
    <!-- a Number, possibly with its dimension, or a reserved
         word like 'indefinite' -->
<!ENTITY % URI.datatype "CDATA" >
    <!-- used for URI references -->

<!-- end of smil-datatypes-1.mod -->
```

### A.3.2 SMIL 2.0 common attributes

```
<!-- ..................................................................... -->
<!-- SMIL 2.0 Common Attributes Module  ................................... -->
<!-- file: smil-attribs-1.mod

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Common Attributes 1.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/smil-attribs-1.mod"

     ..................................................................... -->

<!-- Common Attributes

     This module declares the common attributes for the SMIL DTD Modules.
-->

<!ENTITY % SMIL.pfx "">

<!ENTITY % id.attrib
 "%SMIL.pfx;id          ID                    #IMPLIED"
>

<!ENTITY % class.attrib
 "%SMIL.pfx;class       CDATA                 #IMPLIED"
>

<!ENTITY % title.attrib
 "%SMIL.pfx;title       %Text.datatype;               #IMPLIED"
>
```

```
<!ENTITY % longdesc.attrib
 "%SMIL.pfx;longdesc     %URI.datatype;                  #IMPLIED"
>

<!ENTITY % alt.attrib
 "%SMIL.pfx;alt          %Text.datatype;                 #IMPLIED"
>

<!ENTITY % SMIL.Accessibility.attrib "
 %longdesc.attrib;
 %alt.attrib;
">

<!ENTITY % Core.extra.attrib "" >
<!ENTITY % Core.attrib "
  xml:base %URI.datatype; #IMPLIED
  %id.attrib;
  %title.attrib;
  %SMIL.Accessibility.attrib;
  %Core.extra.attrib;
">

<!ENTITY % I18n.extra.attrib "" >
<!ENTITY % I18n.attrib "
  xml:lang %LanguageCode.datatype; #IMPLIED
  %I18n.extra.attrib;"
>

<!ENTITY % SMIL.Description.attrib "
 %SMIL.pfx;abstract        %Text.datatype;   #IMPLIED
 %SMIL.pfx;author          %Text.datatype;   #IMPLIED
 %SMIL.pfx;copyright       %Text.datatype;   #IMPLIED
">

<!ENTITY % SMIL.tabindex.attrib "
 %SMIL.pfx;tabindex        %Number.datatype;   #IMPLIED
">

<!-- ================== BasicLayout ======================================= -->
<!ENTITY % SMIL.regionAttr.attrib "
 %SMIL.pfx;region         CDATA #IMPLIED
">

<!ENTITY % SMIL.fill.attrib "
 %SMIL.pfx;fill (remove|freeze|hold|transition|auto|default) 'default'
">

<!ENTITY % SMIL.fillDefault.attrib "
 %SMIL.pfx;fillDefault (remove|freeze|hold|transition|auto|inherit) 'inherit'
">

<!-- ================== HierarchicalLayout ================================ -->
<!ENTITY % SMIL.backgroundColor.attrib "
 %SMIL.pfx;backgroundColor     CDATA    #IMPLIED
">
<!ENTITY % SMIL.backgroundColor-deprecated.attrib "
 %SMIL.pfx;background-color     CDATA    #IMPLIED
">

<!ENTITY % SMIL.Sub-region.attrib "
 %SMIL.pfx;top     CDATA    'auto'
 %SMIL.pfx;bottom  CDATA    'auto'
 %SMIL.pfx;left    CDATA    'auto'
 %SMIL.pfx;right   CDATA    'auto'
 %SMIL.pfx;height  CDATA    'auto'
 %SMIL.pfx;z-index CDATA    #IMPLIED
">

<!ENTITY % SMIL.fit.attrib "
 %SMIL.pfx;fit            (hidden|fill|meet|scroll|slice)   #IMPLIED
">

<!-- ================ Registration Point attribute for media elements ============ -->
<!-- integrating language using HierarchicalLayout must include regPoint   -->
<!-- attribute on media elements for regPoint elements to be useful        -->

