Synchronized Multimedia Integration Language (SMIL) Boston Specification

W3C Working Draft 25 February 2000

This version:
http://www.w3.org/TR/2000/WD-smil-boston-20000225
(Other formats: single PostScript file, single PDF file, zip archive)
Latest version:
http://www.w3.org/TR/smil-boston
Previous version:
http://www.w3.org/TR/1999/WD-smil-boston-19991115
Editors:
Jeff Ayars (RealNetworks), Dick Bulterman (Oratrix), Aaron Cohen (Intel), Erik Hodge (RealNetworks), Philipp Hoschka (W3C), Eric Hyche (RealNetworks), Ken Day (Macromedia), Kenichi Kubota (Panasonic), Rob Lanphier (RealNetworks), Nabil Layaïda (INRIA), Philippe Le Hégaret (W3C), Thierry Michel (W3C), Jacco van Ossenbruggen (CWI), Lloyd Rutledge (CWI), Bridie Saccocio (RealNetworks), Patrick Schmitz (Microsoft), Warner ten Kate (Philips), Ted Wugofski (Gateway).

Abstract

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

- Define a simple XML-based language that allows authors to write interactive multimedia presentations. Using SMIL Boston, 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 Boston components should be used for integrating timing into XHTML [XHTML10].

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 document is the third Working Draft of the specification for the next version of SMIL code-named "Boston". It has been produced as part of the W3C Synchronized Multimedia Activity. The document has been written by the SYMM Working Group *(members only)*. The goals of this group are discussed in the SYMM Working Group charter *(members only)*.

Many parts of the document are still preliminary, and do not constitute full consensus within the Working Group. Also, some of the functionality planned for SMIL Boston is not contained in this draft. Many parts are not yet detailed enough for implementation, and other parts are only suitable for highly experimental implementation work.

At this point, the W3C SYMM WG seeks input by the public on the concepts and directions described in this specification. Please send your comments to www-smil@w3.org. Since it is difficult to anticipate the number of comments that come in, the WG cannot guarantee an individual response to all comments. However, we will study each comment carefully, and try to be as responsive as time permits.

This working draft may be updated, replaced or rendered obsolete by other W3C documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This document is work in progress and does not imply endorsement by the W3C membership.

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

Quick Table of Contents

Full Table of Contents

1. About SMIL Boston

Editors

Philipp Hoschka (ph@w3.org), W3C
Aaron Cohen (aaron.m.cohen@intel.com), Intel

1.1 Introduction

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

- Define a simple XML-based language that allows authors to write interactive multimedia presentations. Using SMIL Boston, 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 Boston components should be used for integrating timing into XHTML.

SMIL Boston is defined as a set of markup modules, which define the semantics and an XML syntax for certain areas of SMIL functionality. All modules have an associated Document Object Model (DOM).

SMIL Boston 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 Boston 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 (or however we denote SMIL documents vs. integration documents) must support the deprecated SMIL 1.0 attribute names as well as the new SMIL Boston names.

This specification is structured as a set of sections, defining module:

- Section 2 presents an overview of the individual modules, and gives example profiles.
- Section 3 defines the declarative animation module.
- Section 4 presents the content control module, such as the switch and preload elements.
- Section 5 describes the SMIL Boston basic layout module.
- Section 6 defines the linking module.
- Section 7 presents the media object module.
- Section 8 defines the metadata module.
- Section 9 defines the SMIL Boston structure module including the head, and

body elements.
- Section 10 defines the SMIL timing and synchronization module.
- Section 11 describes the means of integrating SMIL timing into other XML-based languages.
- Section 12 presents the transition effects module.
- Section 13 defines the SMIL DOM interfaces for all of the above modules.

This specification also defines three profiles that are built using the above SMIL modules:

- Section 14 defines the SMIL Boston Language Profile.
- Section 15 defines the HTML + SMIL Language Profile.
- Section 16 describes the SMIL Basic Language Profile requirements.

Finally, this specification defines a number of baseline media formats to be widely supported by SMIL players:

- Section 17 presents a list of baseline media formats.

1.2 Acknowledgements

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

- Jin Yu, Compaq
- Pietro Marchisio, CSELT
- Lynda Hardman, CWI
- Jacco van Ossenbruggen, CWI
- Lloyd Rutledge, CWI
- Ted Wugofski, Gateway (Invited Expert)
- Masayuki Hiyama, Glocomm
- Keisuke Kamimura, Glocomm
- Michelle Y. Kim, IBM
- Steve Wood, IBM
- Nabil Layaïda, INRIA
- Muriel Jourdan, INRIA
- Aaron Cohen, Intel
- Wayne Carr, Intel
- Ken Day, Macromedia
- Daniel Weber, Matsushita
- Patrick Schmitz, Microsoft
- Debbie Newman, Microsoft
- Pablo Fernicola, Microsoft
- Wo Chang, NIST

- Didier Chanut, 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
- Geoff Freed, WGBH
- Philipp Hoschka, W3C
- Philippe Le Hégaret, W3C
- Thierry Michel, W3C.

2. Synchronized Multimedia Integration Language (SMIL) Modules

Editors:
Warner ten Kate <warner.ten.kate@philips.com>,
Ted Wugofski <wugofted@gateway.com>,
Patrick Schmitz <pschmitz@microsoft.com>.

2.1 Introduction

Since the publication of SMIL 1.0 [SMIL10], interest in the integration of SMIL concepts with the HTML, the Hypertext Markup Language [HTML40], and other XML languages, has grown. Likewise, the W3C HTML Working Group is specifying how XHTML, the Extensible Hypertext Markup Language [XHTML10], can be integrated with other languages. The strategy considered for integrating respective functionality with other XML languages is based on the concepts of *modularization* and *profiling* [MODMOD], [SMIL-MOD], [XMOD], [XPROF].

Modularization is a solution in which a language's functionality is partitioned into sets of semantically-related elements. *Profiling* is the combination of these feature sets to solve a particular problem. For the purposes of this specification we define:

element
: An element is a representation of a semantic feature. An element has one representation in any given syntax.

module
: A module is a collection of semantically-related elements.

module family
: A module family is a collection of semantically-related modules. Each element is in one and only one module family. Modules in a module family are generally ordered by increasing functionality (each module is generally inclusive of the previous module in the module family).

profile
: A profile is a collection of modules particular to an application domain or language. For example, the SMIL profile corresponds to the collection of modules that make up the SMIL language. Likewise, an enhanced television profile would correspond to the collection of modules for media-enhancement of broadcast television. In general, a profile would include only one module from a particular module family.

profile family
: A profile family is a collection of profiles which all share a common set of modules. Those modules are defined as mandatory to a profile which wishes to be part of that profile family. Examples are the XHTML family and the SMIL family.

SMIL functionality is partitioned into modules based on the following design requirements:

1. Ensure that a profile may be defined that is completely backward compatibility with SMIL 1.0.
2. Ensure that a module's semantics maintain compatibility with SMIL semantics (this includes content and timing).
3. Specify modules that are isomorphic with other modules based on W3C recommendations.
4. Specify modules that can complement XHTML modules.
5. Adopt new W3C recommendations when appropriate and not in conflict with other requirements.
6. Specify how the modules support the document object model.

The first requirement is that modules are specified such that a collection of modules can be "recombined" in such a way as to be backward compatible with SMIL (it will properly play SMIL conforming content).

The second requirement is that the semantics of SMIL must not change when they are embodied in a module. Fundamentally, this ensures the integrity of the SMIL content and timing models. This is particularly relevant when a different syntax is required to integrate SMIL functionality with other languages.

The third requirement is that modules be isomorphic with other modules from other W3C recommendations. This will assist designers when sharing modules across profiles.

The fourth requirement is that specific attention be paid to providing multimedia functionality to the XHTML language. XHTML is the reformulation of HTML in XML.

The fifth requirement is that the modules should adopt new W3C recommendations when they are appropriate and when they do not conflict with other requirements (such as complementing the XHTML language).

The sixth requirement is to ensure that modules have integrated support for the document object model. This facilitates additional control through scripting and user agents.

These requirements, and the ongoing work by the SYMM Working Group, led to a partitioning of SMIL functionality into nine modules.

2.2 SMIL Modules

SMIL functionality is partitioned into nine (9) modules :

1. Animation Module
2. Content Control Module
3. Layout Module
4. Linking Module
5. Media Object Module

6. Metainformation Module
7. Structure Module
8. Timing and Synchronization Module
9. Transition Effects Module

Each of these modules introduces a set of semantically-related elements, properties, and attributes.

Further, there are the DOM modules [DOM1], [DOM2], [SMIL-DOM]. A profile may include DOM support. The part of DOM being supported, corresponds to the modules being selected in the profile.

2.2.1 Animation Module

The Animation Module provides a framework for incorporating animation onto a timeline (a timing model) and a mechanism for composing the effects of multiple animations (a composition model). The Animation Module defines semantics for the animate, set, animateMotion, and animateColor elements:

Elements	Attributes	Minimal Content Model
animate	TBD	TBD
set	TBD	TBD
animateMotion	TBD	TBD
animateColor	TBD	TBD

When this module is used, it adds the animate, set, animateMotion, and animateColor elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the content model of the body element of the Structure Module.

2.2.2 Content Control Module

The Content Control Module provides a framework for selecting content based on a set of test attributes. The Content Control Module defines semantics for the switch element.

Elements	Attributes	Minimal Content Model
switch	TBD	TBD
-	test attributes	N/A

When this module is used, it adds the switch, element to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds this element to the content model of the body element of the Structure Module. It also adds this element to the content model of the a element of the Linking Module. It also adds this element to the content model of the head element of the Structure Module.

Further, when this module is used, the test attributes are added to the attribute lists of all the elements in the Layout Module , the Media Object Module , the Timing and Synchronization Module , and the Transition Effect Module .

Effectuation applies only when the mentioned Modules are part of the profile at hand, of course.

2.2.3 Layout Module

The Layout Module provides a framework for spatial layout of visual components. The Layout Module defines semantics for the layout, root-layout, and region elements.

Elements	Attributes	Minimal Content Model
layout	TBD	TBD
root-layout	TBD	TBD
region	TBD	TBD

When this module is used, it adds the layout element to the content model of the head element of the Structure Module. It also adds this element to the content model of the switch element of the Content Control Module.

2.2.4 Linking Module

The Linking Module provides a framework for relating documents to content, documents and document fragments. The Linking Module defines semantics for the a and area elements.

Elements	Attributes	Minimal Content Model
a	TBD	TBD
area	TBD	TBD

When this module is used, it adds the area and a elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the content model of the body element of the Structure Module.

2.2.5 Media Object Module

The Media Object Module provides a framework for declaring media. The Media Object Module defines semantics for the ref, animation, audio, img, video, text, and textstream elements.

Elements	Attributes	Minimal Content Model
ref	TBD	TBD
img, text	TBD	TBD
audio, video, animation, textstream	TBD	TBD

When this module is used, it adds the ref, animation, audio, img, video, text, and textstream elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the content model of the body element of the Structure Module. It also adds these elements to the content model of the a element of the Linking Module.

2.2.6 Metainformation Module

The Metainformation Module provides a framework for describing a document, either to inform the human user or to assist in automation. The Metainformation Module defines semantics for the meta element.

Elements	Attributes	Minimal Content Model
meta	TBD	TBD

When this module is used, it adds the meta element to the content model of the head element of the Structure Module.

2.2.7 Structure Module

The Structure Module provides a framework for structuring a SMIL document. The Structure Module defines semantics for the smil, head, and body elements.

Elements	Attributes	Minimal Content Model
smil	Core, Accessibility, xmlns	head?, body?
head	Core, Accessibility, profile	meta*, (switch | layout)?
body	Core, Accessibility	(Schedule | MediaContent | MediaControl | LinkAnchor)*
-	skipContent	N/A

This module is a mandatory part in any profile family labeled "SMIL".

When this module is used the id, title, and skipContent attributes are added to all other modules used, including modules from other, non-SMIL, origine.

2.2.8 Timing and Synchronization Module

The Timing and Synchronization Module provides a framework for describing timing structure, timing control properties, and temporal relationships between elements. The Timing and Synchronization Module defines semantics for par, seq, and excl elements. In addition, this module defines semantics for attributes including begin, dur, end, repeatCount, repeatDur, and others.

@@ Make "and others" explicit.

@@ These enumerations need check on completeness and correctness.

Elements	Attributes	Minimal Content Model
par, seq, excl	TBD	TBD
	begin, end, dur, repeatCount, repeatDur, TBD	TBD

This module is mandatory in any profile incorporating SMIL modules. By that, it is a mandatory module in any profile in the SMIL family. Note that upon building a profile which integrates SMIL timing with other, non-SMIL, modules, that the elements from this Timing and Synchronization module may appear as attributes to the elements from the other XML language, rather than as these elements themselves.

The timing attributes are used by all the elements in the Media Object Module , the Linking Module , the Content Control Module , and the Timing and Synchronization Module . Effectuation applies only when those Modules are part of the profile, of course. As upon integration with non-SMIL modules, the elements from this module may appear as attributes instead of elements, the referenced timing attributes are also used by those non-SMIL elements.

2.2.9 Transition Effects Module

The Transition Effects Module defines a taxonomy of transition effects as well as semantics and syntax for integrating these effects into XML documents

Elements	Attributes	Minimal Content Model
TBD	TBD	TBD

When this module is used, it adds the TBD element to the content model of the layout element of the Layout Module.

2.3 Isomorphism

A requirement for SMIL modularization is that the modules be isomorphic with other modules from other W3C recommendations. Isomorphism will assist designers when sharing modules across profiles. The Table below lists the isomorphism between SMIL and XHTML modules.

Table -- Isomorphism between SMIL modules and their corresponding XHTML modules.

SMIL modules		XHTML modules	
Module Name	**Elements**	**Module Name**	**Elements**
Animation	animate, set, animateMotion, animationColor	-	-
Content Control	switch	-	-
Layout	layout, region, root-layout	Stylesheet	style
Linking	a, area	Hypertext	a
		Client-side Image Map	map, area
Media Object	ref, audio, video, text, img, animation, textstream	Object	object, param
		Image	img
		Applet	applet, param
Metainformation	meta	Metainformation	meta
		Link	link
		Base	base
Structure	smil, head, body	Structure	html, head, body, title, span, div
Timing and Synchronization	par, seq, excl	-	-
Transition Effects	transition	-	-

As can be seen in the table, the Metainformation module appears in both SMIL and HTML. Work is underway to define a single module that can be shared by both SMIL and HTML. In SMIL Boston the Linking Module has been adapted towards isomorphism with the corresponding modules in XHTML.

2.4 Multimedia Profiles

There are a range of possible profiles that may be built using SMIL modules. Four profiles are defined to inform the reader of how profiles may be constructed to solve particular problems:

- Lightweight Presentations Profile
- SMIL-Boston Profile
- SMIL-Basic Profile
- HTML+SMIL Profile
- Web Enhanced Media Profile

These example profiles are non-normative.

2.4.1 Lightweight Presentations Profile

The Lightweight Presentations Profile handles simple presentations, supporting timing of text content. The simplest version of this could be used to sequence stock quotes or headlines on constrained devices such as a palmtop device or a smart phone. This example profile might include the following SMIL modules:

- Timing and Synchronization Module
- Transition Effects Module
- Animation Module

This profile may be based on XHTML modules [XMOD] with the addition of Timing and Synchronization Module.

2.4.2 SMIL-Boston Profile

The SMIL-Boston Profile supports the timeline-centric multimedia features found in language of the SMIL family. This profile is specified in the SMIL Boston Profile and includes the following SMIL modules:

- Structure Module
- Metainformation Module
- Timing and Synchronization Module
- Transition Effects Module
- Animation Module
- Content Control Module
- Media Object Module
- Layout Module
- Linking Module

2.4.3 SMIL-Basic Profile

The SMIL-Basic Profile supports a leightweight version of the SMIL-Boston profile and is intended for use with resource-constrained devices such as mobile phones. This profile is part of the SMIL family and might include the following SMIL modules:

@@ Keep aligned with the requirements document.

- Structure Module
- Timing and Synchronization Module
- Layout Module
- Media Object Module
- Linking Module

2.4.4 HTML+SMIL Profile

The HTML+SMIL Profile integrates SMIL timing into HTML. This profile is specified in the HTML+SMIL Profile and includes the following SMIL modules:

- Timing and Synchronization Module
- Transition Effects Module
- Animation Module
- Content Control Module
- Media Object Module
- Linking Module

This profile uses XHTML modules for structure and layout and SMIL modules for multimedia and timing. Since the Linking modules from the XHTML modules [XMOD] and the SMIL modules are isomorphic, basically the Linking Module may come from either module set. However, the SMIL Linking Module adds some additional attributes and semantics.

@@ Aren't these attributes and semantics already added through the Timing & Synchronization Module?

2.4.5 Web Enhanced Media Profile

The Web Enhanced Media Profile supports the integration of multimedia presentations with broadcast or on-demand streaming media. The primary media will often define the main timeline. This profile might include the following SMIL modules:

- Timing and Synchronization Module
- Transition Effects Module
- Media Object Module
- Linking Module

This profile is a lightweight version of the HTML+SMIL Profile in that it supports a smaller subset of functionality taken from the XHTML and SMIL modules. It differs from the SMIL-Basic Profile through its integration with XHTML.

3. The SMIL Animation Module

Editors
Patrick Schmitz (pschmitz@microsoft.com), (Microsoft)
Aaron Cohen (aaron.m.cohen@intel.com), (Intel)
Ken Day (kday@macromedia.com), (Macromedia)

3.1 Introduction

@@ "SMIL Boston" is used here for clarity -- need to distinguish SMIL 1.0, the (standalone) SMIL Animation module now in "last call", and this module. This will be corrected prior to going to Last Call.

This section defines the SMIL Boston Animation module. SMIL animation is a framework for incorporating animation onto a time line and a mechanism for composing the effects of multiple animations. It includes a set of basic animation elements that can be applied to any XML-based language. Since these elements and attributes are defined in a module, designers of other markup languages can reuse the functionality in the SMIL animation module when they need to include animation in their language.

This module is built upon the functionality of the first version of the SMIL Animation [SMIL-ANIMATION] module, currently in last call. The timing model included in the first version is in turn based upon SMIL 1.0 [SMIL10], with some changes and extensions to support interactive (event-based) timing. The extensions in that version of Antimation are compatible with a core subset of the functionality expected to be included in the SMIL Timing module.

This two-version approach has been used in order to facilitate release of a first version of SMIL Animation well before SMIL will be ready to go to Recommendation status.

In this version, the SMIL animation module has been reworked to directly use the SMIL timing module. It does not redefine timing markup specifically for the purpose of animation. It has also been extended to include time containers like `<par>` and `<seq>,` which were not supported in the first version.

The reader is presumed to have read and be familiar with the SMIL Timing module, on which this module depends.

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

The examples in this document that include syntax for a host language use SMIL, SVG, XHTML and CSS. These are provided as an indication of possible integrations with various host languages. @@May be changed to SMIL-only examples prior to going to Recommendation.

Unresolved intra-SMIL references

@@@ Refs to other SMIL modules, to be fixed:

[wd-timing-repeatAttrs]
: Definition of repeatCount & repeatDur attrs.

[wd-timing-TimingAndRealWorldClockTime]
: This was a section in the "standalone" draft. Is there a counterpart in the Timing module?

[wd-timing-Restart]
: Definition of restart

[wd-timing-TimingAttrsEntity]
: DTD for timing attributes

[wd-timing-PropagatingTimes]
: This was a section in the "standalone" draft. Is there a counterpart in the Timing module?

[wd-some-IDAttribute]
: Definition of the ID attribute.

3.2 Overview and terminology

3.2.1 Basics of animation

Animation is inherently time-based. SMIL animation is defined in terms of the SMIL timing model. The animation capabilities are described by new elements with associated attributes and semantics, as well as the SMIL timing attributes. Animation is modeled as a function that changes the *presented value* of a specific attribute over time.

Animation is defined as a time-based manipulation of a *target element* (or more specifically of some *attribute* of the target element, the *target attribute*). The animation defines a mapping of time to values for the target attribute. This mapping takes into account all aspects of timing, as well as animation-specific semantics. It is based on an *animation function* that produces a value for the target attribute for any time within the simple duration.

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

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

The animation function is evaluated as needed over time by the implementation, and the resulting values are applied to the presentation value for the target attribute. Animation functions are continuous in time and can be sampled at whatever frame rate is appropriate for the rendering system. The syntactic representation of the animation function is independent of this model, and may be described in a variety of ways. The animation elements in this specification support syntax for a set of discrete or interpolated values, a path syntax for motion based upon SVG paths, key-frame based timing, evenly paced interpolation, and variants on these features. Animation functions could be defined that were purely or partially algorithmic (e.g. a random value function or a motion animation that tracks the mouse position) . In all cases, the animation exposes this as a function of time.

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

Animations can be defined to either override or add to the base value of an attribute. In this context, the base value may be the DOM value, or the result of other animations that also target the same attribute. This more general concept of a base value is termed the *underlying value.*Animations that add to the underlying value are described as *additive* animations. Animations that override the underlying value are referred to as *non-additive* animations.

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

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

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

3.2.2 Animation function values

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

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

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

3.2.3 Symbols used in the semantic descriptions

`f(t)`
: The simple animation function that maps times within the simple duration to values for the target attribute (0 <= t <= simple duration). Note that while **`F(t)`** defines the mapping for the entire animation, **`f(t)`** has a simplified model that just handles the simple duration.

`F(t)`
: The effect of an animation for any point in the animation. This maps any non-negative time to a value for the target attribute. A time value of 0 corresponds to the time at which the animation begins. Note that **`F(t)`** combines the animation function **`f(t)`** with all the other aspects of animation and timing controls.

`B`
: The begin of an animation.

`d`
: The simple duration of an animation.

`AD`
: The active duration of an animation. This is the period during which time is actively advancing for the animation. This includes any effect of repeating the simple duration, but does not include the time during which the animation may be frozen.

`AE`
: The active end. This is the end of the active duration of an animation.

3.3 Animation model

This section describes the attribute syntax and semantics for describing animations. The specific elements are not described here, but rather the common concepts and syntax that comprise the model for animation. Document issues are described, as well as the means to target an element for animation. The animation model is then defined by building up from the simplest to the most complex concepts: first the simple duration and animation function **`f(t)`**, and then the overall behavior **`F(t)`**. Finally, the model for combining animations is presented, and additional details of implications of the timing model on animation are described.

3.3.1 Specifying the animation target

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

The Target attribute

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

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

The `attributeType` attribute is optional. By default, the animation runtime will resolve the names according to the following rule: If there is a name conflict and `attributeType` is not specified, the CSS namespace is matched first (if CSS is supported in the host language), followed by the default namespace for the target element.

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

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

attributeName = **<attributeName>**
: Specifies the name of the target attribute. An XMLNS prefix may be used to indicate the XML namespace for the attribute. The prefix will be interpreted in the scope of the target element.

attributeType = "**CSS" | "XML" | "auto**"
: Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following (values are case-sensitive):
  "CSS"
  : This specifies that the value of "attributeName" is the name of a CSS property, as defined for the host document. This argument value is only meaningful in host language environments that support CSS.

  "XML"
  : This specifies that the value of "attributeName" is the name of an XML attribute defined in the default XML namespace for the target element. If the value for `attributeName` has an XMLNS prefix, the implementation must

use the associated namespace as defined in the scope of the target element.

"auto"

The implementation should match the attributeName to an attribute for the target element. The implementation must first search through the CSS namespace for a matching property name, and if none is found, search the XML namespace.

This is the default.

The Target element

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

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

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

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

`targetElement` = "**<IDREF>**"

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

`href` = *uri-reference*

This attribute specifies an XLink locator, referring to the target element to be animated.

When integrating animation elements into the host language, the language designer should avoid including both of these attributes. If however, both attributes must be included in the host language, and they both occur in an animation element, the XLink "href" attribute takes precedence over the "targetElement" attribute.

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

This example uses the simpler `targetElement` syntax:

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

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

```
<animate href="#foo" attribute="bar" .../>
```

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

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

`type` = 'simple'
: Identifies the type of XLink being used. To link to the target element, a simple link is used, and thus the attribute value is fixed to "simple".

`actuate` = 'onLoad'
: Indicates that the link to the target element is followed automatically (i.e., without user action).
: @@ This may be in conflict with the Linking module. OTOH, for our purposes it means basically the same thing. Need to be consistent, of course.

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

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

3.3.2 Specifying the animation function f(t)

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

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

from = "**<value>**"
: Specifies the starting value of the animation.

to = "**<value>**"
: Specifies the ending value of the animation.

by = "**<value>**"
: Specifies a relative offset value for the animation.

values = "**<list>**"
: A semicolon-separated list of one or more values. Vector-valued attributes are supported using the vector syntax of the `attributeType` domain.

The animation values specified in the animation element must be legal values for the specified attribute. See also Animation function value details .

Leading and trailing white space, and white space before and after semi-colon separators, will be ignored.

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

If a list of values is used, the animation will apply the values in order over the course of the animation (pacing and interpolation between these values is described in "Animation function calculation modes ", below. If a list of *values* is specified, any *from*, *to* and *by* attribute values are ignored.

The simpler *from/to/by* syntax provides for several variants. Note that `from` is optional, but that one of `by` or `to` must be used (unless of course a list of `values` is provided). It is not legal to specify both `by` and `to` attributes - if both are specified, only the `to` attribute will be used (the `by` will be ignored). The combinations of attributes yield the following classes of animation:

from-to animation
: Specifying a `from` value and a `to` value defines a simple animation, equivalent to a `values` list with 2 values. The animation function is defined to start with the `from` value, and to finish with the `to` value.

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

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

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

The last two forms "*by animation*" and "*to animation*" have additional semantic constraints when combined with other animations. The details of this are described below in the section How from, to and by attributes affect additive behavior .

Interpolation and indefinite simple durations

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

Examples

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

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

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

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

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

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

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

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

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

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

Animation function calculation modes

By default, a simple linear interpolation is performed over the values, evenly spaced over the duration of the animation. Additional attributes can be used for finer control over the interpolation and timing of the values. The `calcMode` attribute defines the basic method of applying values to the attribute. The `keyTimes` attribute provides additional control over the timing of the animation function, associating a time with each value in the `values` list. Finally, the `keySplines` attribute provides a means of controlling the pacing of interpolation *between* the values in the `values` list.

`calcMode` = "**discrete" | "linear" | "paced" | "spline**"

Specifies the interpolation mode for the animation. This can take any of the following values. The default mode is "linear", however if the attribute does not support linear interpolation (e.g. for strings), the `calcMode` attribute is ignored and discrete interpolation is always used.

"discrete"

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

"linear"

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

"paced"

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

"spline"

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

`keyTimes` = "**<list>**"

A semicolon-separated list of time values used to control the pacing of the animation. Each time in the list corresponds to a value in the `values` attribute list, and defines when the value should be used in the animation function. Each time value in the `keyTimes` list is specified as a floating point value between 0 and 1 (inclusive), representing a proportional offset into the simple duration of the animation element.
If a list of `keyTimes` is specified, there must be exactly as many values in the `keyTimes` list as in the `values` list.
Each successive time value must be greater than or equal to the preceding time value.
The `keyTimes` list semantics depends upon the interpolation mode:

- For linear and spline animation, the first time value in the list must be 0, and the last time value in the list must be 1. The keyTime associated with each value defines when the value is set; values are interpolated between the keyTimes.

- For discrete animation, the first time value in the list must be 0. The keyTime associated with each value defines when the value is set; the animation function uses each value until the next keyTime defined.

If there are any errors in the `keyTimes` specification (bad values, too many or too few values), the animation will have no effect
If the simple duration is indefinite, any <code>keyTimes</code> specification will be ignored.

