

The Wayback Machine - <https://web.archive.org/web/20030803064722/http://www.w3.org:80/TR/2001/R...>



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](#)
 - [2.2.1 SMIL DOM](#)
 - [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*](#)
 - [*The target element*](#)
 - [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*](#)

[*Constraints on manipulating animation elements*](#)

[*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](#)

[4.4.1 SMIL 2.0 PrefetchControl Module Overview](#)

[*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
 - 5.6.2 Multiple Top-Level Window Support
- 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
 - 6.2.3 Relationship with XML Base
 - 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**](#)[10.1 Introduction](#)[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*](#)[*Common definitions*](#)[*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 Inline Transitions 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

13.3.12 Transition Effects Modules

13.4 Extending the SMIL 2.0 Language

13.5 Appendix A: SMIL 2.0 Document Type Definition

13.6 Appendix B: SMIL 2.0 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

14.4.4 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.3.16 SMIL 2.0 HierarchicalLayout](#)
- [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.3.49 SMIL 2.0 rdf](#)
[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 Ossensbruggen, CWI - Lloyd Rutledge, CWI - Olivier Avaro, France Telecom - Ted Wugofski, Gateway (Invited Expert) - Masayuki Hiyama, Glocomm - Keisuke Kamimura, Glocomm - 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 Fericola, 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 Hégarret, 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.

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**.

```
...
<par>
  <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 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 to dereference a runtime value for the customTest; or
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>
      
      <video src="story_1v.rm" region="b" />
      <switch>
        <audio src="story_1w.rm" region="c" customTest="west-coast"/>
        <audio src="story_1e.rm" region="c" customTest="east-coast"/>
        <audio src="story_1n.rm" region="c" customTest="far-north"/>
        <audio src="story_1r.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

values: `true` | `false`

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"/>
      
    </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:

```
<html>
<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 -->
  
</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-val)
Full-clock-val  ::= Hours ":" Minutes ":" Seconds ( "." Fraction)?
Partial-clock-val ::= Minutes ":" Seconds ( "." Fraction)?
Timecount-val   ::= Timecount ( "." Fraction)? (Metric)?
Metric          ::= "h" | "min" | "s" | "ms"
Hours           ::= DIGIT+; any positive number
Minutes         ::= 2DIGIT; range from 00 to 59
Seconds         ::= 2DIGIT; range from 00 to 59
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 appears. This attribute is introduced for future extensibility of SMIL. The functionality is unchanged from SMIL 1.0.

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

Aaron Cohen (aaron.m.cohen@intel.com), Intel
 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">
[region="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:


```

<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

Languages implementing the SMIL BasicMedia Module must define which attributes may be attached to media object elements. In all languages 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>
      
      <audio src="audio1.au"/>
    </par>
    <par>
      
      <audio src="audio2.au"/>
    </par>
    ...
    <par>
      
      <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 then be controlled with the SMIL Timing Module attributes such as `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"/>
  
</par>
```