<!ENTITY % SMIL.regPointAttr.attrib "
 %SMIL.pfx;regPoint  CDATA    #IMPLIED
">

<!ENTITY % SMIL.regAlign.attrib "
 %SMIL.pfx;regAlign  (topLeft|topMid|topRight|midLeft|center|
                      midRight|bottomLeft|bottomMid|bottomRight) #IMPLIED
">

<!ENTITY % SMIL.RegistrationPoint.attrib "
 %SMIL.regPointAttr.attrib;
 %SMIL.regAlign.attrib;
">

<!--==================== Content Control ========================-->
<!-- customTest Attribute, do not confuse with customTest element! -->
<!ENTITY % SMIL.customTestAttr.attrib "
       %SMIL.pfx;customTest              IDREF           #IMPLIED
">

<!-- ======================== SkipContentControl Module ======================== -->
<!ENTITY % SMIL.skip-content.attrib "
       %SMIL.pfx;skip-content          (true|false)    'true'
">

<!-- Content Control Test Attributes -->

<!ENTITY % SMIL.Test.attrib "
       %SMIL.pfx;systemBitrate                 CDATA           #IMPLIED
       %SMIL.pfx;systemCaptions                (on|off)        #IMPLIED
       %SMIL.pfx;systemLanguage                CDATA           #IMPLIED
       %SMIL.pfx;systemOverdubOrSubtitle       (overdub|subtitle) #IMPLIED
       %SMIL.pfx;systemRequired                CDATA           #IMPLIED
       %SMIL.pfx;systemScreenSize              CDATA           #IMPLIED
       %SMIL.pfx;systemScreenDepth             CDATA           #IMPLIED
       %SMIL.pfx;systemAudioDesc               (on|off)        #IMPLIED
       %SMIL.pfx;systemOperatingSystem         NMTOKEN         #IMPLIED
       %SMIL.pfx;systemCPU                     NMTOKEN         #IMPLIED
       %SMIL.pfx;system-bitrate                CDATA           #IMPLIED
       %SMIL.pfx;system-captions               (on|off)        #IMPLIED
       %SMIL.pfx;system-language               CDATA           #IMPLIED
       %SMIL.pfx;system-overdub-or-caption     (overdub|caption) #IMPLIED
       %SMIL.pfx;system-required               CDATA           #IMPLIED
       %SMIL.pfx;system-screen-size            CDATA           #IMPLIED
       %SMIL.pfx;system-screen-depth           CDATA           #IMPLIED
">
```

```
<!-- SMIL Animation Module  ============================================== -->
<!ENTITY % SMIL.BasicAnimation.attrib "
  %SMIL.pfx;values      CDATA #IMPLIED
  %SMIL.pfx;from        CDATA #IMPLIED
  %SMIL.pfx;to          CDATA #IMPLIED
  %SMIL.pfx;by          CDATA #IMPLIED
">

<!-- SMIL Timing Module  ================================================ -->
<!ENTITY % SMIL.BasicInlineTiming.attrib "
  %SMIL.pfx;dur                     %TimeValue.datatype; #IMPLIED
  %SMIL.pfx;repeatCount             %TimeValue.datatype; #IMPLIED
  %SMIL.pfx;repeatDur               %TimeValue.datatype; #IMPLIED
  %SMIL.pfx;begin                   %TimeValue.datatype; #IMPLIED
  %SMIL.pfx;end                     %TimeValue.datatype; #IMPLIED
">

<!ENTITY % SMIL.MinMaxTiming.attrib "
  %SMIL.pfx;min                     %TimeValue.datatype; '0'
  %SMIL.pfx;max                     %TimeValue.datatype; 'indefinite'
">

<!ENTITY % SMIL.BasicInlineTiming-deprecated.attrib "
  %SMIL.pfx;repeat                  %TimeValue.datatype; #IMPLIED
">

<!ENTITY % SMIL.endsync.attrib "
  %SMIL.pfx;endsync             CDATA 'last'
">

<!-- endsync has a different default when applied to media elements -->
<!ENTITY % SMIL.endsync.media.attrib "
  %SMIL.pfx;endsync             CDATA 'media'
">

<!ENTITY % SMIL.TimeContainerAttributes.attrib "
  %SMIL.pfx;timeAction          CDATA #IMPLIED
  %SMIL.pfx;timeContainer       CDATA #IMPLIED
"
>