`keySplines` = "**<list>**"
A set of Bezier control points associated with the `keyTimes` list, defining a cubic Bezier function that controls interval pacing. The attribute value is a semi-colon separated list of control point descriptions. Each control point description is a set of four floating point values: `x1 y1 x2 y2`, describing the Bezier control points for one time segment. The `keyTimes` values that define the associated segment are the Bezier "anchor points", and the `keySplines` values are the control points.
Thus, there must be one fewer sets of control points than there are `keyTimes`.
The values must all be in the range 0 to 1.
This attribute is ignored unless the `calcMode` is set to "spline".
If there are any errors in the `keySplines` specification (bad values, too many or too few values), the animation will have no effect.

If the `keyTimes` attribute is not specified, the values in the `values` attribute are assumed to be equally spaced through the animation duration, according to the `calcMode`:

- For discrete animation, the duration is divided into equal time periods, one per value. The animation function takes on the values in order, one value for each time period.
- For linear and spline animation, the duration is divided into `n-1` even periods, and the animation function is a linear interpolation between the values at the associated times. Note that a linear animation will be a nicely closed loop if the first value is repeated as the last.

Note that for the shorthand forms *to animation* and *from-to animation*, there are only 1 and 2 values respectively. Thus a discrete *to animation* will simply set the "to" value for the simple duration. A discrete *from-to animation* will set the "from" value for the first half of the simple duration and the "to" value for the second half of the simple duration.

Note that if the `calcMode` is set to "paced", the `keyTimes` attribute is ignored, and the values in the `values` attribute are spaced to produce a constant rate of change as the target attribute value is interpolated.

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

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

Examples

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

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

This example illustrates the use of keyTimes:

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

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

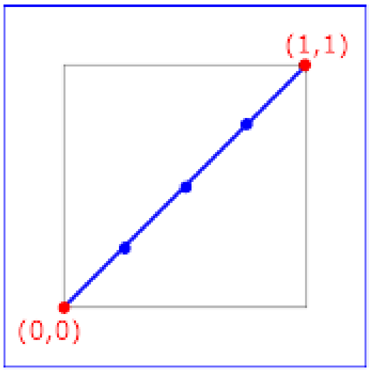
Extending this example to use keySplines:

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

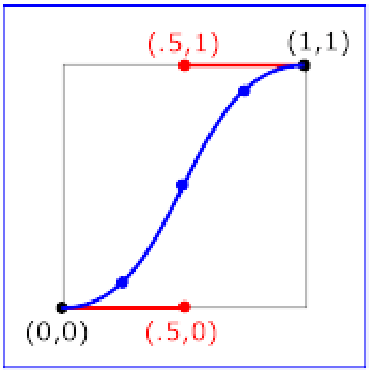
The keyTimes still causes the "x" attribute to have a value of "0" at the start of the animation, "50" after 8 seconds and "100" at the end of the animation. However, the keySplines values define a curve for pacing the interpolation between values. In the example above, the spline causes an ease-in and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

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

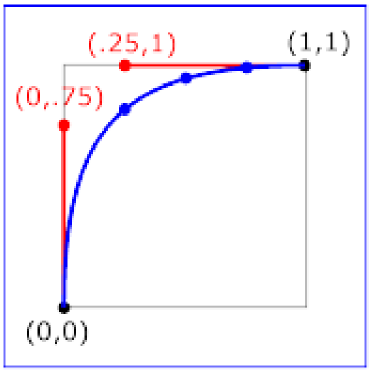
The following figure illustrates the interpretation of the `keySplines` attribute. Each diagram illustrates the effect of `keySplines` settings for a single interval (i.e. between the associated pairs of values in the `keyTimes` and `values` lists.). The horizontal axis can be thought of as the input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



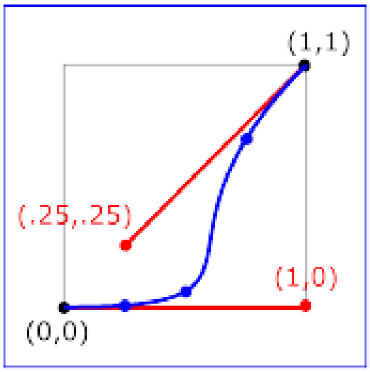
keySplines="0 0 1 1" (the default)



keySplines=".5 0 .5 1"



keySplines="0 .75 .25 1"



keySplines="1 0 .25 .25"

Figure - Illustration of keySplines effect.

To illustrate the calculations, consider the simple example:

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

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

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

For a formal definition of Bezier spline calculation, see *[Foley] pp. 488-491.*

3.3.3 Specifying the animation effect F(t)

As described above, the animation function **`f(t)`** defines the animation for the simple duration. However SMIL Animation allows the author to repeat this, and to specify whether the animation should simply end when the active duration completes, or whether it should be *frozen* at the last value. In addition, the author can specify how each animation should be combined with other animations and the underlying DOM value.

This section describes the syntax and associated semantics for the additional functionality. A detailed model for combining animations is described, along with additional details of implications of the timing model.

Repeated animations

Repeating an animation causes the animation function **`f(t)`** to be "played" several times in sequence. The author can specify either *how many times* to repeat, using the timing attribute `repeatCount`, or *how long*to repeat, using the timing attribute `repeatDur`. Each repeat *iteration* is one instance of "playing" the animation function **`f(t)`**. If the simple duration **`d`** is indefinite, the animation cannot repeat.

The `repeatCount` and `repeatDur` attributes are described in detail in [wd-timing-repeatAttrs].

Examples

In the following example, the 2.5 second animation function will be repeated twice; the active duration **`AD`** will be 5 seconds.

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

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

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

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

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

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

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

Controlling behavior of repeating animation - Cumulative animation

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

Using the path notation for a simple arc, we describe this example as:

```
<img ...>
   <animateMotion path="c( 3 5 8 5 10 0)" dur="10s"
      accumulate="sum" repeatCount="10" />
</img>
```

@@ Pictures would help here

The image moves from the original position along the arc over the course of 10 seconds. As the animation repeats, it builds upon the previous value and begins the second arc where the first one ended. In this way, the image "bounces" across the screen. This could be described as a complete path, but the path description would get quite large, and would be more cumbersome to edit.

Note that cumulative animation only controls how a single animation accumulates the results of the animation function as it repeats. It specifically does not control how one animation interacts with other animations to produce a presentation value. This latter behavior is described in the section Additive animation .

Any numeric attribute that supports addition can support cumulative animation. For example, we can grow the "width" of an SVG "rect" element by 100 pixels in 100 seconds.

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

After 10 seconds, the rectangle is 30 pixels wide. The animation repeats, and builds upon the previous values growing to 40 pixels after 20 pixels, and up to 120 pixels wide after all ten repeats.

The behavior of repeating animations is controlled with the `accumulate` attribute:

accumulate = "**none"** | **"sum**"
Controls whether or not the animation is cumulative.
"sum"
Each repeat iteration after the first builds upon the last value of the previous iteration.
"none"
Repeat iterations are not cumulative, and simply repeat the animation function **`f(t)`**. This is the default.
This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.
Cumulative animation is not defined for "*to animation*". This attribute will be ignored if the animation function is specified with only the `to` attribute. See also Specifying function values .

To produce the cumulative animation behavior, the animation function **`f(t)`** must be modified slightly. Each iteration after the first must add in the last value of the previous iteration - this is expressed as a multiple of the last value specified for the animation function. Note that cumulative animation is defined in terms of the values specified for the animation behavior, and not in terms of sampled or rendered animation values. The latter would vary from machine to machine, and could even vary between document views on the same machine.

Let $f_i(t)$ represent the cumulative animation function for a given iteration `i`.

The first iteration $f_0(t)$ is unaffected by `accumulate`, and so is the same as the original animation function definition.

$$f_0(t) = f(t)$$

Let **`ve`** be the last value specified for the animation function (e.g. the "to" value or the last value in a "values" list). Each iteration after the first (i.e. $f_i(t)$ where $i >= 1$) adds in the computed offset:

$$f_i(t) = (ve * i) + f(t)$$

Freezing animations

@@ Rewrite to make reference to (and use) Timing module’s definition. (Say what it means to freeze an animation, rather than define the fill attribute.)

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

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

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

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

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

The animation ends 15 seconds after the image begin, but the image remains at the top value of 103. The attribute "freezes" the last value of the animation @@ "for the period of time defined by the `fill` attribute" will make sense here once this section is rewritten.

The freeze behavior of an animation is controlled using the "fill "attribute:

`fill` = "**freeze" | "remove**"

This attribute can have the following values:

freeze

The animation effect F(t) is defined to freeze the effect value at the last value of the active duration. The animation effect is "frozen" @@ "for the period of time defined by the `fill` attribute", as above.

remove

The animation effect is removed (no longer applied) when the active duration of the animation is over. After the active end **AE** of the animation, the animation no longer affects the target (unless the animation is restarted - see Restarting animations).

This is the default value.

@@ Need to deal with the other values for fill included in the timing module ’hold’ and maybe ’transition’

This functionality is also useful when a series of motions are defined that should build upon one another, as in this example:

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

The image moves along the first path, and then starts the second path from the end of the first, then follows the third path from the end of the second, and stays at the final point. The semantics of the `additive` attribute are defined in the next section.

Note that if the active duration cuts short the simple duration (including the case of partial repeats), then the freeze value is defined by the shortened simple duration. In the following example, the animation function repeats two full times and then again for one-half of the simple duration. In this case, the freeze value will be 15:

```
<animate from="10" to="20" dur="4s"
         repeatCount="2.5" fill="freeze" .../>
```

In the following example, the `dur` attribute is missing, and so the simple duration is indefinite. The active duration is constrained by `end` to be 10 seconds. Since interpolation is not defined, the freeze value will be 10:

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

Additive animation

It is frequently useful to define animation using offsets or deltas from an attribute's value, rather than absolute values. A simple "grow" animation can increase the width of an object by 10 pixels:

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

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

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

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

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

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

The additive behavior of an animation is controlled by the `additive` attribute:

`additive` = "**replace" | "sum**"
: Controls whether or not the animation is additive.
  "sum"
  : Specifies that the animation will add to the underlying value of the attribute and other lower priority animations.
  "replace"
  : Specifies that the animation will override the underlying value of the attribute and other lower priority animations. This is the default, however the behavior is also affected by the animation value attributes **`by`** and **`to`**, as described in "How from, to and by attributes affect additive behavior ", below.

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

The host language must specify which attributes support additive animation. It may be defined for numeric attributes and other data types for which an addition function is defined. This may include numeric attributes for concepts such as position, widths and heights, sizes, etc. It also may include color (refer to The animateColor element) and other data types as specified by the host language. Some numeric attributes (e.g. a telephone number attribute) may not sensibly support addition.

Attribute types such as strings and Booleans, for which addition is not defined, cannot support additive animation.

While many animations of numerical attributes will be additive, this is not always desired. As an example of an animation that is defined to be non-additive, consider a hypothetical extension animation "mouseFollow" that causes an object to track the mouse.

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

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

How from, to and by attributes affect additive behavior.

The attribute values `to` and `by`, used to describe the animation function , can override the `additive` attribute in certain cases:

- If **`by`** is used without **`from`**, the animation is defined to be additive (i.e. the equivalent of `additive="sum"`).
- If **`to`** is used without **`from`** (i.e. a "to animation"), and if the attribute supports addition, the animation is defined to be a kind of mix of additive and non-additive. The underlying value is used as a starting point as with additive animation, however the ending value specified by the **`to`** attribute overrides the underlying value as though the animation was non-additive.

For the hybrid case of a "*to-animation*", the animation function **`f(t)`** is defined in terms of the underlying value, the specified `to` value, and the current value of **`t`** (i.e. time) relative to the simple duration **`d`**.

v_{cur} is the current base value (at time t)
v_{to} is the defined "to" value

$$f(t) = v_{cur} + ((v_{to} - v_{cur}) * (t/d))$$

Note that if no other (lower priority) animations are active or frozen, this defines simple interpolation. However if another animation is manipulating the base value, the "*to-animation*" will add to the effect of the lower priority, but will dominate it as it nears the end of the simple duration, eventually overriding it completely. The value for **`F(t)`** when a "*to-animation*" is frozen (at the end of the simple duration) is just the `to` value. If a "*to-animation*" is frozen anywhere within the simple duration (e.g. using a repeatCount of "2.5"), the value for **`F(t)`** when the animation is frozen is the value computed for the end of the active duration. Even if other, lower priority animations are active while a "*to-animation*" is frozen, the value for **`F(t)`** does not change.

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

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

The presentation value for "x" in the example above, over the course of the 10 seconds is presented in Figure 2 below. These values are simply computed using the formula described above. Note that the value for **`F(t)`** for A2 is the presentation value for "x".

Time	`F(t)` for A1	`F(t)` for A2
0	0	0
1	-1	0.1
2	-2	0.4
3	-3	0.9
4	-4	1.6
5	-5	2.5
6	-6	3.6
7	-7	4.9
8	-8	6.4
9	-9	8.1
10	-10	10

Figure 2 - Effect of Additive to-animation example

Additive and Cumulative animation

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

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

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

The animation overrides whatever original value was set for "top", and begins at the value 20. It moves down by 10 pixels to 30, then repeats. It is cumulative, so the second iteration starts at 30 and moves down by another 10 to 40. Etc.

When a cumulative animation is also defined to be additive, both features function normally. The accumulated effect for **`F(t)`** is used as the value for the animation, and is added to the underlying value for the target attribute. Refer also to The animation sandwich model .

Restarting animations

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

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

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

3.3.4 Handling syntax errors

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

In particular, this means that if other animation elements are defined to begin or end relative to an animation that "has no effect", the other animation elements will begin and end as though there were no syntax errors. The presentation runtime may indicate an error, but need not halt presentation or animation of the document. Some host languages and/or runtimes may choose to impose stricter error handling (see also Error handling semantics for a discussion of host language issues with error handling). Authoring environments may also choose to be more intrusive when errors are detected.

3.3.5 The animation sandwich model

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

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

The model accounting for the OM and concurrently active or frozen animations for a given attribute is described as a "sandwich", an analogy to the layers of meat and cheeses in a "submarine sandwich". On the bottom of the sandwich is the base value taken from the OM. Each active (or frozen) animation is a layer above this. The layers (i.e. the animations) are placed on the sandwich in order according to *priority*, with higher priority animations placed above lower priority animations. Note that animations manipulate the presentation value coming out of the OM in which the attribute is defined, and pass the resulting value on to the next layer of document processing. This does not replace or override any of the normal document OM processing cascade.

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

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

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

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

elements are defined relative to the same syncbase or event-base.

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

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

Some attributes that support additive animation have a defined legal range for values (e.g. an opacity attribute may allow values between 0 and 1). In some cases, an animation function may yield out of range values. It is up to the implementation to clamp the results at the top of the animation stack to the legal range before applying them to the presentation value. However, the effect of all the animations in the stack should be combined, before any clamping is performed. Although individual animation functions may yield out of range values, the combination of additive animations in the animation stack may still be legal. Clamping only the final result and not the effect of the individual animation functions provides support for these cases. The host language must define the clamping semantics for each attribute that can be animated. As an example, this is defined for The animateColor element .

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

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

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

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

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

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

3.3.6 Implications of Timing Model for animation

The model of timing defined in the Timing module has several important results for animation: the definition of repeat, and the value sampled during the "frozen" state.

When repeating an animation, the arithmetic follows the end-point exclusive model. Consider the example:

```
<animation dur="4s" repeatCount="4" .../>
```

At time 0, the simple duration is sampled at 0, and the first value is applied. This is the *inclusive* begin of the interval. The simple duration is sampled normally up to 4 seconds. However, 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( activeTime, d )
```

or

`F(t) = f( 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.

This implies that the last value of an animation function **`f(t)`** may never actually be applied (e.g. for a linear interpolation). In the case of an animation that does not repeat and does not specify **`fill="freeze"`**, this may in fact be the case. However, in the following example, the appropriate value for the frozen state is clearly the "to" value:

```
<animation from="0" to="5" dur="4s" fill=freeze .../>
```

This does not break the interval timing model, but does require an additional qualification for the animation function **`F(t)`** while in the frozen state:

- If the active duration is an even multiple of the simple duration, the value to apply in the frozen state is the last value defined for the animation function **`f(t)`**.

The definition of accumulate also aligns to this model. The arithmetic is effectively inverted and values accumulate by adding in a *multiple* of the last value defined for the animation function **`f(t)`**.

3.3.7 Animation function value details

Animation function values must be legal values for the specified attribute. Three classes of values are described:

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

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

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

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

1. The first interface converts a string (the animation function value) to a unitless, canonical number (either an integer or a floating point value). This allows animation elements to interpolate between values without requiring specific

knowledge of data types like CSS-length. The interface will likely require a reference to the target attribute, to determine the legal abstract values. If the passed string cannot be converted to a unitless scalar, the animation element will treat the animation function values as strings, and the `calcMode` will default to "discrete".
2. The second interface converts a unitless canonical number to a legal string value for the target attribute. This may, for example, simply convert the number to a string and append a suffix for the canonical units. The animation element uses the result of this to actually set the presentation value.

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

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

3.4 Animation elements

This section defines the syntax and semantics of animation elements. @@ DTD definitions are used in this working draft. The Working Group expects to replace them with schema-based defintions prior to Recommendation.

3.4.1 Common syntax DTD definitions

Timing attributes are defined in the SMIL Timing module.

Animation attributes

```
<!ENTITY % animAttrs
  attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED
  additive       (replace | sum) "replace"
  accumulate     (none | sum) "none"
>

<!ENTITY % animTargetAttr
  targetElement  IDREF  #IMPLIED
>

<!ENTITY % animLinkAttrs
  type     (simple | extended | locator | arc) #FIXED "simple"
  show     (new | embed | replace) #FIXED 'embed'
  actuate  (user | auto) #FIXED 'auto'
  href     CDATA  #IMPLIED
>
```

3.4.2 The animate element

The **<animate>** element introduces a generic attribute animation that requires little or no semantic understanding of the attribute being animated. It can animate numeric scalars as well as numeric vectors. It can also animate discrete sets of non-numeric attributes. The **<animate>** element is an empty element - it cannot have child elements.

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

```
<!ELEMENT animate EMPTY>
<!ATTLIST animate
  %timingAttrs
  %animAttrs
  id              ID      #IMPLIED
  calcMode        (discrete | linear | paced | spline ) "linear"
  values          CDATA   #IMPLIED
  keyTimes        CDATA   #IMPLIED
  keySplines      CDATA   #IMPLIED
  from            CDATA   #IMPLIED
  to              CDATA   #IMPLIED
  by              CDATA   #IMPLIED
>
```

Numerous examples are provided above.

3.4.3 The set element

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

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

The `<set>` element supports all the timing attributes to specify the simple and active durations. However, the `repeatCount` and `repeatDur` attributes will just affect the active duration of the `<set>`, extending the effect of the `<set>` (since it is not really meaningful to "repeat" a static operation). Note that using `fill="freeze"` with `<set>` will have the same effect as defining the timing so that the active duration is "indefinite".

The `<set>` element supports a more restricted set of attributes than the `<animate>` element (in particular, only one value is specified, and no interpolation control is supported):

```
<!ELEMENT set EMPTY>
<!ATTLIST set
  %timingAttrs
  id              ID      #IMPLIED
  attributeName   CDATA   #REQUIRED
  attributeType   CDATA   #IMPLIED
  to              CDATA   #IMPLIED
>
```

`to` = "**<value>**"

Specifies the value for the attribute during the duration of the `<set>` element. The argument value must match the attribute type.

Examples

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

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

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

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

3.4.4 The animateMotion element

In order to abstract the notion of motion paths across a variety of layout mechanisms, we introduce the **`<animateMotion>`** element. This describes motion in the abstract - the host language defines the layout model and must specify the precise semantics of motion.

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

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

The **`<animateMotion>`** element adds an additional syntax alternative for specifying the animation, the "`path`" attribute. This allows the description of a path using a subset of the SVG path syntax. Note that if a path is specified, it will override any specified values for `values` or `from/to/by` attributes.

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

```
<!ELEMENT animateMotion EMPTY>
<!ATTLIST animateMotion
  %timingAttrs
  id             ID     #IMPLIED
  additive       (replace | sum) "replace"
  accumulate     (none | sum) "none"
  calcMode       (discrete | linear | paced | spline) "paced"
  values         CDATA  #IMPLIED
  from           CDATA  #IMPLIED
  to             CDATA  #IMPLIED
  by             CDATA  #IMPLIED
  keyTimes       CDATA  #IMPLIED
  keySplines     CDATA  #IMPLIED
  path           CDATA  #IMPLIED
  origin         (default) "default"
/>
```

`path` = "**<path-description>**"

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

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

Move To commands - "M <x> <y>" or "m <dx> <dy>"
Start a new sub-path at the given (x,y) coordinate. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands.

Line To commands - "L <x> <y>" or "l <dx> <dy>"
Draw a line from the current point to the given (x,y) coordinate which becomes the new current point. A number of coordinate pairs may be specified to draw a polyline.

Horizontal Line To commands - "H <x>" or "h <dx>"
Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). Multiple x values can be provided (although this generally only makes sense for the relative form).

Vertical Line To commands - "V <y>" or "v <dy>"
Draws a vertical line from the current point (cpx, cpy) to (cpx, y). Multiple y values can be provided (although generally only makes sense for the relative form).

Closepath commands - "Z" or "z"
The "closepath" causes an automatic straight line to be drawn from the current point to the initial point of the current subpath.

Cubic bezier Curve To commands -
"C <x1> <y1> <x2> <y2> <x> <y>" or
"c <dx1> <dy1> <dx2> <dy2> <dx> <dy>"
Draws a cubic Bezier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. Multiple sets of coordinates may be specified to draw a polybezier.

When a `path` is combined with "linear" or "spline" `calcMode` settings, the number of values is defined to be the number of points defined by the path, unless there are "move to" commands within the path. A "move to" command does not count as an additional point for the purpose of `keyTimes` and `spline`, and should not define an additional "segment" for the purposes of timing or interpolation. When a `path` is combined with a "paced" `calcMode` setting, all "move to" commands are considered to have 0 length (i.e. they always happen instantaneously), and should not be considered in computing the pacing.

`calcMode`

Defined as above in Animation function calculation modes , but note that the default `calcMode` for `animateMotion` is "paced". This will produce constant velocity motion across the path.
The use of "discrete" for the `calcMode` together with a "`path`" specification is allowed, but is generally not useful (it will simply jump the target element from point to point).
The use of "linear" for the `calcMode` with more than 2 points described in "`values`", "`path`" or "`keyTimes`" may result in motion with varying velocity. The "linear" `calcMode` specifies that time is evenly divided among the segments defined by the "`values`" or "`path`" (note: any "`keyTimes`" list

defines the same number of segments). The use of "linear" does not specify that time is divided evenly according to the *distance* described by each segment.
For motion with constant velocity, `calcMode` should be set to "paced".
For complete velocity control, calcMode can be set to "spline" and the author can specify a velocity control spline with "`keyTimes`" and "`keySplines`".

`origin` = "**default**"

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

@@Should add an example, although some are included above.

3.4.5 The animateColor element

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

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

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

```
<!ELEMENT animateColor EMPTY>
<!ATTLIST animateColor
  %animAttrs
  %timingAttrs
  id            ID     #IMPLIED
  calcMode      (discrete | linear
                 | paced | spline ) "linear"
  values        CDATA  #IMPLIED
  from          CDATA  #IMPLIED
  to            CDATA  #IMPLIED
```

```
    by              CDATA   #IMPLIED
    keyTimes        CDATA   #IMPLIED
    keySplines      CDATA   #IMPLIED
  >
```

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

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

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

Note to implementers: When `animateColor` is specified as a "to animation", the animation function should assume Euclidean RGB-cube distance where deltas must be computed. See also Specifying function values and How from, to and by attributes affect additive behavior . Similarly, when the `calcMode` attribute for `animateColor` is set to "paced", the animation function should assume Euclidean RGB-cube distance to compute the distance and pacing.

3.5 Integrating SMIL Animation into a host language

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

3.5.1 Required host language definitions

The host language designer must provide the basis for animation semantics in the context of the particular host language.

The host language designer must integrate the SMIL Timing module into the host language.

3.5.2 Required definitions and constraints on animation targets

Specifying the target element

The host language designer must choose whether to support the `targetElement` attribute, or the XLink attributes for specifying the target element . Note that if the XLink syntax is used, the host language designer must decide how to denote the XLink namespace for the associated attributes. The namespace can be fixed in a DTD, or the language designer can require colonized attribute names to denote the XLink namespace for the attributes. The required XLink attributes have fixed values, and so may also be specified in a DTD, or can be required on the animation elements. Host language designers may require that the optional XLink attributes be specified. These decisions are left to the host language designer - the syntax details for XLink attributes do not affect the semantics of SMIL Animation.

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

Target attribute issues

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

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

Additive and cumulative animation is supported for any attribute for which animation is supported and for which addition is defined by the host language for the underlying data type, unless the attribute is specifically excluded from cumulative and additive animation.

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

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

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

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

The host language must specify the formats supported for numeric attribute values. This includes integer values and especially floating point values for attributes such as `keyTimes` and `keySplines`. As a reasonable minimum, host language designers are encouraged to support the format described in [CSS2]. The specific reference within the CSS specification for these data types is section 4.3.1 Integers and real numbers of [CSS2].

Integrating animateMotion functionality

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

Example: SVG

As an example, SVG [SVG] integrates SMIL Animation. It specifies which of the elements, attributes and CSS properties may be animated. Some attributes (e.g. "viewbox" and "fill-rule") support only discrete animation, and others (e.g. "width", "opacity" and "stroke") support interpolated and additive animation. An example of an attribute that does not support any animation is the "xlink:actuate" attribute on the `<use>` element (the value of this attribute is fixed to "auto" in the DTD).

@@ The XLink syntax used here may be out of date (actuate=auto is now actuate=onLoad?). Once SVG/XLink settles on values for actuate, this section must be updated.

SVG details the format of numeric values, describing the legal ranges and allowing "scientific" (exponential) notation for floating point values.

3.5.3 Constraints on manipulating animation elements

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

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

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

3.5.4 Extending animation

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

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

3.5.5 Error handling semantics

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

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

3.5.6 SMIL Animation namespace

Language designers can choose to integrate SMIL Animation as an independent namespace, or can integrate SMIL Animation names into a new namespace defined as part of the host language. Language designers that wish to put the SMIL Animation functionality in an isolated namespace should use the following namespace:

@@ URI to be confirmed by W3C webmaster. Differs from [SMIL-ANIMATION].

4. SMIL Content Control

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

4.1 Introduction

This Section defines the SMIL content control module. This module contains elements and attributes which provide for runtime content choices and optimized content delivery. Since these elements and attributes are defined in a module, designers of other markup languages can reuse the functionality in the SMIL content control module when they need to include media content control in their language. Conversely, language designers incorporating other SMIL modules do not need to include the content module if other content control functionality is already present.

Proposed Extensions to SMIL 1.0 content control functionality include:

- Allow definition of priorities for different media objects. This allows for example dropping certain objects from the presentation or dropping layers in a layered encoding when there are insufficient resources (e.g. bandwidth, CPU).
- Allow additional test-attributes (e.g. CPU-type, ...).
- Allow author-defined test-attributes.
- Allow user to see media objects that are important to him/her even though author excluded them at the current bitrate (accessibility requirement).
- Allow display of time dependent links in a static list (accessibility requirement).
- Allow declaration of media objects to be preloaded, as bandwidth allows, to improve presentation quality.

4.2 Content Selection

SMIL 1.0 provides a "test-attribute" mechanism to process an element only when certain conditions are true, e.g. when the client has a certain screen-size. 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 to have the client select one of them. SMIL Boston includes these features and extends them by supporting new system test-attributes, as well as the ability to customize a presentation to an individual viewer by providing author defined, user selected test-attributes.

4.2.1 The `<switch>` Element

The switch element allows an author to specify a set of alternative elements from which only one acceptable element should be chosen. In SMIL Boston, an element is acceptable if the element is a SMIL Boston element, the media-type can be decoded (if the element declares media), and all of the test-attributes of the element evaluate to "true". When integrating content control into other languages, the language designer must specify what constitutes an "acceptable element."

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.

Thus, authors should order the alternatives from the most desirable to the least desirable. Furthermore, authors should 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). Implementations should NOT arbitrarily pick an object within a <switch> when test-attributes for all child elements fail.

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.

Attributes

The switch element can have the following attributes:

id
: An XML identifier

title
: This 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).

4.2.2 Predefined 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 one of the test attributes specified for an element evaluates to "false", the element carrying this attribute is ignored.

Within the list below, the concept of "user preference" may show up. 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.

This version of SMIL defines the following test attributes. Note that some 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.

systemBitrate (system-bitrate)
: This attribute specifies 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 users 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)
: This attribute allows authors to distinguish between a redundant text equivalent of the audio portion of the presentation (intended for audiences such as those with hearing disabilities or those learning to read who want or need this information) and text intended for a wide audience. The attribute can has the value "on" if the user has indicated a desire to see closed-captioning information, and it has the value "off" if the user has indicated that they don’t wish to see such information. Evaluates to "true" if the value is "on", and evaluates to "false" if the value is "off".

systemLanguage (system-language)
: The attribute value is a comma-separated list of language names as defined in [RFC1766].

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

Evaluates to "false" otherwise.

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

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

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

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi", presented simultaneously in the original Maori and English versions, would call for:

```
<audio src="foo.rm" systemLanguage="mi, en"/>
```

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

Authoring note: Authors should realize that if several alternative language objects are enclosed in a "switch", and none of them matches, this may lead to situations such as a video being shown without any audio track. It is thus recommended to include a "catch-all" choice at the end of such a switch which is acceptable in all cases.

systemOverdubOrCaption (system-overdub-or-caption)

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

systemRequired (system-required)

This attribute specifies the name of an extension. The extension may be a newly adopted language element or attribute, or may be the namespace prefix or URI for a namespace extension. Evaluates to "true" if the extension is supported by the implementation, otherwise, this evaluates to "false". [NAMESPACES]

systemScreenSize (system-screen-size)

Attribute values have the following syntax:

```
screen-size-val ::= 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 SMIL playback engine is capable of displaying a presentation of the given size. Evaluates to "false" if the SMIL playback engine is only capable of displaying smaller presentations.

systemScreenDepth (system-screen-depth)

This attribute specifies the depth of the screen color palette in bits required for displaying the element. The value must be greater than 0. Typical values are 1, 8, 24, 32 .... Evaluates to "true" if the SMIL playback engine is capable of displaying images or video with the given color depth. Evaluates to "false" if the SMIL playback engine is only capable of displaying images or video with a smaller color depth.

systemOverdubOrSubtitle

This attribute specifies whether subtitles or overdub is rendered for people who are watching a presentation where the audio may be in a language in which they are not fluent. This attribute can have two values: "overdub", which selects for substitution of one voice track for another, and "subtitle", which means that the user prefers the display of subtitles.

systemAudioDesc
: 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 for blind users like **systemCaptions** provides text captions for deaf users. The attribute has the value "on" if the user has indicated a desire to hear audio descriptions, and it has the value "off" if the user has indicated that they don't wish to hear audio descriptions. Evaluates to "true" if the value is "on", and evaluates to "false" if the value is "off".

systemOperatingSystem
: TBD

systemCPU
: TBD

systemContentLocation
: TBD (i.e. Streaming/Stored)

system???
: TBD (i.e. Selecting embedded information (element in aggregate))

system????
: TBD (i.e. Costs of accessing a stream, free or Pay-Per-View)

systemComponent
: CDATA that describes a component of the playback system, e.g. user-agent component/feature, number of audio channels, codec, HW mpeg decoder, etc.