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 hypothetical SMIL metadata RDF Schema:

```
<?xml version="1.0" ?>
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">

  <head>
    <meta id="meta-smil1.0-a" name="Publisher" content="W3C" />
    <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.smil"
        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:AccessLevelAccessibilityGuidelines="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#scene1"/>
      <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

The **id** attribute uniquely identifies an element within a document. Its value is an XML identifier.

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 XML, 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 [<spar>](#) 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.

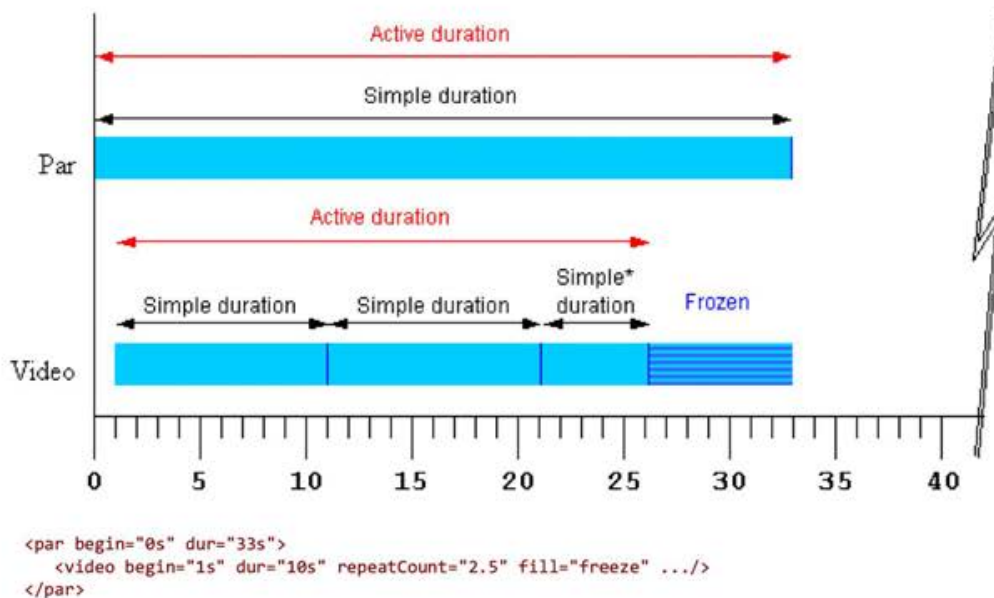


Figure 1 - Strip diagram of basic timing support. The starred "Simple*" duration indicates that the simple duration is partial (i.e. it is cut off early).

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.

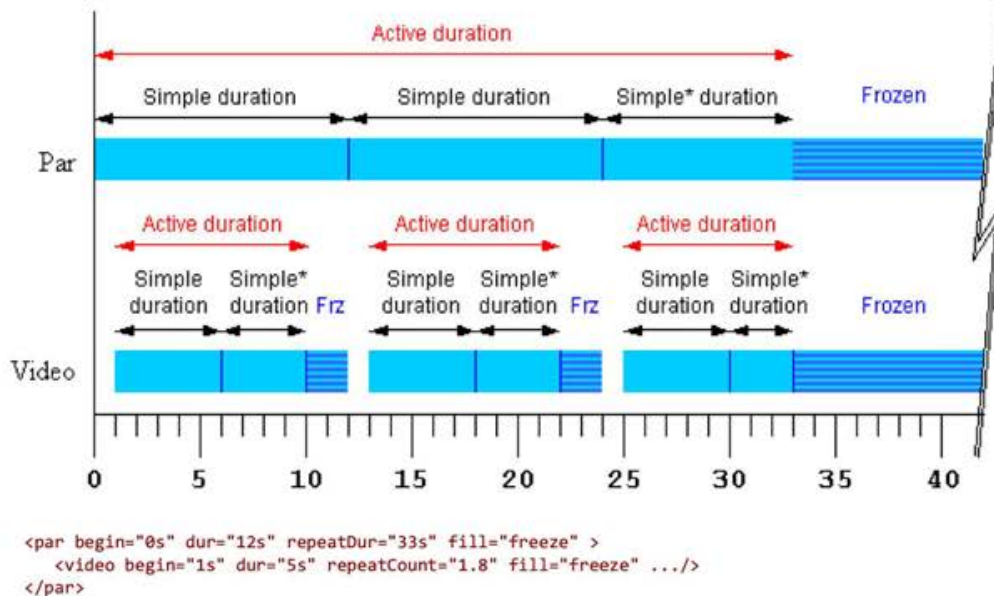


Figure 2 - Strip diagram of time container constraints upon child elements. The starred "Simple*" durations indicate that the simple duration is partial (i.e. it is cut off early).

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.

The time model depends upon several definitions for the host document: A host document is presented over a certain time interval.

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 [Reserved 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 the time values), the host language must specify how the user agent deals with this.
- 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 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 its [endsync](#) 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 synchbase begin timing. The [img](#) element begins 2 seconds after the [audio](#) element begins.

```
<par>
  <audio id="song1" src="song1.au" />
  
</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 synchbase, 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-synchbase-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 synchbase declaration, a semi-colon separated list of values.

smil-1.0-synchbase-value : "id(" Id-value ") ("(" ("begin" | "end" | Clock-value) ")") ?

Deprecated. Describes a synchbase and an offset from that synchbase. 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" ... />
  
</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 ...>
...

...
</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" .../>
</seq>
```



```

    <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>
...
</html>
```

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'):

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

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 '-'), the 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 synchbase.

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 synchbase 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-synchbase-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.
- If the clock value is omitted, it defaults to "0".
- No embedded white space is allowed between a syncbase element and a time-symbol.
- White space will be ignored before and after a "+" 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.
3. Times are expressed in local time, as defined for the presentation location. The local time zone of the end-user platform is used.

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( 2000-01-01T00:00:00.0Z )"
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 is a SMIL 1.0 identifier value of the form "id(" IDREF ")". The semantics are identical to those of the Id-value immediately above. 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 of **endsync**. 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
//   pre-visited at the current time to see if they are done.
//   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
            if( endsyncID == c.ID )
                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.

```


```

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.