<!ENTITY % SMIL.RestartTiming.attrib "
  %SMIL.pfx;restart (always|whenNotActive|never|default) 'default'
">

<!ENTITY % SMIL.RestartDefaultTiming.attrib "
  %SMIL.pfx;restartDefault (inherit|always|never|whenNotActive) 'inherit'
">

<!ENTITY % SMIL.SyncBehavior.attrib "
  %SMIL.pfx;syncBehavior (canSlip|locked|independent|default) 'default'
  %SMIL.pfx;syncTolerance %TimeValue.datatype;               'default'
">

<!ENTITY % SMIL.SyncBehaviorDefault.attrib "
  %SMIL.pfx;syncBehaviorDefault (canSlip|locked|independent|inherit) 'inherit'
  %SMIL.pfx;syncToleranceDefault  %TimeValue.datatype;              'inherit'
">

<!ENTITY % SMIL.SyncMaster.attrib "
  %SMIL.pfx;syncMaster    (true|false)                'false'
">

<!-- ================== Time Manipulations ============================== -->
<!ENTITY % SMIL.TimeManipulations.attrib "
  %SMIL.pfx;accelerate          %Number.datatype; '0'
  %SMIL.pfx;decelerate          %Number.datatype; '0'
  %SMIL.pfx;speed               %Number.datatype; '1.0'
  %SMIL.pfx;autoReverse         (true|false)      'false'
">

<!-- ================== Media Objects ================================== -->
<!ENTITY % SMIL.MediaClip.attrib "
  %SMIL.pfx;clipBegin       CDATA   #IMPLIED
  %SMIL.pfx;clipEnd         CDATA   #IMPLIED
">
<!ENTITY % SMIL.MediaClip.attrib.deprecated "
  %SMIL.pfx;clip-begin      CDATA   #IMPLIED
  %SMIL.pfx;clip-end        CDATA   #IMPLIED
">

<!-- ================== Streaming Media ================================ -->
<!ENTITY % SMIL.Streaming-media.attrib "
  %SMIL.pfx;port                CDATA   #IMPLIED
  %SMIL.pfx;rtpformat           CDATA   #IMPLIED
  %SMIL.pfx;transport           CDATA   #IMPLIED
">

<!ENTITY % SMIL.Streaming-timecontainer.attrib "
"
>

<!-- ================== Transitions Media ============================== -->
<!ENTITY % SMIL.Transition.attrib "
 %SMIL.pfx;transIn              CDATA       #IMPLIED
 %SMIL.pfx;transOut             CDATA       #IMPLIED
">

<!-- end of smil-attribs-1.mod -->
```

### A.3.3 SMIL 2.0 qualified names module

```
<!-- ....................................................................... -->
<!-- SMIL Qualified Names Module  ........................................... -->
<!-- file: smil-qname-1.mod

     This is SMIL 2.0.

        Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
        See http://www.w3.org/Consortium/Legal/.

        Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

       PUBLIC "-//W3C//ENTITIES SMIL Qualified Names 1.0//EN"
       SYSTEM "http://www.w3.org/2001/SMIL20/smil-qname-1.mod"

     ....................................................................... -->
```

```
<!-- SMIL Qualified Names

     This module is contained in two parts, labeled Section 'A' and 'B':

       Section A declares parameter entities to support namespace-
       qualified names, namespace declarations, and name prefixing
       for SMIL and extensions.

       Section B declares parameter entities used to provide
       namespace-qualified names for all SMIL element types:

         %animation.qname; the xmlns-qualified name for <animation>
         %video.qname;     the xmlns-qualified name for <video>
         ...