Examples

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 media player evaluates each of the "choices" (elements within the switch) one at a time, looking for an acceptable bitrate given the known characteristics of the link between the media player and media server.

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

2) Choosing between audio resources with different bitrates

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

```
...
<switch>
   <audio src="joe-audio-better-quality" systemBitrate="16000" />
   <audio src="joe-audio" systemBitrate="8000" />
</switch>
...
```

3) Choosing between audio resources in different languages

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

```
...
<switch>
   <audio src="joe-audio-french" systemLanguage="fr"/>
   <audio src="joe-audio-english" systemLanguage="en"/>
</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 can choose one of the alternatives.

```
...
<par>
  <text .../>
  <switch>
    <par systemScreenSize="1280X1024" systemScreenDepth="16">
    ........
    </par>
    <par systemScreenSize="640X480" systemScreenDepth="32">
    ...
    </par>
    <par systemScreenSize="640X480" systemScreenDepth="16">
    ...
    </par>
  </switch>
</par>
...
```

5) Distinguishing caption tracks from stock tickers

In the following example, captions are shown only if the user wants captions on.

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

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>
...
```

4.2.3 System Test Attribute In-Line Use

During the development of the SMIL 1.0, the issue of content selectability within a presentation received a great deal of attention. Early on, it was decided that a `<switch>` construct would form the basic selection primitive in the language. A `<switch>` allows a series of alternatives to be specified for a particular piece of content, one of which is selected by the runtime environment for presentation. An example of how a `<switch>` might be used to control the alternatives that could accompany a piece of video in a presentation would be:

```
...
<par>
  <video src="anchor.mpg" ... />
  <switch>
    <audio src="dutch.aiff"   systemLanguage="DU" systemCaptions="overdub" ... />
    <audio src="english.aiff" systemLanguage="EN" systemCaptions="overdub"... />
    <text  src="dutch.html"   systemLanguage="DU" systemCaptions="captions"... />
    <text  src="english.html" systemLanguage="EN" systemCaptions="captions"... />
  </switch>
</par>
...
```

This fragment (which is pseudo-SMIL for clarity) says that a video is played in parallel with one of: Dutch audio, English audio, Dutch text, or English text. SMIL does not specify the selection mechanism, only a way of specifying the alternatives. While `<switch>`-based content control is a powerful mechanism, it comes with two problems.

First, it restricts the resolution of a `<switch>` to a single alternative. (If you want Dutch audio and Dutch text, you need to specify a compound `<switch>` statement, but in so doing, you always get the compound result.)

Second, and more restrictively, it requires the author to explicitly state all of the possible combinations of input streams during authoring. If the user wanted Dutch audio and English text, this possibility must have been considered at authoring time.

A solution to both problems is to allow in-line use of System Test Attributes, as given in the following document fragment:

```
...
<par>
  <video src="anchor.mpg" ... />
  <switch>
    <audio src="dutch.aiff"   systemLanguage="DU" systemCaptions="overdub" ... />
    <audio src="english.aiff" systemLanguage="EN" systemCaptions="overdub"... />
    <text  src="dutch.html"   systemLanguage="DU" systemCaptions="captions"... />
    <text  src="english.html" systemLanguage="EN" systemCaptions="captions"... />
  </switch>
</par>
...
```

This example says: a video is accompanied by four other data objects, all of which are (logically) shown in parallel. This is, of course, exactly what happens: all five do run in parallel, but it could be that only the video and one audio stream are actually selected by the user (or a user agent) to be rendered during the presentation. At author time you know which logical streams are available, but it is only at runtime that you know which combination of all potentially available stream actually meet the user’s needs. Logically, 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 in the case that many independent alternatives exist.

4.2.4 User Groups

The provision of `<switch>`-based and in-line system test attributes provides a selection mechanism based on general system attributes. This version of SMIL extends this notion with the definition of user test attributes. User test attributes allow presentation authors to define their own test attributes for use in a specific document.

The elements used to provide user group functionality are:

The `<user_attributes>` element

A section within the SMIL head that contains definitions of each of the user groups. The elements within the section define a collection of author-specified test attributes that can be used in the document.

The `<u_group>` element

An author-defined grouping of related media objects. These are defined within the section delineated by the `<user_attributes>` elements that make up part of the document header, and they are referenced within a media object definition.

The `<u_group>` element supports the following attributes:

- **id**: the internal name of the attribute.
- **title**: a string that can be used by a user-interface to provides a selection mechanism.
- **u_state**: the evaluated state of the `<u_group>`. The initial state for the `<u_group>` is given in the value of this attribute, if unspecified, it defaults to **RENDERED**. The run-time state is defined by the user or the user agent via the SMIL DOM. If a particular playback environment does not (or cannot) support user selection, the **u_state** attribute controls the author-specified default presentation.
- **override**: the author is given the ability to block overrides to the initial state by explicitly prohibiting this in the `<u_group>` definition. It is up to the runtime environment to enforce this attribute. The attribute can also be used to influence adaptive behavior at lower level in the transport hierarchy.

  It would be good to have more explanation of this last use.

In addition to the `<user_attribute>` and `<u_group>` elements, this module provides a **u_group** attribute that can be applied to content requiring selection.

The `u_group` attribute

The **u_group** attribute is evaluated as a test attribute, if the **u_group** attribute evaluates to true, the associated element is evaluated, otherwise it and its content is skipped.

The following example shows how user groups can be applied within a SMIL document:

```
 1 <smil>
 2    <head>
 3       <layout>
 4          <!-- define projection regions -->
 5       </layout>
 6       <user_attributes>
 7          <u_group id="nl_aud" u_state="RENDERED" title="Dutch Audio Cap" override="allowed" />
 8          <u_group id="uk_aud" u_state="NOT_RENDERED" title="English Audio Cap" override="allowed" />
 9          <u_group id="nl_txt" u_state="NOT_RENDERED" title="Dutch Text Cap"override="allowed" />
10          <u_group id="uk_txt" u_state="NOT_RENDERED" title="English Text Cap" override="allowed" />
11       </user_attributes>
12    </head>
13    <body>
14       ...
15       <par>
16          <video src="announcer.rm" region="a"/>
17          <text src="news_headline.html" region="b"/>
18          <audio src="story_1_nl.rm" u_group="nl_aud"/>
19          <audio src="story_1_uk.rm" u_group="uk_aud-cam"/>
20          <text src="story_1_nl.html" u_group="nl_txt" region="c"/>
21          <text src="story_1_uk.html" u_group="uk_txt" region="d"/>
22       </par>
23       ...
24    </body>
25 </smil>
```

Lines 6 through 11 define the available groups. Each group contains an identifier and a title (which can be used by the user interface agent to label the group), as well as the (optional) initial state definition and override flag.

In line 7, a `<u_group>` named "nl_aud" is defined for Dutch audio captions that is initially set to **RENDERED**. The other groups in this (very simple) example are set to **NOT_RENDERED**.

In lines 15 through 22, a SMIL `<par>` construct is used to identify a portion of a presentation. In this `<par>`, a single video (line 16) is accompanied by two audio streams (18,19) and two text streams (20,21), one each for English and Dutch. The `<par>` also contains a text title that contains a headline.

The interaction of the user interface and the initial state determine which objects are rendered. Note that the same attributes are used across the entire document, meaning that the user only needs to select his/her content preferences once to control related groups of information. In the example, user is free to have the video and headline text accompanied by any combination of English and Dutch captions. (Note that if two audio captions are selected, the player will need to determine how these are processed for delivery.)

While this example shows in-line use of user groups, the groups could also be applied as test attributes in a `<switch>`. Similarly, the system test attributes typically found in a `<switch>` could also be used in-line as a control attribute on an element along with the **u_group** attribute.

A previous version of this specification used camelCase for the user group elements and attributes instead of the underlined convention used here. We need to standardize this across the SMIL modules.

4.3 Presentation Priority/Grouping

The following is still under development by the SYMM Working Group. The working group is interested in considering this functionality but the syntax and semantics described here are only preliminary thinking.

Define a means to group collections of objects that share a common policy. A Channel defines a partitioning of elements into groups each group has a common set of access policies control use of quasi-physical resources: - priority - common server - common access rights / charging model - local resource use (layout, devices, etc.)

4.4 User-Centered Adaptation

The following is still under development by the SYMM Working Group. The working group is interested in considering this functionality but the syntax and semantics described here are only preliminary thinking.

Focus on presentation as collection of content: each of the components may have a different user-level representation, encoding:

- (natural) language
- level of semantic detail
- ability / rights to access particular type of content

At author-time, you know alternatives; at use-time, you select

4.5 Presentation Optimization

4.5.1 The `<prefetch>` element

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 to make the document playback more smoothly. 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.

The `<prefetch>` element, like media object elements, can have `id` and `src`. If SMIL Boston Timing is integrated into the document, `begin`, `end`, `dur`, `clipBegin`, and `clipEnd` attributes are also available. The `id` and `src` elements are the same as for other media objects `id` names the element for reference in the document and `src` names the resource to be prefetched. When a media object with the same `src` URL is encountered the user-agent can use any data it prefetched to begin playback without rebuffering or other interruption. The timing attributes `begin`, `end`, `dur` would constrain the presentation time period for prefetching the element. At the end of the presentation time specified by `end` or `dur`, the prefetch operation should stop. The `clipBegin` and `clipEnd` elements are used to identify the part of the src clip to prefetch, if only the last 30s of the clip are being played, we don't want to prefetch it from the beginning. Likewise if only the middle 30 seconds of the clip are begin played, we don't want to prefetch more data than will be played.

The `mediaSize`, `mediaTime`, and `bandwidth` Attributes

In addition to the attributes allowed on Media Object Elements, the following attributes are allowed:

`mediaSize : 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 : 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%.

`bandwidth : 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%

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

For descrete media (non-time based media like text/html or image/png) using the `mediaTime` attribute causes the entire resource to be fetched.

Documents must still playback even when the prefetch elements are ignored, although rebuffering or pauses in presentation of the document may occur.

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. Note that prefetching data from a URL that changes the content dynamically is dangerous if the entire resource isn't prefetched as the 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-cachable would have to be refetched each time it was played, where content that is cachable could be prefetched once, with the results of the prefetch cached for future use.

If the `clipBegin` or `ClipEnd` in the media object are different from the prefetch, an implementation can use any data that was fetched and applies 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; @@ range?
Subframes          ::= 2DIGIT; @@ range?
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
```

Examples

1) Prefetch the image so it can be displayed immediately after the video ends:

```
<smil>
  <body>
    <seq>
      <par>
        <prefetch id="endimage"
```

```
src="http://www.w3c.org/logo.gif"/>
        <text id="interlude"
src="http://www.w3c.org/pleasewait.html" fill="freeze"/>
      </par>
      <video id="main-event"
src="rtsp://www.w3c.org/video.mpg"/>
      <image src="http://www.w3c.org/logo.gif"
fill="freeze"/>
    </seq>
  </body>
</smil>
```

No timing is specified so default timing applies in the above example. 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.w3c.org/up.gif"/>
    <prefetch id="downimage"
src="http://www.w3c.org/down.gif"/>
    ....
    <!-- script will change the graphic on rollover -->
    <img src="http://www.w3c.org/up.gif"/>
  </body>
</html>
```

4.6 Open Issues

Can prefetch elements be used as timebases for sync? This could be an useful capability to be supported. We should be able to start a prefetch and not play the content until it completes. This means that prefetch has to have effective begin and end, depending upon how long it actually takes to get the data. Of course, if prefetching is optional, we need to decide when the begin and end events fire. However this introduces the problem of how to handle errors. Even though the prefetch may not be allowed or fail, there may be other things dependant upon the timing of the prefetch element. In this case it is appropriate for the element's timing to continue and fire begin\end events as if the prefetch element ran to completion. Since this is all very complicated, and prefetch is intended to be transparent, one idea is that we explicitly prohibit prefetch from being a syncbase. This is not as simple as it sounds, say that a prefetch element is in the middle of a <seq>. Maybe the simplest solution is to allow prefetch as a syncbase, and to say that for sync purposes, all prefetch elements always have duration zero, and fire begin\end events event if the prefetch itself fails or is not allowed

5. SMIL Layout Module

Editors
Aaron Cohen (aaron.m.cohen@intel.com), Intel
Dick Bulterman (Dick.Bulterman@oratrix.com), Oratrix

5.1 Introduction

This Section defines the SMIL layout module. This module contains elements and attributes allowing for positioning of media elements on the rendering surface (either visual or acoustic). 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. Therefore, language designers incorporating other SMIL modules do not need to include the layout module if sufficient layout functionality is already present.

The major changes with respect to the layout elements and attributes in SMIL 1.0 [SMIL10] is the addition of support for:

- multiple top-level layout windows,
- hierarchical region definition within a layout window

Other changes are minor. SMIL 1.0 already provides for using alternative layout models, for example CSS [SMIL-CSS2], [CSS2], and these can provide much of the additional functionality desired over SMIL basic layout.

It is the intention of this version of the Layout Module to align SMIL Boston Layout with current CSS2 functionality. There are some conflicts in mapping CSS2 layout to a language, such as SMIL, where the layout hierarchy is not reflected in the XML structure of the SMIL document. This necessitated dropping desirable features that could not be directly supported by mapping CSS to SMIL such as: multiple z-ordering within hierarchical regions, the alignment of objects within regions, and object-specific placement offsets within regions. It is desired that a future version of W3C layout technology will add support for these features to the SMIL language. A future version of the Layout module may include proof-of-concept support for these features.

5.2 Brief overview of SMIL basic layout

SMIL 1.0 includes a basic 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 `background-color`. 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 SMIL 1.0 basic layout syntax, CSS2 syntax, or some other syntax.

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 basic layout model:

```
<layout>

<region id="r" top="15" 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:

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

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

```
<layout type="text/css">

video, img { top:15px; left:20px; width:100px; height=50px; }

</layout>
```

Note that multiple layout models can be specified within a `<switch>` element, each with a different `type`. The first layout with a `type` supported by the implementation will be the one used.

5.3 Extensions to SMIL 1.0 Basic Layout

The extensions proposed for SMIL/Boston fall into two groups:

- multiple top-level layout windows,
- hierarchical region definition within a layout window

The characteristics of each extension group will be presented in this section. The full syntax will be described in later sections.

5.3.1 Multiple Top-Level Window Support

In SMIL 1.0, each presentation was rendered into a root window of a specific size/shape. The root window contained regions to manage the rendering of specific media objects.

This specification supports the concept of multiple top-level windows. Since there is no longer a single root window, we use the term *top-level* instead. The assignment of the regions to individual top level windows allows independent placement and resizing of each top-level window.

A top level window is declared in a manner similar to the SMIL 1.0 root layout window, except that multiple instances of the top level may occur:

```
<layout>
  <top-layout id="WinV" title=" Video  " width="320" height="240"/>
  <region id="pictures" title="pictures" height="100%" fit="meet"/>
  </top-layout>

  <top-layout id="WinC" title=" Captions  " width="320" height="60">
    <region id="captions" top="WinC" title="caption text" top="90%" fit="meet"/>
  </top-layout>
</layout>
```

In this example, two top-level windows are defined ("WinV" and "WinC"), and two regions are defined with one region assigned to WinV and the other to WinC. The definitions of the top-level windows and the contained regions use the new hierarchical layout functionality, as discussed in the next section.

The top-level windows function as rendering containers only, that is, they do not carry temporal significance. In other words, each window does not define a separate timeline or any other time-container properties. There is still a single master timeline for the SMIL presentation, no matter how many top-level windows have been created. This is important to allow synchronization between media displayed in separate top-level windows.

All top level windows are opened as soon as the presentation is started. If a window is closed (by the user) while any of the elements displayed in that window are active, there is no effect on the timeline of those elements. However, a player may choose not to decode content as a performance improvement.

For SMIL 1.0 compatibility, the `<root-layout>` element will continue to support SMIL 1.0 layout semantics. The new `<top-layout>` element will support the extension semantics and an improved, nested syntax.

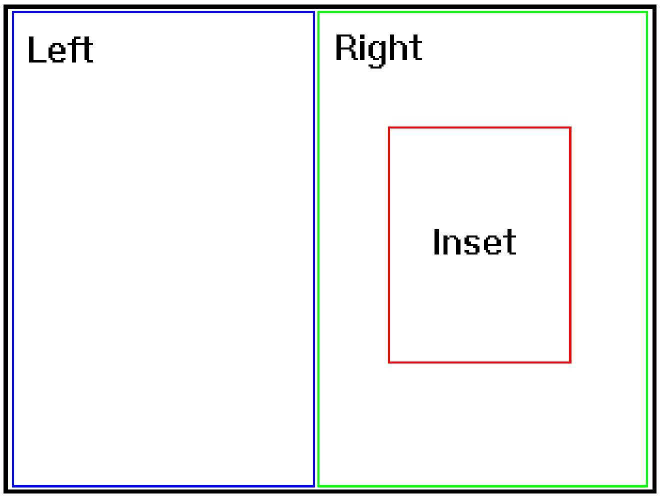
Note also that any one region may belong to at most one top-level (or root-level) window. Regions not declared as children of a `<top-layout>`element belong to the `<root-layout>` window. If no `<root-layout>` element has been declared, the region is assigned to a default window according to SMIL 1.0 layout semantics.

5.3.2 Hierarchical Region Layout

A new feature in this layout module is support for hierarchical layout. This allows for the declaration of regions nested inside other regions, much like regions are laid out inside the top level window declared by the `<top-layout>` element. For example, the following declares a top level window of 640 by 480 pixels, regions "left" and "right" which covers the left and right sides of the window respectively, and a subregion "inset" that is centered within "right".

```
<layout>
        <top-layout width="640px" height="480px" />
        <region id="left" top="0px" left="0px" width="320px" height="480px" />
        <region id="right" top="0px" left="320px" width="320px" height="480px">
                <region id="inset" top="140px" left="80" width="160px" height="200px" />
        </region>
</layout>
```

The resulting layout looks like this:



5.4 SMIL basic layout syntax and semantics

5.4.1 Elements and attributes

This section defines the elements and attributes that make up the SMIL basic layout module.

The `<layout>` element

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

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

It is recommended that profiles including the SMIL layout module also support the SMIL Content Control module. A document can then support multiple alternative layouts by enclosing several `<layout>` elements within the SMIL `<switch>` element. This could also be used to describe the document's layout using different layout languages. Support for the system test attributes in the SMIL Content Control module also enables greater author flexibility as well as user accessibility.

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

Element attributes

id
: This value uniquely identifies the layout element within a document. Its value is an XML identifier.

type
: This attribute specifies which layout language is used in the layout element. If the player does not understand this language, it must skip the element and all of its content up until the next `</layout>` tag. The default value of the type attribute is "text/smil-basic-layout". This identifier value supports SMIL 1.0 layout semantics. To enable the multiple top-level window and hierarchical layout extensions in this specification, declare the value of this attribute to be "text/smil-extended-layout".

Element content

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

region
root-layout

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

region
root-layout

top-layout

If the type attribute of the <layout> element has another value, the element contains character data.

The <region> element

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

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

```
<smil>
  <head>
    <layout>
        <root-layout width="320" height="480" />
        <region id="a" top="5" />
    </layout>
  </head>
  <body>
    <text region="a" src="text.html" dur="10s" />
  </body>
</smil>
```

The position of a region, as specified by its "top" and "left" attributes, is always relative to the parent geometry, which is defined by the parent element. For <region> elements whose immediate parent is a layout element, the region position is defined relative to the root window declared in the sibling <root-layout> element. For <region> elements that are children of a <top-layout> element the region position is defined relative to the top level window declared in the parent <top-layout> element.

For <region> elements whose immediate parent is another <region> element, the sub-region position is defined relative to the position of the region defined by the parent element. Note that this is only allowed for regions that are descendants of a <top-layout> element.

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

Note that a sub-region may be defined in such a way as to extend beyond the limits of its parent. In this case the sub-region should be clipped to the parent boundaries.

Element attributes

The `<region>` element can have the following visual attributes:

backgroundColor
: The use and definition of this attribute are identical to the "background-color" property in the CSS2 specification, except that SMIL basic layout does not require support for "system colors".

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

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

fit
: This attribute specifies the behavior if the intrinsic height and width of a visual media object differ from the values specified by the height and width attributes in the `<region>` element. This attribute does not have a 1-1 mapping onto a CSS2 property, but can be simulated in CSS2.
This attribute can have the following values:

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

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

meet
: Scale the visual media object while preserving its aspect ratio until its height or width is equal to the value specified by the height or width attributes, while none of the content is clipped. The object's left top corner is positioned at the top-left coordinates of the box, and empty space at the left or bottom is filled up with the background color.

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

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

The default value of "fill" is "hidden".

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

id
: A region element is applied to a position-able element by setting the region attribute of the position-able element to the id value of the region. The value of this attribute must be a valid XML identifier.
The "id" attribute is required for `<region>` elements.

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

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

title
: This 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).
It is strongly recommended that all `<region>` elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element

can be introduced into a SMIL document without this attribute.

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

volume
: Specifies the relative volume of an audio media element assigned to play within the given region. Valid values are any number between '0' and '100'. '0' represents the *minimum audible* volume level and 100 corresponds to the *maximum comfortable* level.

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

z-index
: This attribute defines the level of the region within the parent region stacking context. Elements assigned to higher level regions are rendered in front of lower level regions within the same parent region. Child regions are always placed in front of their parent region. This results in a two stage sorting of region visibility: first by parent-child containment, and then between regions within the same parent by assigned z-order.

The use and definition of this attribute are identical to the "z-index" property in the CSS2 specification, with the following exception:
If two boxes generated by elements A and B within the same parent region have the same stack level, then:

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

The default value of "z-index" is zero. To maintain compatibility with the CSS2 specification, the z-index attribute must always be zero for regions declared within a `<top-level>` element. Regions declared as children of the `<layout>` element may set the z-index to any CSS2 valid value.

The `<region>` element can have the following auditory attributes:

azimuth
: Specifies the direction that the audio media element assigned to the region appears to emanate from.
Position is described in terms of an angle within the range '-360deg' to '360deg'.

The value ’0deg’ means directly ahead in the center of the sound stage. ’90deg’ is to the right, ’180deg’ behind, and ’270deg’ (or, equivalently and more conveniently, ’-90deg’) to the left. The following values are also supported:

left-side
: Same as ’270deg’.

far-left
: Same as ’300deg’.

left
: Same as ’320deg’.

center-left
: Same as ’340deg’.

center
: Same as ’0deg’.

center-right
: Same as ’20deg’.

right
: Same as ’40deg’.

far-right
: Same as ’60deg’.

right-side
: Same as ’90deg’.

elevation
: Specifies the elevation that the audio media element assigned to the region appears to emanate from. Value is an angle, between ’-90deg’ and ’90deg’. ’0deg’ means on the forward horizon, which loosely means level with the listener. ’90deg’ means directly overhead and ’-90deg’ means directly below.

Element content

If the `<region>` element is a descendant of a <top-level> element, it may contain other `<region>` elements as children. Otherwise, the `<region>` element is an empty element.

The `<root-layout>` element

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

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

Element attributes

The `<root-layout>` element can have the following attributes:

backgroundColor
    Defined in backgroundColor under the `<region>` element.
background-color
    Defined in background-color under the `<region>` element.
height
    Sets the height of the root element. Only length values are allowed.
id
    Defined in id under the `<region>` element.
title
    Defined in title under the `<region>` element.
width
    Sets the width of the root element. Only length values are allowed.

Element content

The `<root-layout>` element is an empty element.

This element supports the SMIL 1.0 syntax where the `<root-layout>` element is an empty sibling of the top level `<region>` elements. Hierarchical region layout is not supported on windows declared with the `<root-layout>` element. That is, `<region>` elements that are assigned to the window declared by `<root-layout>` may not be nested.

The semantics of the `<root-layout>` element are as in SMIL 1.0: the attributes of the `<root-layout>` element determine the size of the top-level presentation window, and the declared sibling regions are arranged within this top level window.

The `<top-layout>` element

The `<top-layout>` element determines the size of the a window in which the SMIL presentation is rendered, as well as serving as a top level window in which to place child `<region>` elements.

Multiple `<top-layout>` elements may appear within a single `<layout>` element, each declaring an independent top level window.

Element attributes

The `<top-layout>` element can have the following attributes:

backgroundColor
    Defined in backgroundColor under the `<region>` element.
background-color
    Defined in background-color under the `<region>` element.
height
    Sets the height of the root element. Only length values are allowed.
id
    Defined in id under the `<region>` element.

title
: Defined in title under the `<region>` element.

width
: Sets the width of the root element. Only length values are allowed.

Element content

The `<top-layout>` element may contain `<region>` elements, or be empty.

The `<top-layout>` element supports the SMIL extended layout facilities. Allowing multiple `<top-layout>` elements within a single `<layout>` element supports multiple top level windows. Allowing the nesting of regions within a `<top-layout>` element provides support for hierarchical layout.

Each instance of a `<top-layout>` element determine the size of a separate top-level presentation window, and the descendant regions are arranged within this top level window.

The `region` attribute

The "region" attribute is applied to an element in order to specify which rendering region is assigned to the element. The attribute specifies the XML identifier of the abstract rendering region (either visual or acoustic) defined within the layout section of the document. If no rendering surface with the given identifier is defined in the layout section, the values of the formatting properties of this element are defined by the default layout .

The language integrating this module must specify which elements have a "region" attribute and any inheritance of the attribute.

5.4.2 SMIL basic layout language details

SMIL basic layout is consistent with the visual rendering model defined in CSS2, it reuses the formatting properties defined by the CSS2 specification, and newly introduces the "fit" attribute [CSS2]. The reader is expected to be familiar with the concepts and terms defined in CSS2.

Fixed property values

Editor: This should probably be moved to the language profile.

The following stylesheet defines the values of the CSS2 properties "display" and "position" that are valid when using SMIL basic layout with the SMIL language. These property values are fixed:

```
a           {display:block}

anchor      {display:block}

animate     {display:none}
```

```
animation   {display: block;
             position: absolute}
area        {display:block}
body        {display: block}
head        {display: none}
excl        {display: block}
img         {display: block;
             position: absolute}
layout      {display: none}
meta        {display: none}
par         {display: block}
region      {display: none}
ref         {display: block;
             position: absolute}
root-layout {display: none}
seq         {display: block}
smil        {display: block}
switch      {display:block}
text        {display: block;
             position: absolute}
textstream  {display: block;
             position: absolute}
video       {display: block;
             position: absolute}
```

Any other XML language using SMIL basic layout will have to define similar fixed attributes for its elements. Note that as a result of these definitions, all absolutely positioned elements (`<animation>`, `<img>`, `<ref>`, `<text>`, `<textstream>` and `<video>`) are contained within a single containing block defined by the content edge of the root element.

Default values

A profile integrating the SMIL basic layout module must define default values for all layout-related attributes of elements. These should be consistent with the initial values of the corresponding properties in CSS2.

5.5 Differences from SMIL 1.0 basic layout

This section lists the differences between this layout module and SMIL 1.0 basic layout.

- Support for system and user test attributes on the `<layout>` element. In this manner, an appropriate layout can be selected for users with different accessibility requirements, or players with different capabilities.
- Support for a hierarchical layout model; regions with parent/child relationships.
- Support for additional CSS2 compatible properties including:
  - Alternative placement methods, e.g. bottom/right.
  - Provide for control of acoustic rendering: audio levels, mixing, and placement in space.

5.6 Open Issues

- The editors have had to make some compromises and limitations in order to remain compatible with CSS2:
  - Cannot support sub-region placement, or layout position markers since there is no direct way to represent both region and position attributes on the same elements with CSS2.
  - Cannot support z-order on hierarchical regions because CSS2 has no way to assign an element to a layout hierarchy that is distinct from the XML tree.
- There is a strong dependency on the content control work to extend the use of test attributes both for accessibility and platform support.

6. The SMIL Linking Module

Editor

Lloyd Rutledge (Lloyd.Rutledge@cwi.nl), (CWI)

6.1 Introduction

The SMIL linking module defines the SMIL document elements for navigation hyperlink. These are navigations through the SMIL presentation that can be triggered by user interaction or other triggering events. SMIL provides only for in-line link elements. Links are limited to uni-directional single-headed links (i.e. all links have exactly one source and one destination resource).

XPointer [XPTR] allows components of XML documents to be addressed in terms of their placement in the XML structure rather than on their unique identifiers. This allows referencing of any portion of an XML document without having to modify that document. Without XPointer, pointing within a document may require adding unique identifiers to it, or inserting specific elements into the document, such as a named anchor in HTML. XPointers are put within the fragment identifier part of a URI. The SMIL specification does not require that browsers be able to process XPointers in SMIL URI attributes.

XLink (XML Linking Language) [XLINK] defines a set of generic attributes that can be used when defining linking elements in an XML-encoded language. SMIL borrows some constructs and concepts from XLink, mostly to stay consistent with HTML. SMIL does not conform to XLink.

Both XLink and XPointer are subject to change. At the time of this document's writing, neither is a full W3C recommendation. This document is based on the public Working Drafts ([XLINK], [XPTR]). It may change as these two formats change.

6.2 Linking into SMIL documents

The SMIL Linking Module supports name fragment identifiers and the '#' connector. The fragment part is an id value that identifies one of the elements within the referenced SMIL document. With this construct, SMIL supports locators as currently used in HTML (e.g. it uses locators of the form "http://foo.com/some/path#anchor1"), with the difference that the values are of unique identifiers and not the values of "name" attributes. Of course, this type of link can only target elements with an "id" attribute. Links using fragments enable authors to encode links to a SMIL presentation at the start time of a particular element rather than at the beginning of its presentation. If a link containing a fragment part is followed, the presentation should start as if the user had fast-forwarded the presentation represented by the destination document to the effective begin of the element designated by the fragment. See the discussion of linking to timing constructs in the SMIL Timing and Synchronization Module for more information.

There are special semantics defined for following a link containing a fragment part into a document containing SMIL timing. These semantics are defined in the SMIL Timing and Synchronization Module. In addition, the following rules apply for linking into a document written in the SMIL language:

1. It is forbidden to link to elements that are the content of <switch> elements. If the element addressed by the link is content of a <switch> element, then the presentation should start with the <switch> element.
2. If the fragment part id is not defined within the target document, the SMIL presentation should start from the beginning as if no fragment part were present in the URI.

6.2.1 Error handling

When a link into a SMIL document contains an unresolvable fragment identifier ("dangling link") because it identifies an element that is not actually part of the document, SMIL software should ignore the fragment identifier, and start playback from the beginning of the document.

When a link into a SMIL document contains a fragment identifier which identifies an element that is the content of a <switch> element, SMIL software should interpret this link as going to the parent <switch> element instead. If the parent is also a <switch>, then the link should be considered as accessing the first switch ancestor element whose parent is not also a <switch>. The result of the link traversal is thus to play the child of the located <switch> element that passes the usual switch child selection process.

6.3 Link Elements

The link elements allows the description of navigational links between objects.

SMIL linking provides only for in-line link elements. Links are limited to uni-directional single-headed links. That is, all links have exactly one source and one destination resource.

6.3.1 Handling of Links in Embedded Documents

Due to its integrating nature, the presentation of a SMIL document may involve other (non-SMIL) applications or plug-ins. For example, a SMIL browser may use an HTML plug-in to display an embedded HTML page. Vice versa, an HTML browser may use a SMIL plug-in to display a SMIL document embedded in an HTML page. Note that this is only one of the supported methods of integrating SMIL and HTML. Another alternative is to use the merged language approach. See the SMIL Timing and Integration Module for further details.

In embedded presentations, links may be defined by documents at different levels and conflicts may arise. In this case, the link defined by the containing document should take precedence over the link defined by the embedded object. Note that since this might require communication between the browser and the plug-in, SMIL implementations may choose not to comply with this recommendation.

If a link is defined in an embedded SMIL document, traversal of the link affects only the embedded SMIL document.

If a link is defined in a non-SMIL document which is embedded in a SMIL document, link traversal can only affect the presentation of the embedded document and not the presentation of the containing SMIL document. This restriction may be relaxed in future versions of SMIL.

6.3.2 The `<a>` Element

The functionality of the <a> element is very similar to the functionality of the <a> element in HTML 4.0 [HTML40]. For synchronization purposes, the <a> element is transparent. That is, it does not influence the synchronization of its child elements. <a> elements may not be nested. An <a> element must have an href attribute.

An <a> element can specify several triggers for its traversal simultaneously. For example, the element's content visual media can be selected by the user or the key specified by the accesskey attribute can be typed to trigger a traversal. In cases where multiple triggers are specified, any of them can activate the link's traversal. That is, an "or" is applied to the list of triggering conditions to determine if traversal occurs.

Traversal occurs if one of the conditions for traversal is met during the time that the <A> element is active. The means for determining is an <A> element is active are the same for determining if a media object is playing. This can be done through:

- natural progression forward along the presentation timeline due to time passing, possible by linking to a time before the <A> element and letting the presentation progress to it
- linking to the element's beginning by linking to the <A> element itself
- linking to an element playing in parallel with the <A> element, perhaps starting the <A> element at a moment after it its scheduled start time but before its scheduled end time.

Attributes

href
: This attribute contains the URI of the link's destination.
: The "href" attribute is required for `<a>` elements.

sourceVolume
: This attribute sets the volume of audio media objects in the presentation containing the link when the link is followed. Ignored if the presentation does not contain audio media objects. This attribute can have the same values as the "volume" property in CSS2 [CSS2].

destinationVolume
: This attribute sets the volume of audio media contained in the remote resource. Ignored if the remote resource does not contain audio media. This attribute can have the same values as the "localVolume" attribute.

sourcePlaystate
: This attribute controls the temporal behavior of the presentation containing the link when the link is traversed. It can have the following values:

- "play": When the link is traversed, the presentation containing the link continues playing.
- "pause": When the link is traversed, the presentation containing the link pauses. When the display of the destination resource ends, the originating presentation resumes playing.

  What "end" means needs to be defined. For example, it could be when the user closes the display window or when a continuous media object ends. This may need to be left up to the profile or even the implementation to define.
- "stop": When the link is traversed, the presentation containing the link stops. That is, it is reset to the beginning of the presentation. The termination of the destination resource will not cause the originating presentation to continue or restart.

The default value of the "sourcePlaystate" attribute depends on the value of the "show" attribute. If the "show" attribute has the value "new", the default for the "sourcePlaystate" attribute is "play". If the "show" attribute has the value "replace" or the deprecated value "pause", then the default for the "sourcePlaystate" attribute is "pause".

destinationPlaystate
: This attribute controls the temporal behavior of the resource identified by the `href` attribute when the link is followed. It only applies when this resource is a continuous media object. It can have the same values as the "sourcePlaystate" attribute.

show
: This attribute specifies how to handle the current state of the presentation at the time in which the link is activated. The following values are allowed:

- "new": The presentation of the destination resource starts in a new context, not affecting the source resource. If both the presentation containing the link and the remote resource contain audio media, both are played in parallel.
- "pause": This value is deprecated in favor of setting the the "show" attribute to "new" and the "sourcePlaystate" attribute to "pause".
- "replace": The current presentation is paused at its current state and is replaced by the destination resource. If the player offers a history mechanism, the source presentation resumes from the state in which it was paused when the user returns to it. The default value of "sourcePlaystate" is "pause" when the "show" attribute has the value "replace".

The default value of "show" is "replace".

accesskey
: This attribute assigns a keyboard key whose activation by the user in turn activates this link. It has the same meaning as the attribute of the same name in HTML 4.0 [HTML40].

tabindex
: This attribute provides the same functionality as the "tabindex" attribute in HTML 4.0 [HTML40]. It specifies the position of the element in the tabbing order for the current document. The tabbing order defines the order in which elements will receive focus when navigated by the user via the keyboard. At any particular point in time, only elements with an active timeline are taken into account for the tabbing order, and inactive elements that are are ignored for the tabbing order.

target
: This attribute defines either the existing display environment in which the link should be opened (e.g. a SMIL region, an HTML frame or another named window), or triggers the creation of a new display environment with the given name. Its value is the identifier of the display environment. If no currently active display environment has this identifier, a new display environment is opened and assigned the identifier of the target. When a presentation uses different types of display environments (e.g. SMIL regions and HTML frames), the namespace for identifiers is shared between these different types of display environments. For example, one cannot use a "target" attribute with the value "foo" twice in a document, and have it point once to an HTML frame, and then to a SMIL region. If the element has both a "show" attribute and a "target" attribute, the "show" attribute is ignored.

actuate
: The actuate attribute determines whether or not the link is triggered by some event or automatically traversed when its time span is active. It is the same as the actuate attribute of XLink [XLINK]. Its default value is "onRequest", which means something must trigger the link traversal. This trigger is defined by the user interaction and event attributes of the <A> element. A value of "onLoad" can also be assigned. This value indicates that the link is automatically traversed when its time span is active.

title
: Defined in the SMIL Metainformation Module.

id
: Standard XML ID attribute, for referential use.

begin
: Defined in SMIL Timing and Synchronization module.

dur
: Defined in "SMIL Timing and Synchronization" module.

end
: Defined in "SMIL Timing and Synchronization" module.

The `<a>` element can also have the attributes listed below, with the same syntax and semantics as in HTML 4.0 [HTML40]:

- onclick
- ondblclick
- onmousedown
- onmouseup
- onmouseover
- onmousemove
- onmouseout
- onkeypress
- onkeydown
- onkeyup
- onfocus
- onblur

Element Content

The <a> element can be empty or contain the following children:

animation
: Defined in section on media object elements.

audio
: Defined in section on media object elements.

img
: Defined in section on media object elements.

par
: Defined in section on par elements.

ref
: Defined in section on media object elements.

seq
: Defined in section on seq elements.

excl
: Defined in section on excl elements.

switch
: Defined in section on switch elements.

text
: Defined in section on media object elements.

textstream
: Defined in section on media object elements.

video
: Defined in section on media object elements.

Examples

Example 1

The link starts up the new presentation replacing the presentation that was playing.

```
<a href="http://www.cwi.nl/somewhereelse.smi">
     <video src="rtsp://foo.com/graph.imf" region="l_window"/>
</a>
```

Example 2

The link starts up the new presentation in addition to the presentation that was playing.

```
<a href="http://www.cwi.nl/somewhereelse.smi" show="new">
     <video src="rtsp://foo.com/graph.imf" region="l_window"/>
</a>
```

This could allow a SMIL player to spawn off an HTML browser:

```
<a href="http://www.cwi.nl/somewebpage.html" show="new">
     <video src="rtsp://foo.com/graph.imf" region="l_window"/>
</a>
```

Example 3

The link starts up the new presentation and pauses the presentation that was playing.

```
<a href="http://www.cwi.nl/somewhereelse.smi" show="new" behavior="pause">
     <video src="rtsp://foo.com/graph.imf" region="l_window"/>
</a>
```

Example 4

The following example contains a link from an element in one presentation A to the middle of another presentation B. This would play presentation B starting from the effective begin of the element with id "next".

```
Presentation A:

     <a href="http://www.cwi.nl/presentationB#next">
       <video src="rtsp://foo.com/graph.imf"/>
     </a>


Presentation B (http://www.cwi.nl/presentation):

      ...
      <seq>
        <video src="rtsp://foo.com/graph.imf"/>
        <par>
          <video src="rtsp://foo.com/timbl.rm" region="l_window"/>
          <video id="next" src="rtsp://foo.com/v1.rm" region="r_window"/>
                 ^^^^^^^^^
          <text src="rtsp://foo.com/caption1.html" region="l_2_title"/>
```

```
      <text src="rtsp://foo.com/caption2.rtx" region="r_2_title"/>
    </par>
  </seq>
...
```

6.3.3 The <area> Element

The functionality of the <a> element is restricted in that it only allows associating a link with a complete media object. The HTML 4.0 "area" element [HTML40] has demonstrated that it is useful to associate links with spatial portions of an object's visual display.

The semantics of the <area> element in SMIL is the same as it is for HTML in that:

1. The <area> element allows associating a link destination, specified by the 'href' attribute, with spatial portions of a visual object. In contrast, the <a> element only allows associating a link with complete media.
2. The <area> element allows making a subpart of the media object the destination of a link, using the "id" attribute.
3. The <area> element allows breaking up an object into spatial subparts, using the "coords" attribute.

It extends the syntax and semantics of the HTML <area> element by providing for linking from non-spatial portions of the media object's display. These extensions are:

1. The area element allows breaking up an object into temporal subparts, using the "begin" and "end" attributes. The values of the begin and end attributes are relative to the beginning of the media object.
2. The area element allows breaking up an XML-defined object into syntactic components, using the "fragment" attribute. The spatial and temporal portion of the display that activates the link is defined in terms of the syntactic structure of that object. This allows portions of the display of XML code integrated in a SMIL presentation to be starting areas for links in SMIL. An example is having an HTML file format the text for a menu of items. These are displayed as part of a SMIL presentation. Each item can be clicked upon to activate a link in the containing SMIL presentation.

The <anchor> element is deprecated in favor of <area>.

Attributes

The <area> element can have the attributes listed below, with the same syntax and semantics as in HTML 4.0 [HTML40]:

- id
- class
- style
- title

- lang
- dir
- onclick
- ondblclick
- onmousedown
- onmouseup
- onmouseover
- onmousemove
- onmouseout
- onkeypress
- onkeydown
- onkeyup
- shape
- coords
- href
- nohref
- alt
- accesskey
- onfocus
- onblur

The following lists attributes that are newly introduced by this specification, and attributes that are extended with respect to HTML 4.0 [HTML40]:

begin
: Defined in SMIL Timing and Synchronization module.

sourcePlaystate
: Defined in Section on the `<a>` element.

show
: Defined in Section on the `<a>` element.

coords
: This attribute is extended to be identical with the "coords" attribute in HTML 4.0. That is, it can take the values needed when the "shape" attribute has the value "circle" or "poly".

dur
: Defined in "SMIL Timing and Synchronization" module.

end
: Defined in "SMIL Timing and Synchronization" module.

sourceVolume
: Defined in Section on the `<a>` element.

destinationVolume
: Defined in Section on the `<a>` element.

destinationPlaystate
: Defined in Section on the `<a>` element.

fragment
: This is a media-specific reference to a portion of the media referenced in the "src" attribute of the parent media object. This attribute is used to place a hotspot in a media file that refers back to the containing SMIL presentation. In order for the "fragment" attribute to be used, the media object integrated with the parent media object element must be addressable by the fragment. If the referenced media object is an XML file, then the value of the "fragment" attribute is a fragment identifier, the part that comes after a # in a URI

Editor Note: This functionality is preliminary. The intent of the fragment attribute is to enable linking from an embedded document back into the main SMIL presentation. Several open issues: What mechanism does the player use to insert the link into the embedded document, and what semantics must be adhered to? How does this affect the DOM event flow? What is the interaction with the "coords" attribute?

show
: Defined in Section on the `<a>` element.

tabindex
: Defined in Section on the `<a>` element.

target
: Defined in Section on the `<a>` element.

title
: Defined in Section on the `<a>` element.

Element Content

The `<area>` element is empty.

Examples

1) Decomposing a video into temporal segments

In the following example, the temporal structure of an interview in a newscast (camera shot on interviewer asking a question followed by shot on interviewed person answering) is exposed by fragmentation:

```
<smil>
  <body>
    <video src="video" title="Tom Cruise interview 1995" >
      <seq>
        <area id="firstQ" dur="20s" title="first question" />
        <area id="firstA" dur="50s" title="first answer" />
      </seq>
    </video>
  </body>
</smil>
```

2) Associating links with spatial segments In the following example, the screen space taken up by a video clip is split into two sections. A different link is associated with each of these sections.

```
<smil>
  <body>
    <video src="video" title="Tom Cruise interview 1995" >
      <area shape="rect" coords="5,5,50,50"
              title="Journalist" href="http://www.cnn.com"/>
      <area shape="rect" coords="5,60,50,50"
              title="Tom Cruise" href="http://www.brando.com" />
   </video>
  </body>
</smil>
```

3) Associating links with temporal segments

In the following example, the duration of a video clip is split into two sub-intervals. A different link is associated with each of these sub-intervals.

```
<smil>
  <body>
    <video src="video" title="Tom Cruise interview 1995" >
      <seq>
        <area dur="20s" title="first question"
              href="http://www.cnn.com"/>
        <area dur="50s" title="first answer"
              href="http://www.brando.com"/>
      </seq>
   </video>
  </body>
</smil>
```

4) Associating links with spatial subparts

In the following example, the screen space taken up by a video clip is split into two sections. A different link is associated with each of these sections.

```
<video src="http://www.w3.org/CoolStuff">
  <area href="http://www.w3.org/AudioVideo" coords="0%,0%,50%,50%"/>
  <area href="http://www.w3.org/Style"      coords="50%,50%,100%,100%"/>
</video>
```

5) Associating links with temporal subparts

In the following example, the duration of a video clip is split into two subintervals. A different link is associated with each of these subintervals.

```
<video src="http://www.w3.org/CoolStuff">
  <area href="http://www.w3.org/AudioVideo" begin="0s" end="5s"/>
  <area href="http://www.w3.org/Style"      begin="5s" end="10s"/>
</video>
```

6) Jumping to a subpart of an object

The following example contains a link from an element in one presentation A to the middle of a video object contained in another presentation B. This would play presentation B starting from second 5 in the video. That is, the presentation would start as if the user had fast-forwarded the whole presentation to the point at which the designated fragment in the "CoolStuff" video begins.

```
Presentation A:

<a href="http://www.cwi.nl/mm/presentationB#tim">
   <video id="graph" src="rtsp://foo.com/graph.imf" region="l_window"/>
</a>


Presentation B:

<video src="http://www.w3.org/CoolStuff">
  <area id="joe" begin="0s" end="5s"/>
  <area id="tim" begin="5s" end="10s"/>
</video>
```

7) Combining different uses of links

The following example shows how the different uses of associated links can be used in combination.

```
Presentation A:

<a href="http://www.cwi.nl/mm/presentationB#tim">
  <video id="graph" src="rtsp://foo.com/graph.imf" region="l_window"/>
</a>


Presentation B:

<video src="http://www.w3.org/CoolStuff">
  <area id="joe" begin="0s" end="5s" coords="0%,0%,50%,50%"
          href="http://www.w3.org/"/>
  <area id="tim" begin="5s" end="10s" coords="0%,0%,50%,50%"
          href="http://www.w3.org/Tim"/>
</video>
```

8) Associating links with syntactic subparts

Below is an example with an integrated HTML file that displays a menu of

```
link one
link two
```

The user can click on one of the menu items, and the matching HTML file is displayed. That is, if user clicks on "link one", the "Link1.html" file is displayed in the "LinkText" region.

The menu HTML file contains the code:

```
<A NAME="link1">link one</A><BR>
<A NAME="link2">link two</A>
```

The SMIL file is:

```
<smil>
  <head>
    <layout>
      <region id="HTML"     width="100" height="100"/>
      <region id="LinkText" width="100" top   ="100"/>
```

```
    </layout>
  </head>
  <body>
    <par>
      <text region="HTML" src="namedanchs.html" dur="indefinite">
        <area fragment="link1" href="#LinkOne"/>
        <area fragment="link2" href="#LinkTwo"/>
      </text>
      <excl -- or something like 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 Media Object Module

Editors

Philipp Hoschka (ph@w3.org), W3C
Rob Lanphier (robla@real.com), RealNetworks

7.1 Introduction

This section defines the SMIL media object module. This module contains elements and attributes used to describe media objects. Since these elements and attributes are defined in a module, 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.

7.2 The `ref`, `animation`, `audio`, `img`, `video`, `text` and `textstream` elements

The media object elements allow the inclusion of media objects into a SMIL presentation. Media objects are included by reference (using a URI).

There are two types of media objects: media objects with an intrinsic duration (e.g. video, audio) (also called "continuous media"), and media objects without intrinsic duration (e.g. text, image) (also called "discrete media").

Anchors and links can be attached to visual media objects, i.e. media objects rendered on a visual abstract rendering surface.

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.

7.2.1 Element Attributes

Languages implementing the SMIL Media Object Module must define which attributes may be attached to media object elements. In all languages implementing the SMIL Media Object Module, media object elements can have the following attributes:

abstract

A brief description of the content contained in the element.

alt

For user agents that cannot display a particular media-object, this attribute specifies alternate text. It is strongly recommended that all media object elements have an "alt" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

If the content of these attributes is read by a screen-reader, the presentation should be paused while the text is read out, and resumed afterwards.

author

The name of the author of the content contained in the element.

begin

Defined in SMIL Timing Module

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.
Values in the clipBegin attribute have the following syntax:

```
Clip-value        ::= [ Metric ] "=" ( Clock-val | Smpte-val ) |
                      "marker" "=" name-val
Metric            ::= Smpte-type | "npt"
Smpte-type        ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val         ::= Hours ":" Minutes ":" Seconds
                      [ ":" Frames [ "." Subframes ]]
Hours             ::= Digit Digit
                  /* see XML 1.0 for a definition of ´Digit´*/
Minutes           ::= Digit Digit
Seconds           ::= Digit Digit
Frames            ::= Digit Digit
Subframes         ::= Digit Digit
name-val          ::= ([^<&"] | [^<&´])*
                  /* Derived from BNF rule [10] in [XML10]
                     Whether single or double quotes are
                     allowed in a name value depends on which
                     type of quotes is used to quote the
                     clip attribute value */
```

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 with 29.97 frames per second. 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.

The time value has the format hours:minutes:seconds:frames.subframes. If the frame value is zero, it may be omitted. Subframes are measured in one-hundredth 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
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 string.

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" />
```

"clipBegin" may also be expressed as "clip-begin" for compatibility with SMIL 1.0. Software supporting the SMIL Boston Language Profile must be able to handle both "clipBegin" and "clip-begin", whereas software supporting only the SMIL media object 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". When used in conjunction with the timing attributes from the SMIL Timing Module, this attribute is applied before any SMIL Timing Module attributes.

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 (such as audio, video or another presentation) that should be played. 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 Boston 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.

copyright

The copyright notice of the content contained in the element.

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. When the media object has associated hyperlinked content, this attribute should provide information about the hyperlinked content.

If the content of these attributes is read by a screen-reader, the presentation should be paused while the text is read out, and resumed afterwards.

port

This provides the RTP/RTCP port for a media object transferred via multicast. It is specified as a range, e.g., port="3456-3457" (this is different from "port" in SDP, where the second port is derived by an algorithm). Note: For transports based on UDP in IPv4, the value should be in the range 1024 to 65535 inclusive. For RTP compliance it should start with an even number. For applications where hierarchically encoded streams are being sent to a unicast address, this may be necessary to specify multiple port pairs. Thus, the range of this request may contain greater than two ports. This attribute is only interpreted if the media object is transferred via RTP and without using RTSP.

readIndex

This attribute specifies the position of the current element in the order in which `longdesc` and `alt` text are read out by a screen reader for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros. The default value is 0.

Elements that contain `alt` or `longdesc` attributes are read by a screen reader 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.

rtpformat

This field has the same semantics as the "fmt list" sub-field in an SDP media description. It contains a list of media format payload IDs. For audio and video, these will normally be a media payload type as defined in the RTP Audio/Video Profile (RFC 1890). When a list of payload formats is given, this implies that all of these formats may be used in the session, but the first of these formats is the default format for the session. For media payload types not explicitly defined as static types, the rtpmap element may be used to provide a dynamic binding of media encoding to RTP payload type. The encoding names in the RTP AV Profile do not specify a complete set of parameters for decoding the audio encodings (in terms of clock rate and number of audio channels), and so they are not used directly in this field. Instead, the payload type number should be used to specify the format for static payload types and the payload type number along with additional encoding information should be used for dynamically allocated payload types. This attribute is only interpreted if the media object is transferred via RTP.

src

The value of the src attribute is the URI of the media object.

stripRepeat

Strip the intrinsic repeat value of the underlying media object. For example, many animated GIFs intrinsically repeat indefinitely. The stripRepeat attribute 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 stripRepeat 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`. Values: "true" or "false". Default: "false".

title

This 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). It is strongly recommended that all media object elements have a "title" attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

transport

This attribute has the same syntax and semantics as the "transport" sub-field in a SDP media description. It defines the transport protocol that is used to deliver the media streams. Currently defined values for this are: "src-attr" and "RTP/AVP", but alternate values may be defined by IANA. The default value for this is "src-attr", which indicates that the transport is derived from the URL given in the `src` attribute. The other defined value for this field is "RTP/AVP". RTP/AVP is the IETF's Realtime Transport Protocol using the Audio/Video profile carried over UDP. The complete definition of RTP/AVP can be found in [RFC1890].

@@ this may be better to derive from the "src" parameter, which could optionally be rtp://___. This would mean that an RTP URL format would need to be defined.

@@ what does it mean when an HTTP URL is coupled with `transport="RTP/AVP"`?

type

Content type of the media object referenced by the "src" attribute. This value takes precedence over other possible sources of the media type (for instance, the "Content-type" field in an HTTP or RTSP 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.

(@@ need to think through if this is what we really want to say; current SMIL implementations probably do not implement this behavior, though it's hard to imagine current implementations *rely* on any other behavior.).

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)

7.2.2 Element Content

Languages utilizing the SMIL Media Object Module must define the complete set of elements which may act as children of media object elements. In all languages implementing the SMIL Media Object Module, the following elements must be part of the supported set of child elements:

- param
- rtpmap

7.2.3 Media object initialization: the `param` element

@@Links below incorrectly point to the HTML spec

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 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 [HTML40] for more information.

Element Description

`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.

Example(s):

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.facethingy.com/herbert.face">
  <param name="mood" value="surly" valuetype="data">
  <param name="accessories" value="baseball-cap,nose-ring" valuetype="data">
</ref>
```

Example(s):

In the following example, run-time data for the object's "Init_values" parameter is specified as an external resource (a GIF file). The value of the `valuetype` attribute is thus set to "ref" and the `value` is a URI designating the resource.

```
<ref classid="http://www.gifstuff.com/gifapplication">
  <param name="Init_values"
         value="./images/elvis.gif">
         valuetype="ref">
</ref>
```

7.2.4 The `rtpmap` element

If the media object is transferred using the RTP protocol, and uses a dynamic payload type, SDP requires the use of the "rtpmap" attribute field. In this specification, this is mapped onto the "rtpmap" element, which is contained in the content of the media object element. If the media object is not transferred using RTP, this element is ignored.