```

<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 example. The sequence has an implicit simple duration of 13 seconds. It will begin to play after 5 seconds, and then will repeat the sequence of three images 3 times. The active duration is thus 39 seconds long.

```
<seq begin="5s" repeatCount="3" >
  
  
  
</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 the element (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 theseq. 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 [\[SMIL 10\]](#).

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 `</span>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.

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 is resolved (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 **syncBehavior** attribute. An element with **syncMaster** set to true will define sync for the "scope" of the time container's synchronization behavior. That is, if the **syncMaster** element's parent time container has **syncBehavior**="locked", the **syncMaster** will also define sync for the ancestor time container. 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 is the 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 $t_c(t_{ps})$ be the computed element active time for an element at the parent simple time t_{ps} , according to the defined synchronization relationship for the element.

Let $t_o(t_{ps})$ be the observed element active time for an element at the parent simple time t_{ps} .

The accumulated synchronization offset O is:

$$O = t_o(t_{ps}) - t_c(t_{ps})$$

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 element in the language. A host language must specify the *intrinsic* timing behavior of each element to which timeAction may be applied. In addition, a 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 *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.
- 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](#) semantics (see also [Attributes for timing integration: timeContainer and timeAction](#) and [Implicit duration of media element time containers](#)).

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 synchbase 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 synchbase 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 synchbase is the begin of the sequence time container.
- 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:

(synchbase-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 that the **excl** has 0 duration unless a child of the **excl** has been added to the timegraph.

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>
    <par begin="englishBtn.activateEvent" >
      <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 made active. The queue is ordered according to rules described in [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

peers = " stop | pause | defer | never "

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) element is deferred until the active element completes its active duration. The rules for adding the element to the queue are described 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 aural media.

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" .../>
  ...
</excl>
```



```

    <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>
  
  
  
</excl>

<excl>
  
  
  
</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">
    
    
    
  </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>
  
  
  
</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

This section is informative

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 last frame of video) will be shown for the remainder of the implicit duration. This only applies to visual media - aural media will simply stop playing.

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* evaluated above, the end attribute, and any repeat behavior defined by the repeatDur and repeatCount 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(zero value, anything) = zero value
- 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** The preliminary active duration of an element, before accounting for [min](#) and [max](#) semantics.
- 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, p_0 , p_1 , and p_2 , and produce an intermediate result, the *Intermediate Active Duration (IAD)* to be used in the computation above.

p_0 is the simple duration from the Simple Duration Table, given above.

If repeatCount is not specified, p_1 has the value *indefinite*. Otherwise, p_1 is the accumulated sum of the specified number of simple durations of the iterations of this element. p_1 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, p_1 will not be resolved until the specified integral number of simple durations has passed.

p_2 is the value of repeatDur. If repeatDur is unspecified, then p_2 will have a value of *indefinite*.

Then *IAD* is given by:

If p_0 equals 0, then

$$IAD = 0$$

Else if repeatDur and repeatCount are unspecified then:

$$IAD = p_0$$

else:

$$IAD = \text{MIN}(p_1, p_2, \text{indefinite})$$

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">
  
  <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>
  
  
  
  <!-- 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.

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 timograph) 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">
  
  
  
</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 $\geq$ 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 $\geq$ 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 $\geq$ 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 $\geq$ 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 >= 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 == 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 synbase 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" .../>
```

Example 3: In the following example, the 2 images define begin times that are mutually dependent, but the first has multiple begin conditions that allow 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".

When repeating an element's simple duration, the arithmetic follows the end-point exclusive model. Consider the example:

```
<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 $t_{last-repeat}$. Some other mechanism (such as endsync logic or a media player) must note when the simple duration ends, and reset the value of $t_{last-repeat}$. If the element has not yet repeated, a value of 0 is used in place of $t_{last-repeat}$.

$$t_s = t_a - t_{last-repeat}$$

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) $t_{last-repeat}$ can be obtained directly for the simple duration **dur** and so the expression can be reduced to:

$$t_s = \text{REMAINDER}(t_a, \text{dur})$$

where **REMAINDER**(*t*, *d*) is defined as $(t - d \cdot \text{floor}(t/d))$.

Converting wall-clock values

When the document begins, the current wall-clock time is noted and saved as $t_{wallclock-begin}$. 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):

$$t_{ra} = t_{wc} - t_{wallclock-begin}$$

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.

To convert from element simple time to element active time requires the begin time of the most recent iteration, $t_{last-repeat}$. If the element does not repeat or has not yet repeated, a value of 0 is used in place of $t_{last-repeat}$.

$$t_a = t_s + t_{last-repeat}$$

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 sought-to time. The elements that are begun in this manner may still be active, may be frozen, or may already have ended at the sought-to time.
- If an element begins and ends within the seek interval (between the current time before the hyperlink traversal and the sought-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 sought-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 sought over during hyperlink traversal (the active duration of the hyperlink begins and ends during the seek interval). However, automatic hyperlinks that are only partially sought 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 sought-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 sought 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 is performed. 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 sought 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 synbase 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, 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 synbase 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 usevent), 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 the element), 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 synbase 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 synbase 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="8s" .../>
  <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 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 0 and 5 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.

10.4.1 Required host language definitions

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 *documentend*. 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

This section is normative

- 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 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 *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">
  
  
  
</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

This module defines the attribute value syntax for the begin and end attributes that allow elements to begin and end based upon markers 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>
  
  
  
</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>
  
  
  
</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>
  <excl>
    <par id="p1">
      ...
    </par>
    <par id="p2">
      ...
    </par>
  </excl>
  <a href="p1"></a>
  <a href="p2"></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>
  
  <video src="vid2.mpg" />
  <video src="vid3.mpg" />
</seq>
```

The implicit synchbase 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">
  
  
  
</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](#).