     SMIL extensions would create a module similar to this one,
     using the '%smil-qname-extra.mod;' parameter entity to insert
     it within Section A.  A template module suitable for this purpose
     ('template-qname-1.mod') is included in the XHTML distribution.
-->

<!-- Section A: SMIL XML Namespace Framework :::::::::::::::::::: -->

<!-- 1. Declare the two parameter entities used to support XLink,
        first the parameter entity container for the URI used to
        identify the XLink namespace:
-->
<!ENTITY % XLINK.xmlns "http://www.w3.org/1999/xlink" >

<!-- This contains the XLink namespace declaration attribute.
-->
<!ENTITY % XLINK.xmlns.attrib
     "xmlns:xlink  %URI.datatype;          #FIXED '%XLINK.xmlns;'"
>

<!-- 2. Declare parameter entities (e.g., %SMIL.xmlns;) containing
        the namespace URI for the SMIL namespace, and any namespaces
        included by SMIL:
-->

<!ENTITY % SMIL.xmlns  "http://www.w3.org/2001/SMIL20/" >

<!-- 3. Declare parameter entities (e.g., %SMIL.prefix;) containing
        the default namespace prefix string(s) to use when prefixing
        is enabled. This may be overridden in the DTD driver or the
        internal subset of a document instance.

     NOTE: As specified in [XMLNAMES], the namespace prefix serves
     as a proxy for the URI reference, and is not in itself significant.
-->
<!ENTITY % SMIL.prefix  "" >

<!-- 4. Declare a %SMIL.prefixed; conditional section keyword, used
        to activate namespace prefixing. The default value should
        inherit '%NS.prefixed;' from the DTD driver, so that unless
        overridden, the default behaviour follows the overall DTD
        prefixing scheme.
-->
<!ENTITY % NS.prefixed "IGNORE" >
<!ENTITY % SMIL.prefixed "%NS.prefixed;" >

<!-- 5. Declare parameter entities (e.g., %SMIL.pfx;) containing the
        prefixing is active, an empty string when it is not.
-->
<![%SMIL.prefixed;[
<!ENTITY % SMIL.pfx  "%SMIL.prefix;:" >
<!ENTITY % SMIL.xmlns.extra.attrib
        "xmlns:%SMIL.prefix;    %URI.datatype;  #FIXED  '%SMIL.xmlns;'" >

]]>
<!ENTITY % SMIL.pfx  "" >
<!ENTITY % SMIL.xmlns.extra.attrib "" >


<!-- declare qualified name extensions here -->
<!ENTITY % smil-qname-extra.mod "" >
%smil-qname-extra.mod;

<!-- 6. The parameter entity %SMIL.xmlns.extra.attrib; may be
        redeclared to contain any non-SMIL namespace declaration
        attributes for namespaces embedded in SMIL. The default
        is an empty string.  XLink should be included here if used
        in the DTD and not already included by a previously-declared
        %*.xmlns.extra.attrib;.
-->

<!-- 7. The parameter entity %NS.prefixed.attrib; is defined to be
        the prefix for SMIL elements if any and whatever is in
                SMIL.xmlns.extra.attrib.
-->
<!ENTITY % XHTML.xmlns.extra.attrib "%SMIL.xmlns.extra.attrib;" >


<!-- Section B: SMIL Qualified Names :::::::::::::::::::::::::::::: -->

<!-- This section declares parameter entities used to provide
     namespace-qualified names for all SMIL element types.
-->

<!ENTITY % SMIL.animate.qname "%SMIL.pfx;animate" >
<!ENTITY % SMIL.set.qname "%SMIL.pfx;set" >
<!ENTITY % SMIL.animateMotion.qname "%SMIL.pfx;animateMotion" >
<!ENTITY % SMIL.animateColor.qname "%SMIL.pfx;animateColor" >

<!ENTITY % SMIL.switch.qname "%SMIL.pfx;switch" >
<!ENTITY % SMIL.customTest.qname "%SMIL.pfx;customTest" >
<!ENTITY % SMIL.customAttributes.qname "%SMIL.pfx;customAttributes" >
<!ENTITY % SMIL.prefetch.qname "%SMIL.pfx;prefetch" >

<!ENTITY % SMIL.layout.qname "%SMIL.pfx;layout" >
<!ENTITY % SMIL.region.qname "%SMIL.pfx;region" >
<!ENTITY % SMIL.root-layout.qname "%SMIL.pfx;root-layout" >
<!ENTITY % SMIL.topLayout.qname "%SMIL.pfx;topLayout" >
<!ENTITY % SMIL.regPoint.qname "%SMIL.pfx;regPoint" >