Attributes

payload

The value of this attribute is a payload format type number listed in the parent element’s "rtpformat" attribute. This is used to map dynamic payload types onto definitions of specific encoding types and necessary parameters.

encoding

This attribute encodes parameters needed to decode the dynamic payload type. The attribute values have the following syntax:

```
encoding-val    ::= ( short-encoding | long-encoding )
short-encoding  ::= encoding-name "/" clock-rate
long-encoding   ::= encoding-name "/" clock-rate "/" encoding-params
encoding-name   ::= name-val
clock-rate      ::= +Digit
encoding-params ::= ??
```

Legal values for "encoding-name" are payload names defined in [RFC1890], and RTP payload names registered as MIME types [draft-ietf-avt-rtp-mime-00].

For audio streams, "encoding parameters" may specify the number of audio channels. This parameter may be omitted if the number of channels is one provided no additional parameters are needed. For video streams, no encoding parameters are currently specified. Additional parameters may be defined in the future, but codec specific parameters should not be added, but defined as separate rtpmap attributes.

Element Content

"rtpmap" is an empty element

Example

```
<audio src="rtsp://www.w3.org/foo.rtp" port="49170"
       transport="RTP/AVP" rtpformat="96,97,98">
  <rtpmap payload="96" encoding="L8/8000" />
  <rtpmap payload="97" encoding="L16/8000" />
  <rtpmap payload="98" encoding="L16/11025/2" />
</audio>
```

7.3 Support for media player extensions

A media object referenced by a media object element is often rendered by software modules referred to as media players that are separate from the software module providing the synchronization between different media objects in a presentation (referred to as synchronization engine).

Media players generally support varying levels of control, depending on the constraints of the underlying renderer as well as media delivery, streaming etc. This specification defines 4 levels of support, allowing for increasingly tight integration, and broader functionality. The details of the interface will be presented in a separate document.

Level 0
: Must allow the synchronization engine to query for duration, and must support cue, start and stop on the player. To support reasonable resynchronization, the media player must provide pause/unpause controls with minimal latency. This is the minimum level of support defined.

Level 1
: In addition to all Level 0 support, the media player can detect when sync has been broken, so that a resynchronization event can be fired. A media player that cannot support Level 1 functionality is responsible to maintain proper synchronization in all circumstances, and has no remedy if it cannot (Level 1 support is recommended).

Level 2
: In addition to all Level 1 support, the media player supports a tick() method for advancing the timeline in strict sync with the document timeline. This is generally appropriate to animation renderers that are not tightly bound to media delivery constraints.

Level 3
: In addition to all Level 2 support, the media player also supports a query interface to provide information about its time-related capabilities. Capabilities include things like canRepeat, canPlayBackwards, canPlayVariable, canHold, etc. This is mostly for future extension of the timing functionality and for optimization of media playback/rendering.

7.3.1 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 Boston 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 Boston 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="0s" clip-end="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 Boston 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 must be the URI of this specification, i.e. @@ http://www.w3.org/AudioVideo/Group/Media/extended-media-object19990707
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:

```
<switch>
  <audio src="radio.wav" clipBegin="marker=song1" clipEnd="marker=moderator1"
   system-required=
     "@@http://www.w3.org/AudioVideo/Group/Media/extended-media-object19990707" />
  <audio src="radio.wav" clip-begin="0s" clip-end="3:50" />
</switch>
```

alt, longdesc

Added the recommendation that if the content of these attributes is read by a screen-reader, the presentation should be paused while the text is read out, and resumed afterwards.

New Accessibility Attributes

readIndex

SDP Attributes

When using SMIL in conjunction with the Real Time Transport Protocol (RTP, [RFC1889]), which is designed for real-time delivery of media streams, a media client is required to have initialization parameters in order to interpret the RTP data. In the typical RTP implementation, these initialization parameters are described in the Session Description Protocol (SDP, [RFC2327]). The SDP description can be delivered in the DESCRIBE portion of the Real Time Streaming Protocol (RTSP, [RFC2326]), or can be delivered as a file via HTTP.

Since SMIL provides a media description language which often references SDP via RTSP and can also reference SDP files via HTTP, a very useful optimization can be realized by merging parameters typically delivered via SDP into the SMIL document. Since retrieving a SMIL document constitutes one round trip, and retrieving the SDP descriptions referenced in the SMIL document constitutes another round trip, merging the media description into the SMIL document itself can save a round trip in a typical media exchange. This round-trip savings can result in a noticeably faster start-up over a slow network link.

This applies particularly well to two primary usage scenarios:

- Pure multicast implementations. This is traditional IETF model where the SDP is sent via some other transport protocol such as SAP, HTTP, or via email.
- RTSP delivery. In this case, the primary value of the SDP description is in the description of media headers delivered in the RTSP DESCRIBE phase, and not in the transport specification. The transport information (such port number negotiation and multicast addresses) is handled in RTSP separately in the SETUP phase.

The following attributes were added to SMIL Boston:

port
rtpformat
transport

Example

```
<audio src="rtsp://www.w3.org/test.rtp" port="49170-49171"
       transport="RTP/AVP" rtpformat="96,97,98" />
```

In addition to these new attributes, the "rtpmap" element was added to complete the SDP functionality.

stripRepeat

The `stripRepeat` attribute was added to provide better timing control over media with intrinsic repeat behavior (such as animated GIFs).

7.3.2 Appendix B: Element Content

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` and `rtpmap` elements are now allowed as children of a SMIL media object. Other new children may also be defined by the host language.

7.3.3 Appendix C: New sections

The `param` element

A new section describing the "param" element provides a generalized mechanism to attach media-specific attributes to media objects.

The `rtpmap` element

A new section describing the "rtpmap" element provides functionality needed to use SMIL as a replacement for SDP.

Support for media player extensions

SMIL Boston introduces the concepts of levels of functionality, which are explained in this section.

7.3.4 Appendix D: Backburner

Listed below are the features that haven't been integrated yet, and may not make it into the final version of SMIL Boston:

- XLink-conformance
- HTML OBJECT tag syntax

8. The SMIL Metadata Module

Editors:
Thierry Michel (tmichel@w3.org), W3C

8.1 Introduction

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 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 Metadata module defined in this specification fully supports the use 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 metadata language for providing information about resources.

8.2 Compatibility with SMIL 1.0 using the meta Element

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.
Each <meta> element specifies a single property/value pair in the name and content attributes, respectively.

8.2.1 Element Attributes

The "meta" element can have the following attributes:

content
: This attribute specifies the value of the property defined in the meta element. The "content" attribute is required for "meta" elements.

id
: This attribute uniquely identifies an element within a document. Its value is an XML identifier.

name
: This attribute identifies the property defined in the meta element.
The "name" attribute is required for "meta" elements.

skip-content
: This attribute is introduced for future extensibility of SMIL. It is interpreted in the following two cases:

- If a new element is introduced in a future version of SMIL, and this element allows SMIL 1.0 elements as element content, the "skip-content" attribute controls whether this content is processed by a SMIL 1.0 player.
- If an empty element in SMIL version 1.0 becomes non-empty in a future SMIL version, the "skip-content" attribute controls whether this content is ignored by a SMIL 1.0 player, 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".

The list of properties 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.

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 contains the title of the presentation.

This specification extends the SMIL 1.0 "meta" element with the following attributes:

[I18N] xml:lang
: This attribute specifies the language used.

http-equiv
: This attribute may be used in place of the name attribute. HTTP servers use this attribute to gather information for HTTP response message headers.

scheme
: This attribute names a scheme to be used to interpret the property's value.

8.2.2 Element Content

The "meta" element is an empty element.

8.3 Extensions to SMIL 1.0 Metadata.

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 with a SMIL documents, 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], a W3C Recommendation 22 February 1999
- Resource Description Framework (RDF) Schema [RDFschema], a W3C Proposed Recommendation 03 March 1999

Metadata within an SMIL document should be expressed in the appropriate RDF namespaces [XML-NS] and should be placed within the **<metadata>** child element to the document's **<smil>** root element. (See example below.) 126

8.3.1 Element Attributes

The "metadata" element can have the following attributes:

id

This attribute uniquely identifies an element within a document. Its value is an XML identifier.

8.3.2 Element Content

The "metadata" element can contain the following child elements:

<RDF> element and its sub-elements.

8.3.3 Using multiple description schemes simultaneously

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 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, etc.).
- Content creators should refer to the Video Metadata Representation, in "A

Comparison of Schemas for Dublin Core-based Video Metadata Representation", which extends Dublin Core properties to cope with video content metadata requirements (e.g., Type, Relation, Format, Coverage, etc.).
- Additionally, the SMIL Metadata Schema (below) contains a set of additional metadata properties that are common across most uses of multimedia. . 126

Individual industries or individual content creators are free to define their own metadata schema, but everyone is encouraged to follow existing metadata standards and use standard metadata schema wherever possible to promote interchange and interoperability. If a particular standard metadata schema does not meet your needs, then it is usually better to define an additional metadata schema in RDF that is used in combination with the given standard metadata schema than to totally avoid the standard schema.

8.4 The SMIL Metadata Schema

(This schema has not yet been defined. Here are some candidate attributes for the schema: LevelAccessibilityGuidelines, ListOfImagesUsed, ListOfAudioUsed, ListOfTextUsed, ListOfTextstreamUsed, ListOfRefUsed, ListOfCodecUsed, etc)

8.5 An Example

Here is an example of how metadata can be included in an SMIL document. The example uses the Dublin Core version 1.0 Schema [DC] and the SMIL Metadata Schema: 126

```
<?xml version="1.0" ?>
<smil xmlns = "http://www.w3.org/TR/.../SMIL-Boston.dtd">
 <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" />

  <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.w3.org/AudioVideo/.../smil-ns#" >

<!-- Metadata about the SMIL presentation -->
   <rdf:Description about="http://www.foo.com/meta.smi"
       dc:Title="An Introduction to the Resource Description Framework"
       dc:Description="The Resource Description Framework (RDF) enables the encoding, exchange and reuse of structured metadata"
       dc:Publisher="W3C"
       dc:Date="1999-10-12"
       dc:Rights="Copyright 1999 John Smith"
       dc:Format="text/smil" >
       <dc:Creator>
          <rdf:Seq ID="CreatorsAlphabeticalBySurname">
             <rdf:li>Mary Andrew</rdf:li>
             <rdf:li>Jacky Crystal</rdf:li>
          </rdf:Seq>
       </dc:Creator>
       <smilmetadata:ListOfVideoUsed>
          <rdf:Seq ID="VideoAlphabeticalByFormatname">
              <rdf:li Resource="http://www.foo.com/videos/meta-1999.mpg"/>
              <rdf:li Resource="http://www.foo.com/videos/meta2-1999.mpg"/>
          </rdf:Seq>
       </smilmetadata:ListOfVideoUsed>
       <smilmetadata:Access LevelAccessibilityGuidelines="AAA"/>
   </rdf:Description>

<!-- Metadata about the video -->
   <rdf:Description about="http://www.foo.com/videos/meta-1999.mpg"
        dc:Title="RDF part one"
        dc:Creator="John Smith"
        dc:Subject="Metadata,RDF"
        dc:Description="RDF basic fonctionalities"
```

```
        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.foo.com/videos/meta-1999.mpg#scene1"/>
             <rdf:li Resource="http://www.foo.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 fonctionalities"
        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="0" end ="30"/>
     <area id="scene2" begin="30" end ="60"/>
   </video>
   <video region="a" src="/videos/meta2-1999.mpg"/>
 </seq>
 </body>
</smil>
```

Note:Validate the above RDF description with SiRPAC; a Simple RDF Parser and Compiler, written by Janne Saarela (W3C).

9. SMIL Structure Module

Editors

Warner ten Kate (warner.ten.kate@philips.com), (Philips Electronics)

9.1 Introduction

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 SMIL-family document types. The Structure Module is a mandatory module in a profile building a member of the SMIL profile family. The Structure Module is isomorphic with the XHTML Structure Module [XMOD].

The SMIL Structure Module is composed out of the smil, head, and body element, 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, head and body elements

This section is Informative.

The attributes and content model of the Structure Module elements is summarized in the following table:

The Elements with their Attributes and Content Model for the SMIL Structure Module.

Elements	Attributes	Minimal Content Model
smil	Core, Accessibility, xmlns	head?, body?
head	Core, Accessibility, profile	meta*, (switch | layout)?
body	Core, Accessibility	(Schedule | MediaContent | MediaControl | LinkAnchor)*
-	skipContent	N/A

The Attribute collections in this table are defined as follows

Core

id (ID), class (NMTOKEN)

Accessibility
: xml:lang (NMTOKEN), title (CDATA)

The collections in the table from the Content Model of the body element are defined as follows

Schedule
: par, seq, excl [Timing and Synchronization Module]

MediaContent
: ref, audio, video, img, animation, text, and textstream [Media Object Module]

MediaControl
: switch [Content Control Module]

LinkAnchor
: a, area [Linking Module]

@@ check on completeness and correctness in final version.

The smil element acts as the root element for all Document Types of the SMIL-Family.

The head element contains information that is not related to the temporal behavior of the presentation.

The body element contains information that is related to the temporal and linking behavior of the document. It acts as the root element to span the timing tree.

The body element has the schedule semantics of a timecontainer equal to that of the seq element [Timing and Synchronization Module]. Note, that in other profiles, where a body element from another (Structure) Module is in use, that body element may have different schedule semantics. For example, in the HTML+SMIL profile , the body element takes the semantics of the par element.

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

The class attribute assigns a class name or 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.

The xml:lang attribute specifies the language of an element, and is specified in XML 1.0 [XML10].

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).

The xmlns attribute declares an XML namespace, and is defined in "Namespaces in XML" [XML-NS].

The profile attribute specifies the profile to which the current document's Document Type conforms.

The skipContent attribute is specified in SMIL 1.0 [SMIL10]. The syntax notation "skip-content" has been deprecated, in favor of "skipContent". skipContent is interpreted in the following two cases:

- If a new element is introduced in a future version of SMIL, and this element allows SMIL 1.0 elements as element content, the skipContent attribute controls whether this content is processed by a SMIL 1.0 player.
- If an empty element in SMIL version 1.0 becomes non-empty in a future SMIL version, the skipContent attribute controls whether this content is ignored by a SMIL 1.0 player, or results in a syntax error.

If the value of the skipContent 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 skipContent is "true".

9.3 Integrating the SMIL Structure Module

This section is Normative.

The SMIL Structure Module is the starting module when building any profile in the SMIL-family. The Structure Module cannot be used for building other, non SMIL-family, profiles. To be called a member of the SMIL-family the profile should at least include the following modules

- SMIL Structure Module
- Timing and Synchronization Module
- Media Object Module

@@ This should probably go elsewhere (Modules Module?).

This means that the SMIL Structure Module must at least be accompanied with the above two other modules. (Those modules can still be used in other, non SMIL-family, profiles.)

The integration of the SMIL Structure Module with other SMIL modules should conform to the descriptions in the SMIL-Boston profile .

When non-SMIL modules are integrated in the 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). With respect to the SMIL Structure module, the Profiling Entities in the DTD need to be overridden. This realizes 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].

9.4 DTD

This section is Normative.

This section specifies the DTD of the SMIL Structure Module.

@@ Check for harmonizing with [XMOD] when that receives REC status.

@@ Update the events naming with XML-DOM and SMIL-DOM.

@@ The xml:base attribute needs to be added, awaiting XLink resolutions. This also requires adaptation in the meta Module. Note, that XHTML knows a separate Base Module.

@@ How to add "skipContent"?

```
<!-- ================================================================= -->
<!-- SMIL Structure Module  ========================================== -->
<!-- file: SMIL-struct.mod

     This is Smil-Boston.
     Copyright 1999 W3C (MIT, INRIA, Keio), All Rights Reserved.

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

     PUBLIC "-//W3C//ELEMENTS SMIL-Boston Document Structure//EN"
     SYSTEM "SMIL-struct.mod"

     ================================================================= -->


<!-- ================== General entities ========================== -->

<!ENTITY % URI "CDATA" >
<!ENTITY % SMIL.ns "SMIL-Boston.dtd" >
<!ENTITY % SMIL.profile "SMIL-Boston.dtd" >

<!ENTITY % core
        "id ID #IMPLIED
         class NMTOKEN #IMPLIED"
>

<!ENTITY % accessibility
        "xml:lang NMTOKEN #IMPLIED
         title CDATA #IMPLIED"
>

<!ENTITY % xml-dom.events
        "onMouseover CDATA #IMPLIED
         onClick CDATA #IMPLIED
         onEtc CDATA #IMPLIED"
>
<!ENTITY % smil-dom.events
        "onBegin CDATA #IMPLIED
         onEnd CDATA #IMPLIED
         onEtc CDATA #IMPLIED"
>
<!ENTITY % dom.events
        "%smil-dom.events;
         %xml-dom.events;"
>
```

```
<!-- ================== Profiling Entities ======================== -->

<!ENTITY % XSmil.attr "" >

<!ENTITY % XBody.attr "" >
<!ENTITY % XBody.content "" >

<!ENTITY % XHead.attr "" >
<!ENTITY % XHead.content "" >


<!-- ================== SMIL Document Root ======================== -->

<!ELEMENT smil (head?,body?)>
<!ATTLIST smil
        %core;
        %accessibility;
        xmlns %URI; #FIXED "%SMIL.ns;"
        %XSmil.attr;
>


<!-- ================== The Document Head ========================= -->

<!ENTITY % layout-section "layout|switch">
<!ENTITY % Head.content "(meta*,(%layout-section;),meta*,(%XHead.content;),meta*)?">

<!ELEMENT head %Head.content;>
<!ATTLIST head
        %core;
        %accessibility;
        profile %URI; #FIXED "%SMIL.profile;"
        %XHead.attr;
>


<!--=================== The Document Body - Schedule Root ========= -->

<!ENTITY % schedule "par|seq|excl">
<!ENTITY % media-object "audio|video|text|img|animation|textstream|ref">
<!ENTITY % content-control "switch">
<!ENTITY % link "a|area">
<!ENTITY % Body.content "%schedule;|%media-object;|%content-control;|%link;">

<!ELEMENT body (%Body.content;|%XBody.content;)*>
<!ATTLIST body
        %core;
        %accessibility;
        %dom.events;
        %XBody.attr;
        dur                  CDATA              #IMPLIED
        repeatCount          CDATA              #IMPLIED
        repeatDur            CDATA              #IMPLIED
        defaultSyncBehavior  (locked | canSLip)  "canSlip"
        defaultSyncTolerance CDATA              #IMPLIED
>


<!-- end of SMIL-struct.mod -->
```

10. The SMIL Timing and Synchronization Module

Editors:
Patrick Schmitz (pschmitz@microsoft.com), (Microsoft)
Jeff Ayars (jeffa@real.com), (RealNetworks)
Bridie Saccocio (bridie@real.com), (RealNetworks)

10.1 Introduction

SMIL 1.0 solved fundamental media synchronization problems and defined a powerful way of choreographing multimedia content. SMIL Boston extends the timing and synchronization support, adding capabilities to the timing model and associated syntax. This section of the document specifies the Timing and Synchronization module.

There are two intended audiences for this module: implementers of SMIL Boston 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. The semantics of the timing model and of the associated markup must remain consistent across all profiles. Any host language that includes SMIL Boston 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.

Some SMIL 1.0 syntax has been changed or deprecated. Only SMIL *document players* must support the deprecated SMIL 1.0 attribute names as well as the new SMIL Boston names. A SMIL *document player* is an application that supports playback of SMIL Language documents (i.e. documents with the associated MIME type "application/smil").

10.2 Overview of SMIL timing

This section is informative.

SMIL Timing defines elements and attributes to coordinate and synchronize the presentation of *media* over time. The term *media* covers a broad range, including *discrete* media types such as still images, text, and vector graphics, as well as *continuous* media types that are intrinsically time-based, such as video, audio and animation.

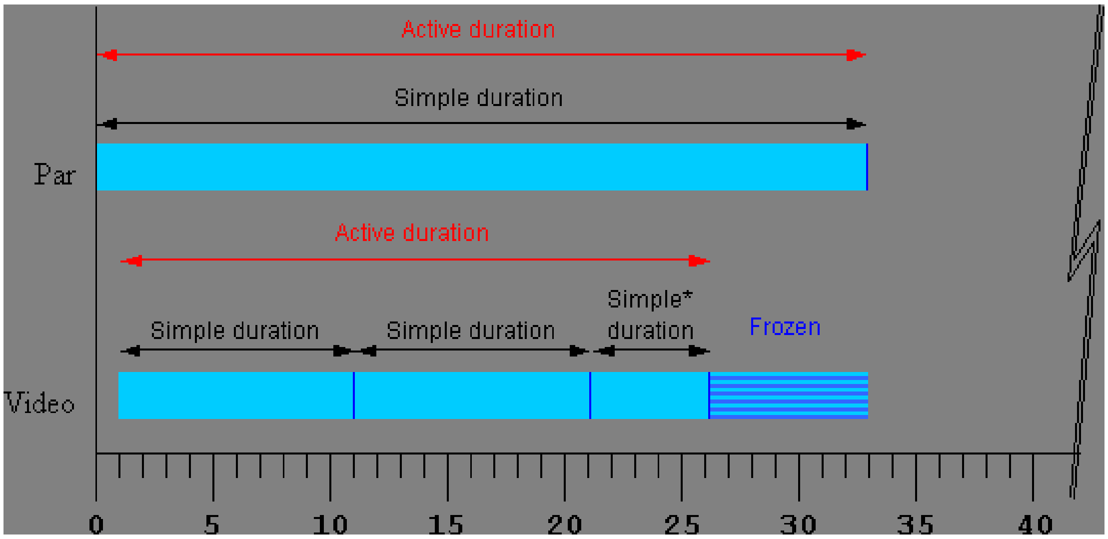
Three synchronization elements support common timing use-cases:

- The <seq> element plays the child elements one after another in a *sequence*.
- The <excl> element plays one child at a time, but does not impose any order.
- The <par> element plays child elements as a group (allowing "parallel" playback).

These elements are referred to as *time containers*. They group their contained children together into coordinated timelines.

SMIL Timing also provides attributes that can be used to specify an element's timing behavior. Elements have a begin, and a *simple duration*. The begin can be specified in various ways - for example, an element can begin at a time, 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.

Figure 1 illustrates the basic support of a repeating element within a simple par time container. The corresponding syntax is included with the diagram.

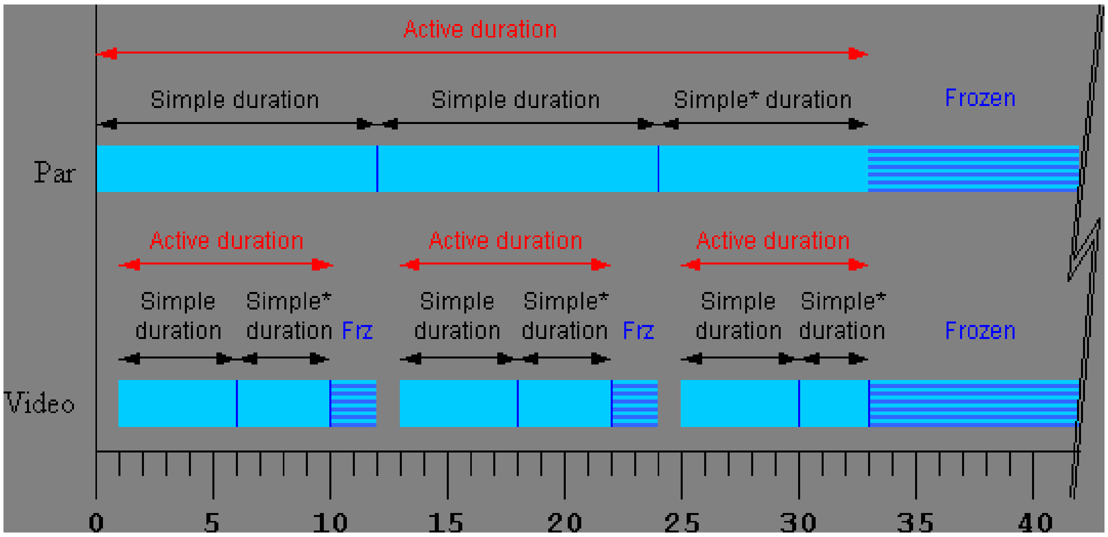



```
<par begin="0s" dur="33s">
   <video begin="1s" dur="10s" repeatCount="2.5" fill="freeze" .../>
</par>
```

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

The attributes that control these aspects of timing can be applied not only to media elements, but to the time containers as well. This allows, for example, an entire sequence to be repeated, and to be coordinated as a unit with other media and time containers. While authors can specify a particular simple duration for a time container, it is often easier to leave the duration unspecified, in which case the simple duration is defined by the contained child elements. When an element does not specify a simple duration, the time model defines an *implicit* simple duration for the element. For example, the implicit simple duration of a sequence is based upon the sum of the active durations of all the children.

Each time container also imposes certain *defaults* and *constraints* upon the contained children. For example in a `<seq>`, elements begin *by default* right after the previous element ends, and in all time containers, the active duration of child elements is *constrained* not to extend past the end of the time container's simple duration. Figure 2 illustrates the effects of a repeating `<par>` time container as it constrains a `<video>` child element.




```
<par begin="0s" dur="12s" repeatDur="33s" fill="freeze" >
   <video begin="1s" dur="5s" repeatCount="1.8" fill="freeze" .../>
</par>
```

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

The SMIL *Timing Model* defines how the time container elements and timing attributes are interpreted to construct a *time graph*. The *time graph* is a model of the presentation schedule and synchronization relationships. 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.

The SMIL Timing and Synchronization syntax and precise semantics are described in the following section. A set of symbols are used in the semantic descriptions:

B
: The time at which an element begins.

d
: The simple duration of an element.

AD
: The active duration of an element. This is the period during which time is actively advancing for the element. This includes any effect of repeating the simple duration, but does not include the time during which the element may be frozen.

AE
: The active end. This is the end of the active duration of an element.

10.3 Language definition

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. The start of the interval in which the document is presented is referred to as the *document begin*. The end of the interval in which the document is presented is referred to as the *document end*. The difference between the end and the begin is referred to as the *document duration*. The formal definitions of presentation and document begin and end are left to the host language designer (see also "Required host language definitions").

10.3.1 Shared timing support

This section defines the set of timing attributes that are common to all of the SMIL synchronization elements.

@@ Need to define "local time" or find a different term.

Basics - begin and dur

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 length of the simple duration is specified using the `dur` attribute. The attribute syntax is described below. The normative syntax rules for each attribute value variant are described below (in Timing Attribute Values); a syntax summary is provided here as an aid to the reader.

begin : smil-1.0-syncbase-value * | begin-value-list | "indefinite"
Defines when the element should begin (i.e. become active).
The attribute value is either a SMIL 1.0 syncbase declaration, a semi-colon separated list of values, or the special value "indefinite".

smil-1.0-syncbase-value * : "id(" id-ref ")" ("(" ("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.
*Note: Only compliant SMIL document players are required to support the SMIL 1.0 syncbase-value syntax. Language designers integrating SMIL Boston Timing and Synchronization into other languages should not support this syntax.

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 below.

"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 Supported Methods section.
Hyperlink-based timing is described in the Hyperlinks and Timing section.

begin-value : (offset-value | syncbase-value | syncToPrev-value | event-value | media-marker-value | wallclock-sync-value)
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 local time on the parent time container.

syncbase-value : (id-ref "." ("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.

syncToPrev-value : ("prev.begin" | "prev.end") (("+" | "-") clock-value)?
Describes a logical syncbase and an offset from that syncbase. The syncbase element is the previous timed sibling element, as reflected in the DOM (or the parent time container if there is no previous sibling). The element begin is defined relative to the begin or active end of the syncbase element. This is equivalent to the default syncbase for children of a `<seq>` time container.

event-value : (id-ref ".")? (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 .

media-marker-value : id-ref ".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].

dur

Specifies the simple duration.

The attribute value can be either of the following:

clock-value

Specifies the length of the simple duration, measured in local time. Value must be greater than 0.

"indefinite"

Specifies the simple duration as indefinite.

Begin value semantics

If no `begin` is specified, the default timing is dependent upon the time container. Children of a `<par>` begin by default when the `<par>` begins (equivalent to `begin="0"`). Children of a `<seq>` begin by default when the previous child ends its active duration (equivalent to `begin="0"`). 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 Restarting elements).

In general, the earliest time in the list determines the begin time of the element. In the case where an element can begin multiple times, the next begin time is the earliest begin time after the current time. There are additional constraints upon the evaluation of the begin time list, detailed in Evaluation of begin and end time lists .

If there is an error in any individual value in the list of begin values, only the individual value will be ignored (as though it were not specified), but the rest of the list will not be invalidated. If no legal value is specified in the list of begin values, the default value for `begin` will be used.

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 know as *timespace conversion*, and is detailed in the section Converting between local and global times .

Dur value semantics

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 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. Note that if a media element has time children (e.g. `animate` or `area` elements), then it is also a `<par>` time container. If the media element has no time 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 typically a function 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 `<seq>`, `<par>` and `<excl>` time containers, including media elements that are also time containers, the implicit simple duration is a function of the children of the time container. For details see the section Time container durations .

If the author specifies a simple duration that is *longer* than the "intrinsic" defined 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 simple duration. This only applies to visual media - aural media will simply stop playing.

Note that when the simple duration is "indefinite", some simple use cases can yield surprising results. See the related example #4 .

Resolving times

Note that when the `begin` attribute refers to an event, or to the begin or active end of another element, it may not be possible to calculate when the begin will happen. For example, if an element is defined to begin on some event, the begin time will not be known until the event happens. When such a time becomes known (i.e. when it can be calculated as a presentation time), the time is said to be *resolved* (see also the discussion of Unifying scheduled and interactive timing).

Examples

The following example shows simple offset begin timing. The `<audio>` element begins 5 seconds after the `<par>` time container begins, and ends 4 seconds later.

```
<par>
   <audio src="song1.au" begin="5s" dur="4s" />
</par>
```

The following example shows syncbase begin timing. The `<img>` element begins 2 seconds after the `<audio>` element begins.

```
<par>
   <audio id="song1" src="song1.au" />
   <img src="img1.jpg" begin="song1.begin+2s" />
</par>
```

Elements can also be specified to begin in response to an event. In this example, the image element begins (appears) when the user clicks on element "show". The image will end (disappear) 3 and a half seconds later.

```
<text id="show" ... />
<img begin="show.click" dur="3.5s" ... />
```

Timing attribute values

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

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
                          | syncToPrev-value | event-value
                          | media-marker-value | wallclock-sync-value)
```

End values

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      ::= (clock-value | syncbase-value
                          | syncToPrev-value | event-value
                          | media-marker-value | wallclock-sync-value)
```

Parsing timing specifiers

Several of the timing specification values have a similar syntax. In addition, XML ID attributes are allowed to contain the dot '`.`' separator character. The backslash character '\' can be used to escape the dot separator within identifier and event-name references. To parse an individual item in a value-list, the following approach defines the correct interpretation.

1. If the value begins with a number or numeric sign indicator (i.e. `'+'` or `'-'`), the value should be parsed as an offset value .
2. Else if the value begins with the token "prev", it should be parsed as a syncToPrev-value .
3. Else if the value begins with the token "wallclock", it should be parsed as a wallclock-sync-value .

4. Else: Build a token substring up to but not including any sign indicator (i.e. strip off any offset). In the following, ignore any '`.`' separator characters preceded by a backslash '`\`' escape 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 string "`.begin`" or "`.end`", then the value should be parsed as a syncbase-value .
    3. Else if the token contains the 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).

@@Note that this approach essentially reserves the following tokens: `prev` and `wallclock` for element IDs, and `begin`, `end` and `marker` for event names.

Clock values

Clock values have the following syntax:

```
Clock-val          ::= ( 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
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

An offset value has the following syntax:

```
offset-value   ::= ( "+" | "-" )?( 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 local time on the parent time container.

The implicit syncbase for an offset value is dependent upon the time container:

- For children of a `<par>` or an `<excl>`, the offset is relative to the begin of the parent `<par>` or `<excl>`.
- For children of a `<seq>`, the offset is relative to the active end of the previous child. If there is no previous child, the offset is relative to the begin of the parent `<seq>`. See also The seq time container .

SMIL 1.0 begin and end values

Note, only compliant SMIL document players are required to support the SMIL 1.0 syncbase-value syntax. Language designers integrating SMIL Boston Timing and Synchronization into other languages should not support this syntax.

```
smil-1-syncbase-value  ::= "id(" id-ref ")"
                           ( "(" ( "begin" | "end" | clock-value) ")" )?
```

ID-Reference values

ID reference values are references to the value of an "id" attribute of another element in the document.

```
Id-value   ::= IDREF
```

The IDREF is a legal XML identifier.

Syncbase values

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"
```

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*. The syncbase element must be another timed element contained in the host document. In addition, the syncbase element may not be a descendent of the current element. If the syncbase element specification refers to an illegal element, the syncbase-value description is ignored (although the entire time value list is not invalidated - only the particular syncbase value).

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 local time on the parent time container. 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
begin="x.begin + 1m"    : End 1 minute after "x" begins
```

Sync To Prev values

A sync-to-prev value has the following syntax:

```
SyncToPrev-value   ::= ( "prev." Time-symbol )
                       ( S  ("+"|"-") S  Clock-value )?
```

A sync-to-prev value is much like a syncbase value, except that the reserved token "prev" is used in place of the Syncbase-element term. The Time-symbol and optional Clock-value offset are as defined for syncbase values .

The *previous element* is the (timed) element that precedes this element within the parent time container (as reflected in the DOM). Note that the parent time container may not be the immediate parent of the current node, in some host documents.

If there is no previous element (i.e. if the current element is the first timed child of the parent time container), then the begin of the parent time container is used as the syncbase (note that the Time-symbol is ignored in this case). The Clock-value offset is nevertheless added to the parent time container begin time, to yield the resulting time value.

@@This requires more complete examples, or we need to include them above somewhere. We need good examples of how this is used.

Examples:

`begin="prev.end-5s"` : Begin 5 seconds before the previous element ends
`begin=" prev.begin "` : Begin when the previous element begins
`begin="prev.begin + 1m"` : End 1 minute after the previous element begins

Event values

An event value has the following syntax:

```
  Event-value       ::= ( Eventbase-element "." )?
Event-symbol
                         ( S  ("+"|"-") S  Clock-value )?
 Eventbase-element ::= Id-value
```

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.
The "Id-value" is the value of an attribute declared to be of type ID (per the XML definition) in the host language, for the event-base element. This 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 attribute is specified (the current element). If this 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). Note that certain elements may specify a different default eventbase. E.g. the SMIL Animation elements (`animate`, `animateMotion`, etc.) specify that the default eventbase is the *target element* of the animation. See also [[SMIL Animation]].

The event value must specify an Event-symbol. This term specifies the name of the event that is raised on the Event-base element. The host language designer must specify which types of events can be used. If an integrating language specifies no supported events, the event-base time value is effectively unsupported for that language.

The last term specifies an optional offset-value that is an offset from the time of the event. The offset is measured in local time on the parent time container. 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.

Note that it is not considered an error to specify an event that cannot be raised on the Event-base element (such as click for audio or other non-visual elements). Since the event will never be raised on the specified element, the event-base value is effectively ignored. Similarly, 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. Host language specifications must include a description of legal event names, and/or allow any name to be used.

The semantics of event-based timing are detailed in the section Unifying Scheduling and Interactive Timing .

Examples:

`begin=" x.load "` : Begin when "load" is observed on "x"
`begin="x.focus+3s"` : Begin 3 seconds after an "focus" event on "x"

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.

```
   Media-Marker-value ::= Id-value ".marker(" S
marker-symbol 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

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]. 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].

```
wallclock-val  ::= "wallclock(" S (DateTime | WallTime)  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 )
```

```
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.

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.

No embedded white space is allowed in wallclock values, although leading and trailing white space characters will be ignored.

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. 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 )"
```

10.3.2 Time manipulations

New element controls for element time behavior are under discussion. Note that an Accessibility requirement for control of the playback speed is related to (but may end up with different syntax different from) the speed control. In general, these time manipulations are suited to animation and non-linear or discrete media, rather than linear continuous media. Not all continuous media types will support time manipulations, e.g. streaming MPEG 1 video playing backwards. A fallback mechanism is described for these cases.

Three new attributes add support for timing manipulations to SMIL Timing, including control over the speed of an element, and support for acceleration and deceleration. The impact on overall timing and synchronization is described. A definition is provided for reasonable fallback mechanisms for media players that cannot support the time manipulations.

Background

A common general application of timing supports animation. The recent integration of SMIL timing with SVG is a good example of the interest in this area. Animation in the more general sense includes the time-based manipulation of basic transforms, applied to a presentation. Some of the effects supported include motion, scaling, rotation, color manipulation, as well as a host of presentation manipulations with a style framework like CSS.

Animation is often used to model basic mechanics. Many animation use-cases are difficult or nearly impossible to describe without a simple means to control pacing and to apply simple effects that emulate common mechanical phenomena. While it is possible to build these mechanisms into the animation behaviors themselves, this requires that every animation duplicate this support. This makes the framework more difficult to extend and customize. In addition, this model allows any animation behavior to introduce individual syntax and semantics for these mechanisms. This makes the authoring model much harder to learn, and complicates the job of any authoring tool designer as well. Finally, this model precludes the use of these mechanisms on structured animations (e.g. a time container with a series of synchronized animation behaviors).

A much simpler model for providing the necessary support centralizes the needed functionality in the timing framework. This allows all timed elements to support this functionality, and provides a consistent model for authors and tools designers. The most direct means to generalize pacing and related functionality is to transform the pacing of *time* for a given element. This is an extension of the transform that is implicitly performed to translate from the general document or presentation time space to the adjusted time space for the element (accounting for the begin time of the element, repeat functionality, etc.). Thus, to control the pacing of a motion animation, a transform is applied that adjusts the pacing of local time for the motion element. If time is scaled to advance faster than normal presentation time, the motion will appear to run faster. Similarly, if the pacing of time is dynamically adjusted, acceleration and deceleration effects are easily obtained. This model is detailed in the sections below.

Overview of support

Three basic time manipulations are proposed:

`speed`
: Controls the pacing (or *speed*) of time. This is the basic manipulation upon which the others are built. The speed effectively scales the rate at which local time plays. As such, speed can modify the effective simple duration.

`accelerate` *and* **`decelerate`**
: Dynamic manipulation of speed to simulate common mechanical motion. Acceleration and deceleration are crucial to motion, rotation, scaling and many other standard transforms. A simple model is presented to allow acceleration from rest at the beginning of the simple duration, and/or deceleration to rest at the end of the simple duration. This model has the advantage that it preserves the simple duration. The model is sometimes presented to authors as *"Ease-In, Ease-Out"*.

`autoReverse`
: Another very common mechanical phenomenon is that of a process that advances and reverses. Some examples include:
  - pendulum motion - a partial rotation that advances and reverses
  - pulsing effects - usually a scale transform, but sometimes an intensity or color change that advances and reverses
  - simple bouncing - motion that advances and reverses

  This support is often represented to authors as *"Play Forwards, then Backwards"*. Because so many common use-cases apply repeat to the modified local time (as in the examples above), this function is modeled as modifying the simple duration. As such, `autoReverse` effectively doubles the simple duration.

When the three features are combined, there is an inherent ordering that can be applied. The accelerate and decelerate features are applied locally on the simple duration, and have no side effects upon the active duration of the element. The autoReverse feature is applied to the simple duration, and doubles it. Thus, autoReverse *wraps* the effect of accelerate and decelerate. Speed has the broadest effect, scaling the progress of local time for the element. Taken from the perspective of a conversion from the document time-space to the local time-space, speed is applied earliest, autoReverse later and and then accelerate and decelerate are applied latest. See also Details of the time manipulations .

Examples

The following motion animation will move the target twice as fast as normal:

```
<animateMotion dur="10s" repeatCount="2" speed="2.0" path= ... />
```

The target will move over the path in 5 seconds, and then repeat this motion. The active duration is thus 10 seconds.

The following rotation (a theoretical extension to the animation platform) will produce a simple pendulum swing on the target (assume that it is a pendulum shape with the transform origin at the top):

```
<animateRotate from="20deg" to="-20deg" dur="1s" repeatCount="indefinite"
       accelerate=".5" decelerate=".5" autoReverse="true" ... />
```

The pendulum swings through an arc in one second, and then back again in a second. It repeats indefinitely. The acceleration and deceleration are specified as a proportion of the simple duration (before autoReverse). As specified, the effect is to accelerate all the way through the downswing, and then decelerate all through the upswing. This produces a very realistic looking animation of real-world pendulum motion. The `rotate` element itself can be very simple, for example interpolating the rotation value in a transform matrix.

Attribute syntax

The `speed` attribute is supported on all timed elements. The argument value expresses a multiple of normal play speed that will be applied to the element and all time descendents. Thus 1.0 is normal speed, and `speed="1"` is a no-op, and `speed="-1"` means play backwards.

speed attribute

The speed attribute controls the local playback speed of an element, to speed up or slow down the effective rate of play. Note that the speed does not specify an absolute play speed, but rather is relative to the playback speed of the parent time container. Thus if a <par> and one of its children both specify a speed of 50%, the child will play at 25% of normal playback speed .

speed
: Defines the playback speed of element time. The value is specified as a multiple of normal (parent time container) play speed.

Legal values are signed floating point values. A value of 0 is not allowed.
The default is "1.0" (no modification of speed).

The details of the speed modification are described in Details of the time manipulations .

accelerate and decelerate attributes

These attributes define a simple acceleration and deceleration of element time, within the simple duration. This is useful for animation, motion paths, etc. The values are expressed as a proportion of the simple duration (i.e. between 0 and 1), and are defined such that the simple duration is not affected (although the normal play speed is increased to compensate for the periods of acceleration and deceleration). Note that these attributes apply to the simple duration; if these attributes are combined with repeating behavior, the acceleration and/or deceleration occurs within each repeat iteration.

The sum of `accelerate` and `decelerate` must not exceed 1. If it does, the deceleration value will be reduced to make the sum legal (i.e. the value of `accelerate` will be clamped to 1, and then the value of `decelerate` will be clamped to 1-`accelerate`).

accelerate
: Defines a simple acceleration of time for the element. Element time will accelerate from a rate of 0 at the beginning up to a *run rate*, over the course of the specified proportion of the simple duration.
The default value is 0 (no acceleration).
Legal values are floating point values between 0 and 1 (inclusive).

decelerate
: Defines a simple deceleration of time for the element. Element time will decelerate from a *run rate* down to 0 at the end of the simple duration, over the course of the specified proportion of the simple duration.
The default value is 0 (no deceleration).
Legal values are floating point values between 0 and 1 (inclusive).

The details of the accelerate and decelerate modifications are described in Details of the time manipulations .

Examples:

In this example, a motion path will accelerate up from a standstill over the first 2 seconds, run at a faster than normal rate for 4 seconds, and then decelerate smoothly to a stop during the last 2 seconds. This makes an animation look more realistic. The `animateMotion` element is defined in the Animation section of SMIL Boston.

```
<img ...>
    <animateMotion dur="8s" accelerate=".25" decelerate=".25" .../>
</img>
```

In this example, the image will "fly in" from off-screen left , and then decelerate quickly during the last second to "ease in" to place. This assumes a layout model that supports positioning (a similar effect could be achieved by animation the position of a `region` in SMIL layout). The `animate` element is defined in the Animation section of SMIL Boston.

```
<img ...>
    <animate attributeName="left" dur="4s" decelerate=".25"
             from="-1000" to="0" additive="sum" />
</img>
```

autoReverse attribute

This defines "play forwards then backwards" functionality. The use of autoReverse effectively doubles the simple duration. When combined with repeating behavior, each repeat iteration will play once forwards, and once backwards. This is useful for animation, especially for mechanical and pendulum motion.

autoReverse
Controls autoReverse playback mode.

Argument values are Booleans.
The default value is false (i.e. play normally).

The details of the autoReverse modification are described in Details of the time manipulations .

In this example, a motion path will animate normally for 5 seconds moving the element 20 pixels to the right, and then run backwards for 5 seconds (from 20 pixels to the right back to the original position), then forwards again and then backwards again, leaving the element at its original location. The active duration of the animation is 20 seconds. The `animateMotion` element is defined in the Animation section of SMIL Boston.

```
<img ...>
    <animateMotion by="20, 0" dur="5s" autoReverse="true" repeatCount="2"/>
</img>
```

Repeating elements

SMIL 1.0 introduced the repeat attribute, which is used to repeat a media element or an entire time container. SMIL Boston 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. The SMIL 1.0 repeat attribute is deprecated in SMIL Boston (it must be supported in SMIL document players for backwards compatibility).

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.

If the simple duration is indefinite, the element cannot repeat. In this case, any `repeatCount` attribute is ignored, although a `repeatDur` attribute value can still constrain the active duration. See also Computing the Active Duration .

repeatCount and repeatDur attributes

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 parent local time to repeat the simple duration.
  - "indefinite"
    The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

At most one of `repeatCount` or `repeatDur` should be specified. If both are specified (and the simple duration is not indefinite), the active duration is defined as the minimum of the specified `repeatDur`, and the simple duration multiplied by `repeatCount`. For the purposes of this comparison, a defined value is considered to be "less than" a value of "indefinite".

If the simple duration is indefinite, any `repeatCount` attribute will be ignored. Any `repeatDur` attribute value (other than "indefinite") will be used to constrain the indefinite simple duration. See also the examples below describing `repeatDur` and an indefinite simple duration.

If the simple duration is 0, any `repeatCount` attribute will be ignored. Any `repeatDur` attribute value will be used to define the active duration by showing the state of the element for the specified duration (this may be constrained by an `end` value - see Controlling active duration). See also the examples below describing `repeatDur` and a simple duration of 0).

@@ If simple duration is 0 and repeatCount is "indefinite" is the active duration 0 or indefinite?

If an element specifying audio media has a simple duration of 0 (e,g, because of `clipBegin` and `clipEnd` values), nothing should played even if the repeatDur specifies an active duration. The time model behaves according to the description, but no audio should be played.

These rules are included in the section Computing the Active Duration .

Examples

Need to create normative examples that demonstrate the new controls, and the interaction with implicit and explicit simple durations. Examples must also demonstrate the interaction of repeating behavior and time container constraints.

@@ Need to add example of repeatCount < 1 and/or repeatDur < simple duration

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 zero (0) or 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 effectively ignored. Nevertheless, this is not considered an error. The active is equal to the simple duration: for the first element, the active duration is 0, and for the second element, the active duration is indefinite.

```
<img src="foo.jpg" repeatCount="2" />
<img src="bar.png" dur="indefinite" repeatCount="2" />
```

In the following example, the simple duration is 0, and so repeat behavior is not meaningful. However, the `repeatDur` determines the active duration. The effect is that the text is shown for 10 seconds.

```
<text src="intro.html" 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" >
   <img src="img1.jpg" dur="5s" />
   <img src="img2.jpg" dur="4s" />
   <img src="img3.jpg" dur="4s" />
</seq>
```

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 Boston 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 players 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 Boston 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.

repeat Attribute

repeat
: *This attribute has been deprecated in SMIL Boston 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".

Note that elements that use the SMIL 1 `repeat` attribute with a value of "indefinite" are defined to end immediately after they begin. I.e. the active duration is effectively defined to be 0. This semantic is specific to the SMIL 1 `repeat` attribute, and does not apply to the new `repeatCount` and `repeatDur` attributes.

Controlling active duration

SMIL Boston provides an additional control over the active duration. The `end` attribute allows the author to constrain the active duration of the animation by specifying an end value using a simple offset, a time base, an event-base or DOM methods calls. The `end` attribute generally *constrains* the active duration that is otherwise defined by `dur` and any repeat behavior, although it will extend an *implicit* simple duration (see examples below). The rules for combining the attributes to compute the active duration are presented in the next section, Computing the active duration .

The normative syntax rules for each attribute value variant are described in the section Timing Attribute Values ; a syntax summary is provided here as an aid to the reader.

end : smil-1.0-syncbase-value * | end-value-list | "indefinite"
: Defines an end value for the animation that can constrain the active duration. The attribute value is either a SMIL 1.0 syncbase declaration, a semi-colon separated list of values, or the special value "indefinite".

  smil-1.0-syncbase-value * : "id(" id-ref ")" ("(" ("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.
    *Note: Only compliant SMIL document players are required to support the SMIL 1.0 syncbase-value syntax. Language designers integrating SMIL Boston Timing and Synchronization should not support this syntax.

  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 below.

  "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 Supported Methods section.

  end-value : (offset-value | syncbase-value | syncToPrev-value | event-value | media-marker-value | wallclock-sync-value)
  : 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 local time on the parent time container.

  syncbase-value : (id-ref "." ("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.

  syncToPrev-value : ("prev.begin" | "prev.end") (("+" | "-") clock-value)?
  : Specifies the previous timed sibling element, as reflected in the DOM, as the syncbase element, and describes the syncbase time and an offset from that syncbase. The end value is defined relative to the begin or active end of the previous sibling element.

  event-value : (id-ref ".")? (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 .

media-marker-value : id-ref ".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].

If `end` specifies an event-value or syncbase-value that is not resolved, the value of `end` is considered to be "indefinite" until resolved.

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 Restarting elements).

In general, the earliest time in the list determines the end value used in Computing the Active Duration . In the case where an element can begin multiple times, the end value used is the earliest end time after the current begin time. There are additional constraints upon the evaluation of the begin and end time lists, detailed in Evaluation of begin and end time lists .

The end value generally constrains all other values, and does not extend the active duration. However it *will* extend an *implicit simple duration*. 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 (media) duration. For the difference between the implicit simple duration and the active duration, the ending state of the media (e.g. the last frame of video) will be shown. This only applies to visual media - aural media will simply stop playing, or will not play at all if the implicit simple duration is 0 (e,g, because of `clipBegin` and `clipEnd` values).

In the following example, 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 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. 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 .

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 clicks on the "gobtn" element. The active duration will end 30 seconds after the parent time container begins. Note that if the user has not clicked on the target element before 30 seconds elapse, the element will never begin.

```
<par>
   <audio src="music.au" begin="gobtn.click" repeatDur="indefinite"
          end="30s" ... />
</par>
```

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.

```
<image src="image.jpg" end="click" />
```

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.