<!ENTITY % SMIL.a.qname "%SMIL.pfx;a" >
<!ENTITY % SMIL.area.qname "%SMIL.pfx;area" >
<!ENTITY % SMIL.anchor.qname "%SMIL.pfx;anchor" >

<!ENTITY % SMIL.ref.qname "%SMIL.pfx;ref" >
<!ENTITY % SMIL.audio.qname "%SMIL.pfx;audio" >
<!ENTITY % SMIL.img.qname "%SMIL.pfx;img" >
<!ENTITY % SMIL.video.qname "%SMIL.pfx;video" >
<!ENTITY % SMIL.text.qname "%SMIL.pfx;text" >
<!ENTITY % SMIL.textstream.qname "%SMIL.pfx;textstream" >
```

```
<!ENTITY % SMIL.animation.qname "%SMIL.pfx;animation" >
<!ENTITY % SMIL.param.qname "%SMIL.pfx;param" >
<!ENTITY % SMIL.brush.qname "%SMIL.pfx;brush" >

<!ENTITY % SMIL.meta.qname "%SMIL.pfx;meta" >
<!ENTITY % SMIL.metadata.qname "%SMIL.pfx;metadata" >

<!ENTITY % SMIL.smil.qname "%SMIL.pfx;smil" >
<!ENTITY % SMIL.head.qname "%SMIL.pfx;head" >
<!ENTITY % SMIL.body.qname "%SMIL.pfx;body" >

<!ENTITY % SMIL.seq.qname "%SMIL.pfx;seq" >
<!ENTITY % SMIL.par.qname "%SMIL.pfx;par" >
<!ENTITY % SMIL.excl.qname "%SMIL.pfx;excl" >
<!ENTITY % SMIL.priorityClass.qname   "%SMIL.pfx;priorityClass">

<!ENTITY % SMIL.transition.qname "%SMIL.pfx;transition" >
<!ENTITY % SMIL.transitionFilter.qname "%SMIL.pfx;transitionFilter" >

<!-- end of smil-qname-1.mod -->
```

### A.3.4 SMIL 2.0 framework module

```
<!-- ..................................................................... -->
<!-- SMIL 2.0 Modular Framework Module  ................................... -->
<!-- file: smil-framework-1.mod

     This is SMIL 2.0.

     Copyright 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
     See http://www.w3.org/Consortium/Legal/.

     Author:     Jacco van Ossenbruggen
     Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/smil-framework-1.mod"

     ..................................................................... -->

<!-- Modular Framework

     This required module instantiates the modules needed
     to support the SMIL 2.0 modularization model, including:

       +  datatypes
       +  namespace-qualified names
       +  common attributes
       +  document model
-->

<!ENTITY % smil-datatypes.module "INCLUDE" >
<![%smil-datatypes.module;[
<!ENTITY % smil-datatypes.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Datatypes 1.0//EN"
            "smil-datatypes-1.mod" >
%smil-datatypes.mod;]]>

<!ENTITY % smil-qname.module "INCLUDE" >
<![%smil-qname.module;[
<!ENTITY % smil-qname.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Qualified Names 1.0//EN"
            "smil-qname-1.mod" >
%smil-qname.mod;]]>

<!ENTITY % smil-attribs.module "INCLUDE" >
<![%smil-attribs.module;[
<!ENTITY % smil-attribs.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Common Attributes 1.0//EN"
            "smil-attribs-1.mod" >
%smil-attribs.mod;]]>

<!ENTITY % smil-model.module "INCLUDE" >
<![%smil-model.module;[
<!-- A content model MUST be defined by the driver file -->
%smil-model.mod;]]>
```

# Appendix B. SMIL 2.0 Schemas

**This section is informative**

A zip archive of all the SMIL 2.0 Schemas is also available

---

## B.1 XML Schema for the SMIL 2.0 functionalities:

### B.1.1 SMIL 2.0 Modules

```
<!--
XML Schema for the SMIL 2.0 modules

This is SMIL 2.0
Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
See http://www.w3.org/Consortium/Legal/.

Public URI: http://www.w3.org/2001/SMIL20/smil20.xsd
Author: Aaron Michael Cohen (Intel)
Revision: 2001/07/31
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```