```
<seq>
  <video dur="5s" repeatCount="3" end="click; next.click" .../>
  <video dur="5s" repeatCount="3" end="click; next.click" .../>
  <video dur="5s" repeatCount="3" end="click; next.click" .../>
  <video dur="5s" repeatCount="3" end="click; next.click" .../>
  <video dur="5s" repeatCount="3" end="click; next.click" .../>
</seq>
```

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.

Computing the active duration

This section still needs work - and will change in the next day or two.

The table in Figure 3 defines a set of "forms" for the simple duration. These forms are used in the table in Figure 4 to delineate the possible combinations of attributes that can contribute to the active duration.

`dur`	Implicit media duration	Simple Duration	Form
number	*	`dur` value	explicit finite
"indefinite"	*	indefinite	"indefinite"
unspecified	0	media dur	implicit 0
unspecified	number	media dur	implicit finite
unspecified	continuous indefinite	indefinite	implicit indefinite
unspecified	unresolved	indefinite	unresolved

Figure 3 - Describing the simple duration

@@There are two forms of table 4 presented. We are trying to decide which presents the information more effectively. You be the judge which makes more sense, which is clearer, and which we should include. Both use essentially the same terminology, and present the same semantics (i.e. there should be no discrepancy in the semantics described, but rather only differences in how the information is presented).

The table in Figure 4 shows the semantics of all possible combinations of simple duration, `repeatCount` and `repeatDur`, and `end`. The following conventions are used in the table:

- The `repeatCount` and `repeatDur` attributes are specified as either:
  - "unspecified" meaning that the attribute is not used, or has an illegal attribute value
  - "finite" meaning that a legal numeric value is specified
  - "indefinite" meaning that the string value "indefinite" was specified
- The `end` attribute column specifies the end value obtained by evaluating the the attribute value according to the rules described in Controlling active duration and Evaluation of begin and end time lists . Note that a list of values yields a single end value at any given point of evaluation.
  - "unspecified" meaning that the attribute is not used, or has an illegal attribute value.
  - "finite" meaning that a legal numeric value is specified, or that a time specified in some other form has been resolved to a specific time.
  - "indefinite" meaning that the string value "indefinite" was specified.
  - "unresolved" meaning that some value was used that cannot (yet) be resolved to a specific time.

- Where the entry is a star ("*"), the value does not matter and can be any of the possibilities.

Note in particular that where a value specifies "unresolved", that the table will be reevaluated (generally using a different row) if and when the associated value becomes resolved. For example if the element specifies:

```
<audio src="5-second.au" end="foo.click" />
```

The active duration is initially defined as equal to the (implicit finite) simple duration. If the user clicks on "foo" before 5 seconds, the end value becomes resolved and the active duration table is re-evaluated to be MIN(d, end-B) which causes the element to end at the time of the click.

Some of the rules and results that are implicit in the table, and that should be noted in particular are:

- If `end` and `dur` are specified but neither of `repeatCount` or `repeatDur` are specified, then the active duration **AD** is defined as the minimum of the simple duration and the duration defined by `end`.
- If only `end` is specified (i.e. none of `dur`, `repeatCount` or `repeatDur` are specified), then the active duration **AD** is defined as the duration defined by `end` (in this case `end` overrides any implicit simple duration).
- If both `end` and either (or both) of `repeatCount` or `repeatDur` are specified, the active duration **AD** is defined by the minimum duration defined by the respective attributes.
- It is possible to have an indefinite simple duration and a defined, finite active duration. The active duration can *constrain* (cut short) the simple duration, but the active duration does not define the simple duration, or change its value (i.e. the simple duration is still indefinite).
- For any active duration and simple duration that are both not indefinite (and non-zero), the number of repeat iterations is defined by the active duration **AD** divided by the simple duration **d** (this may yield partial repeat iterations, just as `repeatCount` can specify).

Note that while the active duration is computed according to the rules in the table, the parent time container places constraints upon the active duration of all children. These constraints may cut short the active duration of any child, and so override the definition described here. For more information, see the section Time Container constraints on child durations .

The following symbols are used in the table as a shorthand:

B
: The begin of an element.

d
: The simple duration of an element.

@@This is a form of the table that has more rows, but may be clearer in delineating the different cases. It uses the term "unresolved indefinite" for end values, which is semantically equivalent to "unresolved" in the second table.

@@Note that in both table the handling of explicit 0 is a bit messy. This is only needed if we want to say that simple Dur of 0 and repeatCount of "indefinite" yields a 0 active duration, rather than an indefinite AD. The tables would simplify if we went with the latter.

Where the Active duration has a "+" suffix, the value may be reevaluated at some point. The footnotes within the row indicate when the value will be reevaluated. Note that when a value is re-evaluated, a different row in the table may apply.

Simple duration(d)	**`repeatCount`**	**`repeatDur`**	**`end`**	**Active Duration**
implicit 0 or explicit 0	(ignored)	unspecified	unspecified	0
implicit 0 or explicit 0	(ignored)	"indefinite"	unspecified	indefinite
explicit finite or implicit finite	unspecified	unspecified	unspecified	**d**
explicit finite or implicit finite	unspecified	unspecified	unresolved indefinite[1] or "indefinite"[2]	**d**+
"indefinite" or implicit indefinite	(ignored)	"indefinite" or unspecified	unspecified	indefinite
"indefinite", implicit 0, or implicit indefinite	(ignored)	"indefinite" or unspecified	unresolved indefinite[1] or "indefinite"[2]	indefinite+
unresolved[3]	number or unspecified	"indefinite" or unspecified	unresolved indefinite[1] or "indefinite"[2]	indefinite+
unresolved[3]	number	"indefinite"	unspecified	indefinite+
unresolved[3]	(ignored)	unspecified	unspecified	indefinite+
explicit finite or implicit finite	number	"indefinite" or unspecified	unspecified	`repeatCount`***d**

explicit finite or implicit finite	number	"indefinite" or unspecified	unresolved indefinite[1] or "indefinite"[2]	`repeatCount`***d**+
(ignored)	"indefinite" or unspecified	number	unspecified	`repeatDur`
(ignored)	"indefinite" or unspecified	number	unresolved indefinite[1] or "indefinite"[2]	`repeatDur+`
"indefinite", implicit 0, or implicit indefinite	(ignored)	number	unspecified	`repeatDur`
"indefinite", implicit 0, or implicit indefinite	(ignored)	number	unresolved indefinite[1] or "indefinite"[2]	`repeatDur+`
unresolved[3]	number	number	unspecified, unresolved indefinite[1], or "indefinite"[2]	`repeatDur+`
explicit finite	unspecified	unspecified	number	MIN(d, `end`-**B**)
explicit finite	unspecified	"indefinite"	number	`end`-**B**
"indefinite", implicit 0, or implicit indefinite	(ignored)	"indefinite" or unspecified	number	`end`-**B**
unresolved[3]	number	"indefinite" or unspecified	number	`end`-**B**+
(ignored)	"indefinite"	"indefinite" or unspecified	number	`end`-**B**
implicit finite or unresolved	unspecified	"indefinite" or unspecified	number	`end`-**B**
explicit finite or implicit finite	"indefinite"	"indefinite" or unspecified	unspecified	indefinite

(ignored)	unspecified	"indefinite"	unresolved indefinite[1] or "indefinite"[2]	indefinite+
(ignored)	"indefinite" or unspecified	"indefinite"	unspecified	indefinite
(ignored)	"indefinite"	"indefinite" or unspecified	unresolved indefinite[1] or "indefinite"[2]	indefinite+
explicit finite or implicit finite	number	number	unspecified	MIN(`repeatDur`, `repeatCount`***d**)
explicit finite or implicit finite	number	number	unresolved indefinite[1] or "indefinite"[2]	MIN(`repeatDur`, `repeatCount`***d**)+
explicit finite or implicit finite	number	number	number	MIN(`end`-**B**, `repeatDur`, `repeatCount`***d**)
(ignored)	"indefinite" or unspecified	number	number	MIN(`end`-**B**, `repeatDur`)
explicit finite or implicit finite	number	"indefinite" or unspecified	number	MIN(`end`-**B**, `repeatCount`***d**)
"indefinite", implicit 0, or implicit indefinite	(ignored)	number	number	MIN(`end`-**B**, `repeatDur`)
unresolved[3]	number	number	number	MIN(`end`-**B**, `repeatDur`)+

Figure. 4a: Computing the active duration for different combinations of the simple duration, `repeatDur` and `repeatCount`, and `end`.

[1] reevaluate if/when `end` becomes resolved by event
[2] reevaluate if/when `end` becomes resolved by DOM
[3] reevaluate when simple duration is resolved

@@This is a form of the table that has fewer rows, and does not call out the values that may be re-evaluated.

Note that any row that includes an "unresolved" value may be re-evaluated at some point (i.e. if the value becomes resolved). Note that when a value is re-evaluated, a different row in the table may apply.

Simple duration **d**	`repeatCount`	`repeatDur`	`end`	Active Duration
expl. finite	unspecified	unspecified	unspecified, "indefinite" or unresolved	**d**
implicit 0, or implicit finite	unspecified	unspecified	unspecified	**d**
"indefinite", impl. indefinite or unresolved	(ignored)	unspecified or "indefinite"	unspecified	indefinite
implicit 0 or explicit 0	"indefinite"	unspecified	unspecified	0
"indefinite", implicit finite, impl. indefinite or unresolved	unspecified or "indefinite"	unspecified or "indefinite"	"indefinite" or unresolved	indefinite
expl. finite, implicit 0, or implicit finite	expl. finite	unspecified or "indefinite"	unspecified, "indefinite" or unresolved	`repeatCount`***d**
expl. finite or implicit finite	unspecified or "indefinite"	expl. finite	unspecified, "indefinite" or unresolved	`repeatDur`
implicit 0, "indefinite", impl. indefinite or unresolved	(ignored)	expl. finite	unspecified, "indefinite" or unresolved	`repeatDur`
expl. finite	unspecified	unspecified	expl. finite	MIN(**d**, `end`-**B**)
expl. finite	"indefinite"	unspecified or "indefinite"	expl. finite	`end`-**B**
expl. finite	unspecified or "indefinite"	"indefinite"	expl. finite	`end`-**B**

implicit finite	unspecified or "indefinite"	unspecified or "indefinite"	expl. finite	`end`-**B**
implicit 0, "indefinite", impl. indefinite or unresolved	(ignored)	unspecified or "indefinite"	expl. finite	`end`-**B**
(anything except a 0 value)	"indefinite"	unspecified or "indefinite"	unspecified, "indefinite" or unresolved	indefinite
(anything)	unspecified or "indefinite"	"indefinite"	unspecified, "indefinite" or unresolved	indefinite
expl. finite or implicit finite	expl. finite	expl. finite	unspecified, "indefinite" or unresolved	MIN(`repeatCount`***d**, `repeatDur`)
expl. finite or implicit finite	expl. finite	expl. finite	expl. finite	MIN(`repeatCount`***d**, `repeatDur`, (`end`-**B**))
*	unspecified or "indefinite"	expl. finite	expl. finite	MIN(`repeatDur`, (`end`-**B**))
implicit 0, "indefinite", impl. indefinite or unresolved	(ignored)	expl. finite	expl. finite	MIN(`repeatDur`, (`end`-**B**))
expl. finite or implicit finite	expl. finite	unspecified or "indefinite"	expl. finite	MIN(`repeatCount`***d**, (`end`-**B**))

Figure. 4b: Computing the active duration for different combinations of the simple duration, `repeatDur` and `repeatCount`, and `end`.

It is possible to combine scheduled and interactive timing, e.g.:

```
<par dur="30s">
   <img id="mutebutton" src="mute.jpg"/>
   <text  src="description.html" />
   <audio src="audio.au" end="mutebutton.click"/>
</par>
```

The image and the text appear for the specified duration of the `<par>` (30 seconds). The audio will stop early if the image is clicked before the active end of the audio (which in this case is the duration of the actual audio media "audio.au").

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 on its own, or in response to user clicks:

```
<seq>
   <img src="slide1.jpg" dur="10s" end="click" />
   <img src="slide2.jpg" dur="10s" end="click" />
   <img src="slide3.jpg" dur="10s" end="click" />
   <!-- etc., etc. -->
</seq>
```

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.

Freezing elements

By default when an element's active duration ends, it is no longer presented (or its effect is removed from the presentation, depending upon the type of element). Freezing an element extends it, using the last state presented in the active 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]).

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. For discrete media, the media is simply displayed as it would be during the active duration. For continuous media, the "frame" that corresponds to the end of the active duration is shown. For algorithmic media like animation, the value defined for the end of the active duration should be used. The syntax of the fill attribute is the same as in SMIL 1.0, with two extensions:

fill : ("remove" | "freeze" | "hold" | "transition")
: This attribute can have the following values:

remove
: Specifies that the element will not extend past the end of the active duration.

freeze
: Specifies that the element will extend past the end of the active duration by "freezing" the element state at the active end. The parent time container of the element determines how long the element is frozen (as described 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).

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. The element is frozen to

extend to its *active duration + the time for transition* within same time container. @@Need to refine this definition to be clearer where the transition duration comes from, etc. Need to specify that this only applies if there is a transition specified as per the Transitions module.

This attribute only has an effect on visual media elements. Non-visual media elements (audio) should ignore this.

Note that `<a>` and `<area>` elements are still sensitive to user activation (e.g. clicks) when frozen. See also the SMIL 1.0 specification [SMIL10].

The default value of the `fill` attribute depends on the element type, and whether the element specifies any of the attributes that define the simple or active duration.

- For elements that act as time containers (including media time container elements as well as `<par>`, `<seq>` and `<excl>`), the default value is "remove".
- For other timed elements, if none of the attributes `dur`, `end`, `repeatCount` or `repeatDur` are specified on the element, then the default value of `fill` is "freeze".
- Otherwise, the default value is "remove".

An element with `fill="freeze"` 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>`. 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>`. 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` time container .

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.

@@Need a good example of freeze on a time container, showing both how it extends any frozen children, as well as how it cuts off and freezes any children that were active at the end.

Restarting elements

When an element is defined to begin at a simple offset (e.g. `begin="5s"`), there is an unequivocal time when the element begins. However, if an element is defined to begin relative to an event (e.g. `begin="foo.click"`), the event can happen at any time, and moreover can happen *more than once* (e.g. if the user clicks on "foo" several times). In some cases, it is desirable to *restart* an element if a second begin event is received. In other cases, an author may want to preclude this behavior.

In SMIL Boston, an element can have a list of begin values. In some cases, the intent is to begin at the earliest of the specified times (e.g. when the user clicks on any one of several images). In other cases, the intent is that the element restart when any of the begin times is encountered.

In addition, if an element is defined to begin relative to when another element begins (using the syncbase- value syntax), the syncbase element can restart. The `restart` attribute is used to control the restart behavior of an element.

`restart` = "**always | whenNotActive | never**"

always
: The element can be restarted at any time.
: This is the default value.

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.

The default value for the restart attribute is "always". This may not be a sensible default in all documents. In particular SMIL Boston documents with streaming media may want restart="never" set on all of the elements. In order to not require *restart="never" be added to every media element in the document, the WG is considering ways to override the default and set a new default for the document.*

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.

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, and only the current begin "instance" of the element is reflected in the presentation.

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 .

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.

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 restart setting for an animation is evaluated when the syncbase element restarts, when the eventbase event happens, or when the DOM method call (e.g. `beginElement()`) happens. For example:

```
<img id="go_btn" dur="indefinite" .../>
<video id="foo" begin="go_btn.click" ... />
<audio id="bar" begin="foo.begin+2s" dur="10s"
       restart="whenNotActive" ..." />
```

If the user clicks on the "go_btn" image at 5 seconds, element "foo" will begin, and element "bar" will be scheduled to begin at 7 seconds. If the user clicks the image again at 6 seconds, "foo" would restart and "bar" would be rescheduled to start at 8 seconds. If the user clicks again at 9 seconds, "foo" would restart but "bar" will not, as it is set to allow restart only when it is not active.

If an element is currently active when a restart is scheduled, the element should end immediately (at the time of the restart evaluation). It should not continue playing until the rescheduled begin actually happens. For example:

```
<img id="go_btn" dur="indefinite" .../>
<video id="foo" begin="go_btn.click" .../>
<audio id="bar" begin="foo.begin+2s" dur="10s" />
```

If the user clicks the image once at 3 seconds, "foo" begins to play and 2 seconds later "bar" will play as well. If the user clicks again at 6 seconds, "foo" restarts immediately, "bar" is stopped, and "bar" will restart at 8 seconds.

Note that using restart can also allow the author to define a single UI event to both begin and end an element, as follows:

```
<img id="toggle" dur="indefinite" .../>
<audio id="foo" begin="toggle.click" end="toggle.click"
   repeatDur="indefinite" restart="whenNotActive" .../>
```

If "foo" were defined with the default restart behavior "always", a second click on the image would simply restart the audio. However, since the second click cannot restart the audio when restart is set to "whenNotActive", the click will just end the active duration and stop the audio. This is sometimes described as "toggle" activation. See also Unifying scheduling and interactive timing .

Note that in SMIL Language documents, a SMIL element cannot be visible before it begins so having a begin="click" means it won't ever begin. In languages with `timeAction` support, this may not be the case. For example, the following is reasonable:

```
<span begin="click" end="click" timeAction="class:highlight">
  Click here to highlight. Click again to remove highlight.
</span>
```

See also "The SMIL Integration Module" for details of language profiles.

Using restart for toggle activation

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:

```
<img id="foo" begin="bar.click" end="bar.click"
              restart="whenNotActive" ... />
```

If "foo" were defined with the default restart behavior "always", a second click on the "bar" element would simply restart the element. However, since the second click cannot restart the element when `restart` is set to "whenNotActive", the element ignores the "begin" specification of the "click" event. The element can then use the "click" event to end the active duration and stop the element.

This is based upon the event sensitivity semantics described in Unifying Scheduling and Interactive Timing .

10.3.3 Time containers

SMIL Boston specifies three time containers: <par>, <seq>, and <excl>.

The par time container

<par>
: A `<par>` container defines a simple parallel time grouping in which multiple elements can play back at the same time.

The default syncbase of the child elements of a `<par>` is the begin of the `<par>`. This is the same element introduced with SMIL 1.0.

The `<par>` element supports all element timing.

The seq time container

<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 default syncbase of a child element 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 default syncbase is the begin of the sequence time container.

Child elements may define an offset from the syncbase, but may not define a different syncbase (i.e. they may not define a begin time relative to another element, or to an event). Thus, the syncbase of child elements of a sequence is always the same as the default syncbase. Note however that child elements *may* define an `end` that references other syncbases, event-bases, etc.

For children of a sequence, only the offset begin values are legal. None of the following begin values may be used:

disallowed begin-values:
: (syncbase-value | syncToPrev-value | event-value | media-marker-value | wallclock-sync-value)

No constraints are placed upon the `end` argument values.

The `<seq>` element itself supports all element timing.

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 a given time. For details, see Hyperlinks and timing and specifically Implications of beginElement() and hyperlinking for seq and excl time containers .

The excl time container

SMIL Boston defines a new time container, `<excl>`.

<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 default syncbase of the child elements of the `<excl>` is indefinite (i.e. equivalent to `begin="indefinite"`).

The `<excl>` element itself supports all element timing.

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 as way so to preserve the sync relationship to the video:

```
<par>
   <video id="vid" .../>
   <excl>
      <par begin="englishBtn.click" >
         <audio begin="vid1.begin" src="english.au" />
      </par>
      <par begin="frenchBtn.click" >
         <audio begin="vid1.begin" src="french.au" />
      </par>
      <par begin="swahiliBtn.click" >
         <audio begin="vid1.begin" src="swahili.au" />
      </par>
   </excl>
</par>
```

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.

Pause behavior

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.

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).

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. 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 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. 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 careful choice of defaults makes common use cases very simple. See the examples below.

<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>).

The <excl> element content model is thus (assume that container content is an updated version of the SMIL 1.0 DTD entity):

```
<!ENTITY % excl-content "priorityClass* | %container-content;">
<!ELEMENT excl (%excl-content;)*>
```

The <priorityClass> element supports a simple set of attributes to describe the behavior of its children:

```
<!ELEMENT priorityClass %container-content;>
<!ATTLIST priorityClass
  id     ID   #IMPLIED
  peers  ( stop | pause | defer | never )  'stop'
  higher ( stop | pause )                  'pause'
  lower  ( defer | never )                 'defer'
>
```

If no <priorityClass> element is used, all the children of the <excl> are considered to be *peers*, with the default `peers` behavior "stop".

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).

`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. This can also be thought of as placing the new element in the pause queue, paused at its very beginning.

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. This can also be thought of as placing the new element in the pause queue, paused at its very beginning. 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.

excl and priorityClass Examples

Note that because of the defaults, the simple cases work without any additional syntax. In the basic case, all the elements default to be peers, and stop one another:

```
<excl dur="indefinite">
   <audio id="song1" .../>
   <audio id="song2" .../>
   <audio id="song3" .../>
   ...
   <audio id="songN" .../>
</excl>
```

is equivalent to the following with explicit settings:

```
<excl dur="indefinite">
   <priorityClass peers="stop">
     <audio id="song1" .../>
     <audio id="song2" .../>
     <audio id="song3" .../>
     ...
     <audio id="songN" .../>
   </priorityClass>
</excl>
```

If the author wants elements to pause rather than stop, the syntax is:

```
<excl dur="indefinite">
   <priorityClass peers="pause">
     <audio id="song1" .../>
     <audio id="song2" .../>
     <audio id="song3" .../>
     ...
     <audio id="songN" .../>
   </priorityClass>
</excl>
```

The audio description use case for visually impaired users would look very similiar 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>
```

Pause queue semantics

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 local 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, in an *inactive* (idle) state. 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.

Time dependency and pause/defer semantics

@@This section has more general impact that just in excl, and should perhaps be moved elsewhere. E.g. when an element is paused with the pause() DOM method, this semantic will apply as well.

When an element is paused, a resolved end time for the element may no longer be resolved (although it *could* be computed in some cases). This change in the end time must be propagated to any sync arc time dependents defined relative to the active end of the paused element. See also the "Propagating Times" section in the Timing draft.

When an element is deferred, sync-arc time-dependents of the element are resolved when the element *actually begins*, and not when it is placed in the queue. Similarly, the begin event is not raised until the element begins.

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).

To specify that a child of the `<excl>` should begin playing by default (i.e., when the `<excl>` begins), specify `begin="0"` on that child element. If children of an `<excl>` are scheduled 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. For example:

```
<excl>
   <img src="image1.jpg" begin="0s" dur="5s"/>
   <img src="image2.jpg" begin="0s" dur="5s"/>
   <img src="image3.jpg" begin="0s" dur="5s"/>
</excl>
```

Given the default semantics for excl, the first image will begin and then be immediately stopped by the second image, which will in turn be immediately stopped by the third image. The net result is that only the third image is seen, and it lasts for 5 seconds. Note that the begin and end events for the first two images are raised and propagated to all time dependents. If the behavior is set to "pause" as in this example, the declared order is effectively reversed:

```
<excl>
   <priorityClass peers="pause">
      <img src="image1.jpg" begin="0s" dur="5s"/>
      <img src="image2.jpg" begin="0s" dur="5s"/>
      <img src="image3.jpg" begin="0s" dur="5s"/>
   </priorityClass>
</excl>
```

In this case, the first image will begin and then be immediately paused by the second image, which will in turn be immediately paused by the third image. The net result is that the third image is seen for 5 seconds, followed by the second image for 5 seconds, followed by the 3rd 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:

```
<excl>
   <img src="image1.jpg" begin="0s".../>
   <img src="image2.jpg" begin="10s; image1.click".../>
   <img src="image3.jpg" begin="20s; image2.click".../>
</excl>
```

Note, some surprising results may occur when combining scheduled and interactive timing within an `<excl>`. If in the above example, the user clicks on image1 and then on image2 before ten seconds have elapsed, image 2 will re-appear at the ten second mark. Image 3 will appear at twenty seconds. The likely intent of this particular use-case would be better represented with a `seq` time container.

Side effects of activation

Children of the `<excl>` can be activated by scheduled timing, hyperlinks, events or DOM methods calls. For all but hyperlink activation, the `<excl>` time container must be active for child elements of the `<excl>` to be activated. With hyperlink activation, the document may be seeked to force the parent `<excl>` to be active, and a seek may occur to the begin time target child if it has a resolved begin time. That is, the normal hyperlink seek semantics apply to a timed child of an `<excl>`.

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 .

Specifying the simple duration of par and excl with endSync

The `endSync` attribute is only valid for `<par>` and `<excl>` time containers, and media elements with timed children (e.g. `animate` or `area` elements). The `endSync` attribute controls the end of the simple duration of these containers, as a function of the children. This is particularly useful with children that have "unknown" duration, e.g. an mpeg movie, that must be played through to determine the duration.

@@ Add more info to the effect that: Note that paused children of an <excl> container have not ended their active duration. Moreover (something to explain the "Elements do not have to play to completion, but must have played at least once") (i try ...) elements with multiple activation due to multiple begin and end value has only to play once to be considered as having ended their active duration.

`endSync` = " first | last | all | id-ref *"*

Legal values for the attribute are:

first
: The `<par>`, `<excl>`, or media element simple 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 active duration.

last
: The `<par>`, `<excl>`, or media element simple 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 begin time.
This is the default value.

all
: The `<par>`, `<excl>`, or media element simple 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.

id-ref
: The `<par>`, `<excl>`, or media element simple duration ends with the specified child. The id must correspond to one of the immediate children of the `<par>` time container.
Example: `<par ... endSync="movie1" ...>`

Semantics of `endSync` and indeterminate children:

- `endSync="first"` means that the time container must wait for any element to actually end its active duration. It does not matter whether the the first element to end was scheduled or interactive.

- `endSync="last"` means that the time container must wait for all elements that have a resolved begin, to end the respective active durations. Note that elements that had an interactive begin, but that became resolved before all scheduled elements ended, are added to the set of children that must end their active durations before the parent can end. 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.
  At the point at which no elements are active, and no elements have a resolved begin time after the current time, the parent time container can end its simple duration.
  Elements with indefinite or unresolved begin times will *not* keep the simple duration of the time container from ending.
  Elements with a resolved begin time but indefinite or unresolved end times *will* keep the simple duration of the time container from ending.
  @@Should we refine this to exclude elements with an explicit "indefinite" duration? Seems like this matches SMIL 1 more closely.
  @@ Also, how should be define "endSync=last" to work with children that have multiple resolved begin times? Must all instances play, or must only one play. We could also say that all elements with a resolved begin time that have not already played once must have ended. The question is then whether we should or should not cut off elements if they are playing a second time.
  Child elements of an `<excl>` that are currently paused and waiting to resume *will* keep the simple duration of the time container from ending.
- `endSync="all"` means that every child element of the time container must end the active duration. In the case of element with multiple begin times, or restarting elements, note that elements do not have to play to completion; they just must have played at least once (here "once" refers to an instance of an active duration, and *not* to one repeat iteration of a repeating element). When all elements have completed the active duration one or more times, the parent can end.
  Note that child elements of an `<excl>` that are currently paused and waiting to resume (and have not already completed the active duration at least once) will keep the simple duration of the time container from ending.
- `endSync=[`*`id-ref`*`]` means that the time container must wait for the referenced element to actually end its active duration. The id-ref must refer to a child of the time container. If the referenced child has an indefinite active duration, then the simple duration of the time container is also indefinite.
  Note that if the referenced child is currently paused and waiting to resume, it will keep the simple duration of the time container from ending.

@@ Do we need a note to call out that in some cases, endSync may define an indefinite simple duration for the time container. This would flow "through the computing the active duration" table accordingly, using "implicit indefinite" as the simple duration.

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 don'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)
//   isResolved(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 dis-qualifying 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. Note that if it has already begun,
            // then we do not care if it has more resolved begins.
            if( c.isActive()
                || ( !c.hasStarted() && 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 do not care if it has more resolved begins.
            if( c.isActive() || !c.hasStarted() )
                return false;
            // else, keep checking (the assumed result is true)
            break;

      } // close switch

   } // close foreach loop

   return assumedResult;

   } // close timeContainerHasEnded()
```

Time container duration

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.

Implicit duration of <par> containers

By default, the simple duration of a `<par>` is defined by the `endSync=last` semantics. The simple duration will end when all scheduled children have ended their respective active durations.

The simple duration of a `<par>` container can be controlled with the `dur` and `endSync` attributes. If the `dur` attribute is specified, the `endSync` attribute is ignored. Using `endSync`, the end of the simple duration can be tied to the active end of the first child that finishes, or to the active end of the last child to finish (the default), or to the active end of a particular child element.

Implicit duration of <seq> containers

By default, the simple 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 simple duration of the `<seq>` is also indefinite.

Implicit duration of <excl> containers

The implicit simple 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, it will be common for `<excl>` time containers to have indefinite simple duration.

Implicit duration of media element time containers

For `endSync={last or all}`: The time children and the intrinsic media duration define the simple duration of the media element time container. If a continuous media duration is longer than the extent of all the time children, the continuous media duration defines the implicit simple duration for the media element time container. If the media is discrete, this is defined as for `<par>` elements.

For `endSync={first}`: The time children and the intrinsic media duration define the simple duration of the media element time container. The element ends when the first active duration ends, as defined above for `endSync` on a `<par>`, but no sooner than the end of the intrinsic media duration of continuous media. If the media is discrete, this is defined as for `<par>` elements.

For `endSync={ID}`: This is defined as for `<par>` elements.

If the calculated implicit simple duration 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 simple duration. This only applies to visual media - aural media will simply stop playing.

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 local time is not frozen during this period, and any children will run normally without being affected by the media intrinsic duration.

Time container constraints on child durations

Time containers place certain overriding constraints upon the child elements. These constraints can cut short the active duration of any child element.

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.

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 begin="5s" dur="indefinite" .../>
   <audio begin="prev.end" .../>
</par>
```

The video will play once for 6 seconds, and then a second time but only for 4 seconds - the last 2 seconds will get cut short and will not be seen. The text shows up for the last 5 seconds of the `<par>`, and the indefinite duration is cut short at the end of the simple duration of the `<par>`. The audio will not show up at all, since it is defined to begin at the end of the active duration of the previous element (the `<text>` element). Since the text element ends when the time container ends, the audio would begin after the time container has ended, and so never is heard. When the <par> repeats the first time, everything has happens just as it did the first time. However the last repeat is only a partial repeat (5 seconds), and so on the video will be seen, but it will not be seen to repeat, and the last second of the video will be cut off.

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 above.

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.

Time container constraints on sync-arcs and events

We need a few good examples to illustrate these concepts.

SMIL 1.0 defined constraints on sync-arc definition (e.g., begin="image1.begin"), allowing references only to qualified siblings. SMIL Boston explicitly removes this constraint. SMIL Boston 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

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 not unlike 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 .

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

@@ If we allowed events on begin in children of sequence, we would have to refine this language to say that an element is sensitive after the active end of the previous element, and until its own active end.

The parent time container must be active for the child element to receive events.

Sequence children are only sensitive to events during their active duration. In the following example, all children listen to the same end event and it works as expected:

```
<seq>
   <img src="img1.jpg" end="foo.click" />
   <img src="img2.jpg" end="foo.click" />
   <img src="img3.jpg" end="foo.click" />
</seq>
```

Negative begin delays

A begin time (ultimately) 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.

If the computed begin offset is negative, the time container constraints specify that the element cannot actually begin until the parent time container begins. Nevertheless, the element behaves as though it had begun earlier. A negative begin offset can be thought of as defining 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, a negative begin offset is equivalent to a `clipBegin` specification with the same magnitude as the offset value.

10.3.4 State transition model

@@This must be updated to reflect the impact of parent time container constraints and the DOM methods (especially including the "paused" state, and possibly Active-to-Active transition for seek()).

State paused: In the paused state, an animation continues to perform the transformation of the specified presentation values that were current at the moment of entering the pause state.

Pause transition: Active to Paused This transition may occur if an animation element has its pause() method called while in the active state, or as a result of <excl> stacking behavior and being interrupted by a sibling of the parent <excl>.

Unpause transition: Paused to Active This transition may occur if an animation element has its unpause() method called while in the pause state, or as a result of <excl> unstacking behavior.

At any moment in time, a timed element is in exactly one of the following states: *idle*, *active, finished* or *frozen*. The state transitions are caused by events called *start*, *restart*, *freeze* and *stop*. Figure 6 shows the legal transitions between the states of an element:

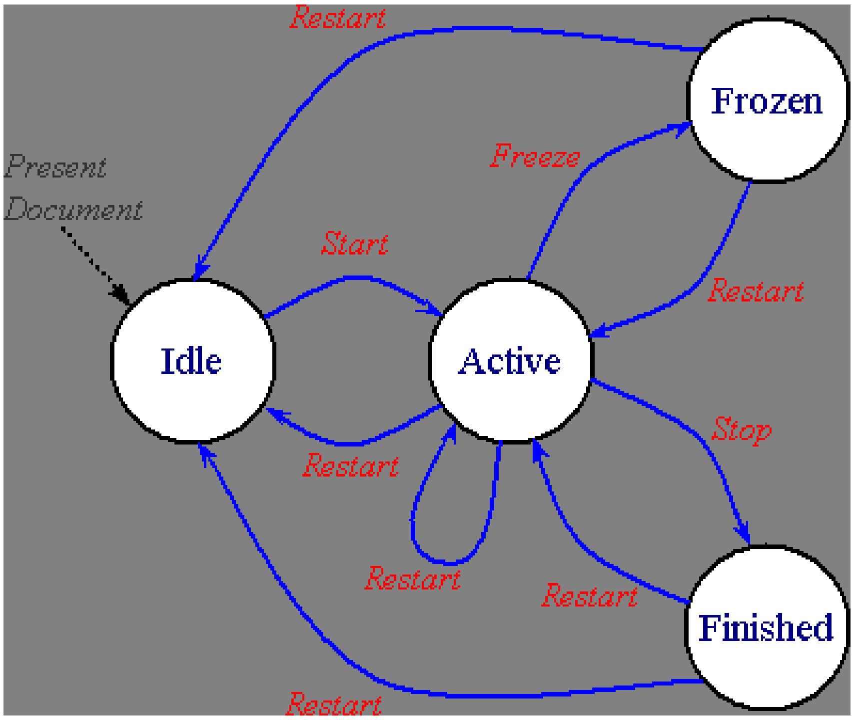


Figure 6: State diagram of an element

The following sections explain the semantics of the states and transitions of a timed element, and explain how to define the state transitions using timing attributes of the element.

Note that the states and transitions are part of the *model*, and do not imply a particular implementation. Note also that an element may transition through more than one state in a virtual instant (i.e. with no time spent in a given state).

The presentation effect of timed elements is generally to display media, or to play a timeline (e.g. for time containers). In some cases, the element may be an animation that manipulates the presentation, but does not directly display anything. In some integration scenarios, the presentation effect of the element may be to apply a stylesheet, or to otherwise modify the presentation. In these discussions, the common case of displaying media or playing a timeline is used to describe the states and transitions. The same semantics should be understood to apply to all defined actions or presentation effects, as specified in the language that integrates SMIL Timing and Synchronization.

Initial state: Idle

When the document that contains a timed element is first presented, the element is created in the *idle* state. This is the common starting state for all timed elements.

When a parent time container repeats or is restarted, all (timed) child elements of the time container will be reset. As part of the element reset, the element is re-initialized to the Idle state. See also Resetting element state .

In the *idle* state a timed element is inactive and does not affect the presentation of the document in any way. The element simply waits for the time or event specified in its `begin` attribute. Note that the element may transition immediately to the active state if the element begins immediately when the document begins.

Start transition: Idle to Active

For an element to become active, the element's parent time container must be active. Given this, a timed element in the *idle* state transitions to the *active* state when the condition specified in the `begin` attribute becomes true. As described in the section on the `begin` attribute, this condition may depend upon one of several factors:

- reaching a particular time in the parent time container simple duration.
- another element beginning or ending its active duration.
- another (media) element reaching a particular point (i.e. marker) in its simple duration.
- the occurrence of an asynchronous event, such as a mouse click.

Additionally, an element may be started by a DOM `beginElement()` or `beginElementAt()` method call, or as the result of being the target of an activated hyperlink.

An element may become active as soon as its parent time container becomes active, if the condition specified in the `begin` attribute is true at that point.

Note that the `begin` attribute can specify a condition that is a list of values. The specific semantics of evaluating the list of values is described in the section Basic - begin and dur .

Active state:

In the active state, a timed element displays the associated media or performs the described timeline associated with the element. The active state includes the entire active duration of the element. The active duration of an element is specified by the interaction between the `dur`, `end`, `repeatDur`, and `repeatCount` attributes as detailed in the section Computing the Active Duration .

Freeze transition: Active to Frozen

If a timed element has the `fill` attribute set to "freeze", "hold" or "transition", upon reaching the end of its active duration, the element will transition to the *frozen* state.

Frozen state:

In the *frozen* state the element will continue to present the last defined state of visual media or the timeline state at the end of the active duration (aural media render nothing during the frozen state). The duration of the frozen state depends upon the value of the `fill` attribute (as described in Freezing elements) and on the parent time container (as described in Time Container constraints on child durations).

The frozen state may have 0 duration, e.g. if the parent time container ends with the element.

Stop transition: Active to Finished

If a timed element has the `fill` attribute set to `remove` (the default), upon reaching the end of its active duration, the element will transition to the *finished* state. Note that the active duration of a child element may end when an ascendant time container ends its simple duration.

Finished state:

In the *finished* state the timed element does not affect the presentation of the document. The duration of the *finished* state depends upon the parent time container. The *finished* state lasts until the end of the *current simple duration* of the parent time container, or until the element is restarted (whichever comes first).

Restart transition: Frozen to Active

The ability of an element to make this transition depends upon the value of the `restart` attribute. If the `restart` attribute value is `always` or `whenNotActive` the element will transition to the active state in response to a DOM `beginElement()` or `beginElementAt()` method call, or an additional begin event. The restart transition effectively resets the state of the animation element; the element's simple and active duration must be recomputed as if it were being started for the first time. See also Resetting element state .

Restart transition: Frozen to Idle

This transition happens the same way as the Frozen to Active transition (immediately above). If the element specifies an offset from the syncbase or eventbase, or if the DOM `beginElementAt()` method call specifies a non-0 offset, then the element is returned to the Idle state until it actually restarts.

Restart transition: Active to Active

An element may receive a DOM `beginElement()` or `beginElementAt()` method call or may receive an additional begin event while in the *active* state. In this case, if the value of the `restart` attribute is "always" the element will re-transition to the *active* state and restart as described above. Any other value for the `restart` attribute will prevent this transition from occurring.

Restart transition: Active to Idle

This transition happens the same way as the Active to Active transition (immediately above). If the element specifies an offset from the syncbase or eventbase, or if the DOM `beginElementAt()` method call specifies a non-0 offset, then the element is returned to the Idle state until it actually restarts.

Restart transition: Finished to Active

An element restart can result from a DOM call or an additional begin event, subject to the restrictions imposed by the `restart` attribute. When in the finished state, an element may re-transition to the active state if the value of the `restart` attribute is "always" or "whenNotActive". If the `restart` attribute is set to "never", this transition can not occur.

Restart transition: Finished to Idle

This transition happens the same way as the Finished to Active transition (immediately above). If the element specifies an offset from the syncbase or eventbase, or if the DOM `beginElementAt()` method call specifies a non-0 offset, then the element is returned to the Idle state until it actually restarts.

10.3.5 Timing model details

Timing and real-world clock times

In this specification, elements are described as having local "time". In particular, many offsets are computed in the local time of a parent time container. However, simple durations can be repeated, and elements can begin and restart in many ways. As such, there is no direct relationship between the local "time" for an element, and the real world concept of time as reflected on a clock.

When the time manipulation attributes are used to adjust the speed and/or pacing within the simple duration, the semantics can be thought of as changing the pace of time in the given interval. An equivalent model is these attributes simply change the pace at which the presentation *progresses* through the given interval. The two interpretations are equivalent mathematically, and the significant point is that the notion of "time" as defined for the simple duration and "local time" should not be construed as real world clock time. For the purposes of SMIL Timing and Synchronization, "time" can behave quite differently from real world clock time.

Interval timing

The SMIL timing model assumes the most common model for *interval timing*. This 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 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 value sampled during the "frozen" state.

When repeating an element 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. However, 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 handling the frozen state.

- If the active duration is an even multiple of the simple duration, the media to show in the frozen state 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.

Unifying scheduling and interactive timing

A significant motivation for SMIL Boston 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 Boston 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 "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.

Background

SMIL Boston 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.

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 active end time. The element still exists within the constraints of the document, but the begin or active 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 can be thought of as *unresolved*.

The event-activation support provides a means of associating an event with the begin or active 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 .

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:

```
<par begin="10s" dur="5s">
   <audio src="song1.au" begin="btn1.click" />
</par>
```

If the user 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.

Event sensitivity

The SMIL Boston timing model supports synchronization based upon DOM events. These can be user interface generated and other kinds of unpredictable 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.

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) Note that an element will not receive an instance of an event that begins any ascendant time container.
   b)

   @@The above semantic is hard to implement based upon a standard DOM event model. We may have to reconsider and say: Note that if the element and its parent time container are both specified to begin with the same event, the behavior is not defined. DOM Level 2 events does not provide a means to order the registered listeners for an event, and so implementations cannot guarantee that the parent will be activated before the child. Authors should avoid this construct in documents.
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 and any specification of the event in `end` is ignored for this event instance.

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.
   b) 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.

These 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:

```
<audio src="bounce.wav" begin="foo.click" end="foo.click+3s"
       restart="whenNotActive"/>
```

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:

```
<audio src="bounce.wav" begin="foo.click" dur="3s"
       restart="whenNotActive"/>
```

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 indeterminate timing, hyperlinking yields a variant on interactive content. 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 .

Details of the time manipulations

Speed control

Speed modifies the pace of time for the element and its descendents, and so modifies the interpretation of the normal timing attributes with respect to the normal pace of (real-world) time. The attributes dur and repeatDur always specify a time in unmodified local time for the element. As a result, the observed simple duration and repeat duration for an element with a modified speed is not the same as the specified speed. This is important to making the model be consistent when the speed cascades in the time containment hierarchy, although it can make authoring somewhat more complex.

Note that a speed attribute on an element does not affect the element begin time. It may affect the element end time, if the end is defined only in terms of the simple duration or repeat duration. An end value (defined by the `end` attribute) is converted to element local time using the speed value. However, the result is that the active duration is not affected by the speed value, since the values (syncbase values, eventbase times, wallclocks times, etc.) are all defined in another timespace and converted to the local timespace. See also the examples below.

To compute the effect of speed on the simple duration or on the active duration if defined with repeat, the following function is used. This function is also used to convert a time in the parent local timespace to a time in the child local timespace that accounts for the speed attribute.

T_{par} is the time in the parent local timespace

T_{el} is the time in the element local timespace

$T_{el} = (T_{par} / speed)$

When speed is applied to a time container, it scales the rate of progress through the time container timeline. This effect cascades. When descendents also specify a speed value, the parent speed and the child speed are multiplied to yield the result. For example:

```
<par speed=2.0>
   <animation begin="2s" dur="9s" speed=0.75 .../>
</par>
```

The observed rate of play of the animation element is 1.5 times the normal play speed. The element begins 1 second after the par begins (the begin offset is scaled by the parent speed), and ends 6 seconds later (9/1.5).

The following example shows how an event based end combines with time filters:

```
<par speed=2.0>
   <animation begin="2s" dur="9s" speed=0.75
        repeatCount="4" end="click" .../>
</par>
```

This behaves as in the first example, but the animation element will repeat 4 times for a total of 24 seconds (in real time), unless a click happens before that. Whenever the click happens, the element ends. A variant on this demonstrates syncbase timing:

```
<par speed=2.0>
   <img id="foo" dur="30s" .../>
   <animation begin="2s" dur="9s" speed=0.75
        repeatCount="4" end="click, foo.end" .../>
</par>
```

The image will display for 15 seconds. The animation element plays at an observed rate of 1.5 times play speed, but it will end after 15 seconds, when the image ends. The animation will have repeated 2.5 times at this point. Note that although the animation has a speed value, this does not impact the semantic of the syncbase timing. When the syncbase, eventbase, wallclock or media marker time is observed to happen, it will be applied anywhere it is used at that real time (although various timespace conversions are applied internally).

Note that in the examples above, the default duration of the `<par>` container is defined as `endSync="last"`. This behavior is not affected by the speed modifications, in the sense that the observed end of the elements will produce the correct simple duration on the parent time container.

@@ Need to include notes about how begin offsets are also scaled by the parent timescale, but not the local element scale.

The following example illustrates an important effect of offset time scaling:

```
<par speed=2.0>
   <img id="foo" dur="30s" .../>
   <animation begin="2s" dur="9s" speed=0.75
        repeatCount="4" end="foo.end+6s" .../>
</par>
```

The image will display for 15 seconds. The animation element plays at an observed rate of 1.5 times play speed, and it will end after 18 seconds. The offset added to the end of the image is scaled by the parent time container speed. The animation will have repeated 3 times at this point.

The following example illustrates the speed modifications along with the fallback for a video element that can only play normal forward speed.

```
<par speed=2.0>
   <video id="foo" dur="30s" accelerate="0.25" ...>
      <area begin="2s" dur="4s" .../>
   </video>
   <animation begin="2s" dur="9s" speed=0.75
        repeatCount="4" end="foo.end+6s" .../>
</par>
```

The video ignores the acceleration and the speed value, and plays at normal speed for 15 seconds. The simple (and active) duration are still constrained by the speed. The area element reflects what the video is playing, and so the area becomes active after 2 seconds and remains active for 4 seconds. The animation element behaves just as it did in the previous example, and ends after 18 seconds and 3 repeats.

Issues with implicit duration and fallback speeds

@@ This should perhaps move to the Fallbacks section

Some media will use a fallback speed (e.g. because it cannot play at the requested speed - see the section Fallbacks for time filters on a media element). When this is the case, and the simple duration is defined by the implicit media duration, the results are sometimes less clean, as the following example illustrates. Assume that the video has an intrinsic duration of 30 seconds, and that this cannot be determined until the video plays through to the end (as is sometimes the case).

```
<par speed=1.5>
   <video id="foo" repeatCount="3" accelerate="0.25" ...>
      <area begin="2s" dur="4s" .../>
   </video>
</par>
```

The video ignores the acceleration and the speed value, and plays at normal speed for 30 seconds. At this point, the simple duration is resolved, and (accounting for the parent speed) is computed to be 20 seconds. At this point the video *should* be halfway into the second repeat iteration, moving 50% faster than normal playspeed. The video should continue playing for another 30 seconds. The fallback rules would define that each repeat iteration would play the first 20 seconds of the 30 second video. Given the linear behavior of the video element, the first 10 seconds will be shown for the (remainder of the) second repeat iteration, followed by the first 20 seconds of video for the third repeat iteration. Needless to say, this may not produce pleasing results. At the very least, authors should avoid mixing speed modifications with media elements that use the implicit simple duration, and cannot play at the specified speed. As stated elsewhere, the time modifications should be used where appropriate.

@@ Note sure if this next is useful:

Note that the timing of all durations, begin and end times and events, is based upon the computed values using the explicit speed and not the fallback speed. If the media player is nominally playing at the explicit speed, but varies somewhat, then the effective speed, end and durations should be used in the time model (i.e. the behavior of the runtime in computing effective versus desired times, and managing synchronization, should be consistent with the behavior of the media player and timing model when the speed is 1.0, or normal play speed). Try: the rules and mechanisms that a runtime uses to compute effective times versus desired times should be consistently applied if the "attempted" speed is normal playspeed (1.0) or offspeed (e.g. 2.0). If the "attempted" speed is not the desired speed, then a strict interpretation of the desired times should be used for effective times.

@@ Proposal to help authors - is this useful?:

Define a new test attribute canPlaySpeed (?) to support alternate presentations depending upon the capabilities of the media players. Should be evaluated at runtime for the given media and associated player. Evaluates true if the media player will attempt to play the specified speed. Evaluates false otherwise. If applied to a time container, then should consider all the descendents of the time container, and only evaluate to true if all descendents will attempt to play the desired speed. In evaluating this, the cascaded desired speed must be used, and not the explicit speed on each element.

Acceleration and Deceleration

@@ Borrow pictures and description of manipulation from keySplines in SMIL Animation? Need to describe that time is actually manipulated and remapped, but can think of it as progress.

The speed or rate of progress through the simple duration must be increased to account for the acceleration/deceleration and preserve the simple duration. The adjusted speed is described as the *run rate*. For an element with both acceleration and deceleration, the speed over the simple duration varies from 0 up to the run rate and then back down to 0.

To compute the run rate over the course of the simple duration, the following formula is used. Let **a** be the value of accelerate, and **b** be the value of decelerate. The run rate **r** is then:

```
r = 1 / ( 1 - a/2 - b/2 )
```

Thus, for example, if the value of accelerate is 1 (i.e. accelerate throughout the entire simple duration), the run rate is 2.

The speed **s(t)** at any point in time **t** (within the simple duration **d**) is defined as a function of the run rate, as follows:

```
For: ( 0 <= t < (a*d) )          I.e. in the acceleration interval
    s(t) = r * ( t / ( a * d ))

For: ( (a*d) <= t <= (d-(b*d)) ) I.e. in the run-rate interval
   s(t) = r

For: ( (d-(b*d)) < t <= d )      I.e. in the deceleration interval
   s(t) = r * ( ( t - (d-(b*d)) ) / ( b*d ))
```

If in place of **t** we use **p**, the proportional progress through the simple duration, the equations simplify somewhat:

```
p = t/d

For: ( 0 <= p < a )          I.e. in the acceleration interval
    s(p) = r * ( p / a )

For: ( a <= p <= (1-b) )     I.e. in the run-rate interval
   s(p) = r

For: ( (1-b) < p <= 1 )      I.e. in the deceleration interval
   s(p) = r * ( ( p - (1-b) ) / b )
```

Examples:

In this example, a motion path will accelerate up from a standstill over the first 2 seconds, run at a faster than normal rate for 4 seconds, and then decelerate smoothly to a stop during the last 2 seconds. This makes an animation look more realistic. The `animateMotion` element is defined in the Animation section of SMIL

Boston.

```
<img ...>
    <animateMotion dur="8s" accelerate=".25" decelerate=".25" .../>
</img>
```

In this example, the image will "fly in" from offscreen left , and then decelerate quickly during the last second to "ease in" to place. This assumes a layout model that supports positioning (a similar effect could be achieved by animation the position of a `region` in SMIL layout). The `animate` element is defined in the Animation section of SMIL Boston.

```
<img ...>
    <animate attributeName="left" dur="4s" decelerate=".25"
             from="-1000" to="0" additive="sum" />
</img>
```

Play Forwards then Backwards

Examples:

A simple example is provided in the syntax description above.

In the following example the motion path will behave as above, but will end at the earlier of 15 seconds or when the user clicks on the image. If the element ends at 15 seconds (if the user does not click), the motion path will leave the element at the end of the defined path, 20 pixels to the right. Note that `repeatDur` and `end` are not affected by the `autoReverse` attribute (although `repeatCount` is).

```
<img ...>
    <animateMotion  by="20, 0" dur="5s" autoReverse="true"
         repeatDur="15" end="click" fill="freeze"/>
</img>
```

Accelerate and decelerate can be combined with autoReverse, and are applied to the unmodified simple duration. For example:

```
<img ...>
    <animateMotion by="20, 0" dur="4s" autoReverse="true"
       accelerate=".25" decelerate=".25" />
</img>
```

This will produce a kind of elastic motion with the path accelerating for 1 second from the original position as it moves to the right, moving slightly faster than normal for 2 seconds, and then decelerating for 1 second as it nears the points 20 pixels to the right. It accelerates back towards the original position and decelerates to the end of the reversed motion path, at the original position.

Speed can also be combined with autoReverse, and modifies the entire effect. This example combines all three time manipulations:

```
<img ...>
    <animateMotion by="20, 0" dur="4s" autoReverse="true"
       speed="0.5" accelerate=".25" decelerate=".25" />
</img>
```

This produces the same effect as in the previous example, except that everything moves half as quickly and takes twice as long. The active duration is 16 seconds.

Timing Model

A theoretical model can be described that assumes that all element local timelines (including any media elements) are completely non-linear and have unconstrained ballistics. This ideal model can be applied to many applications, including pure rendered graphics, text, etc. Nevertheless, many common applications also include media with linear behavior and other constraints on playback. When the timegraph includes media elements that have linear behavior, the model must adapt, and/or provide consistent semantics that accommodate these real world constraints. This sections below include a discussion of these fallback semantics.

Note that while the model does *support* timegraphs with a mix of linear and non-linear behavior, and defines specific semantics for timegraphs that cannot support the ideal non-linear model, it is *not* a goal to provide an ideal alternative presentation for all possible timegraphs with such a mix. It is left to authors and authoring tools to apply the time manipulations in appropriate situations. This section describes both the ideal model as well as the semantics associated with linear-media elements.

Ideal model

In the ideal model, the pace or speed of local time can be manipulated arbitrarily. The graph advances (or is sampled, depending upon your perspective) as the presentation time advances. A time container samples each of its children in turn, so that a graph traversal is performed for each render time. Elements that are idle or stopped (i.e. neither active nor frozen) are pruned from the traversal as an optimization. As the traversal moves down the graph (from time containers to children), each local timeline simply transforms the current time from the parent time-space to the local time space, and then samples the local timeline at the transformed current time. Note that the speed and effects of the time filters effectively cascade down the time graph, since each element transforms local time for itself and all descendents.

When linear media are added to this model and the "current time" (sample) traversal encounters a media element, the media element is effectively told to "sample" at a particular position. Given that linear media can not sample arbitrarily, the semantic that is used is to verify the current position of the media (as observed on the player) against the current theoretical timeline position for the timegraph. Within certain limits (e.g. defined by a syncTolerance attribute), divergence from the theoretical timeline position are ignored. So far, this is just a typical implementation of a synchronization manager.

When the speed (and even direction) of the local timeline can vary from normal forward playspeed, an additional parameter is added to the context for the "current time" traversal to indicates the *speed* of the local timeline. The speed is defined as a proportion of normal forward play speed. Thus a value of "1" is normal forward playspeed, a value of "2" is twice normal forward speed, and a value of "-1" is *backwards*, at the normal play speed. Note that a value of 0 is not allowed. The speed is often broken down logically into the *rate* and the *direction*. The rate is just the absolute value of the speed, and the direction is the "sign" of the speed.

Given the current computed position and speed for the timegraph, a media element with linear behavior can handle the case of non-normal playspeed with the use of fallback semantics. The fallback semantics depend upon how much or how little the media player is capable of. Some may play forwards and backwards but only at the normal rate of play, others may only handle normal forward play speed. The fallback semantics are detailed in the next sections.

Fallbacks for time filters on a media element

@@Describe the properties that the fallback semantics are preserving so that the rationale is clear

When any of the time filters are applied directly to a linear media element, the element can directly examine the attributes and apply one set of fallback semantics. These include:

- If the element cannot support acceleration and deceleration, then these attributes can be ignored. This is safe within the model, as they have no side effects. Note that this applies to acceleration and deceleration applied directly to the media element, but could also be applied to any time container that contains media elements. This is a more drastic fallback than handling the speed on each descendent of the time container, but it may be appropriate in some scenarios.
- If the media cannot play backwards (but can play at other than play speed) and the element specifies a speed less than 0, the element can play at the specified rate but in the forward direction. This will preserve the modified duration, and accommodate the player. Note that this may not be a desired approach in many cases, and so may not be a commonly used fallback.
- If the media cannot play the desired rate, the element can use one of several fallback mechanisms.
    - Just render a frozen frame for the requested simple duration. If the direction is backwards, the element could put up a still of the last frame that would be shown. This may be costly for some media types, and so the first frame of the media would be shown. This fallback approach corresponds to silence (i.e. not playing) for audio elements.
    - If the can play the computed direction but not the computed rate, it can play the closest available rate in the given direction. This fallback may be used in combination with the first approach, especially if the computed speed is far out of the range that the element can play.
    - The element can just play at normal play speed.

  In any case, the simple duration is still constrained by the computed simple duration, *as modified by* the time filters. If the computed simple duration is shorter than the intrinsic media duration at the fallback rate (usually if the speed is > 1), the media is cut short just as for a `dur` value that overrides the intrinsic media duration. If the specified simple duration is longer than the intrinsic media duration (i.e. if the rate is < 1), the media should freeze for the difference, just as it does for `dur` in SMIL 1.0. When an element must freeze, it should ideally respect the direction of the speed, using the last frame for forward speeds and the first frame for backwards speeds.

Note that the semantics of clipBegin and clipEnd are not affected (i.e. they are still respected).
The clipBegin and clipEnd semantics are always interpreted in terms of *normal forward* play speed. I.e. they are evaluated before any effects of time filters have been applied to the time model. This is consistent with the model that they can be evaluated by the media element handler, independent of the time model.

If a fallback semantic is applied to a media element and the media element has child elements (i.e. is a media time container), the local time as passed through to the children should reflect what the media actually performs. Thus if the accelerate and decelerate are ignored, the local time as passed to the child elements should not be filtered for acceleration and deceleration. This is important for cases like anchors that are timed to associate with points in the media. If an author wishes to apply acceleration to the children of the media (e.g. to accelerate a set of animation children), a wrapping <par> with the time filters applied can be inserted under the media time container. See also the next section for a general discussion of time filters and time containers.

@@ Above semantic may be overly complex. It requires that the time runtime know what the media players can actually do so that the proper timing parameters can be passed down to the children. Is this restriction worth it, since playing with a fallback will lead to a different presentation than intended anyway. The presentation visually will act funny anyway, and the entire thing is constrained by the parent time container, so it may not be as bad as noted.

The effect of these fallbacks is illustrated by some examples. In these examples, the video player is assumed to be capable only of normal forward play speed.

In the following example, the acceleration and deceleration can be safely ignored without side effect. Since they are ignored, the area child should become active 1 second after the video begins (i.e. if the video ignores the acceleration, then the effect is not applied to the children either):

```
<video accelerate=0.5 decelerate=0.5 src...>
   <area begin=1s .../>
</video>
```

In the following example, the autoReverse attribute doubles the simple duration, but the video will just play the 3 second simple duration twice, in the forward direction both times. The area child should become active 1 second after the video begins each time (i.e. if the video ignores the reverse play, then the speed change is not applied to the children either):

```
<video autoReverse="true" dur=3s src...>
   <area begin=1s .../>
</video>
```

In the following example, the speed attribute decreases the simple duration by 33% (i.e. one third). The video will play the first two seconds at normal playspeed, rather than three seconds at 1.5 times playspeed. The area child should become active 1 second after the video begins each time (i.e. if the video ignores the speed manipulation, then the speed change is not applied to the children either):

```
<video speed="1.5" dur=3s src...>
   <area begin=1s .../>
</video>
```

Fallbacks for time filters on time containers

When any of the time filters are applied to a time container element, the implementation should generally respect the time filters and deal with the effects of modified time for each media player. Nevertheless, in some situations it may be desirable to apply a fallback semantic at the time container itself.

- If the time container contains only media elements (or time containers and media elements), it may make sense to ignore the acceleration and deceleration attributes. These attributes can safely be treated as hints, and document dominated by linear media may behave better if the acceleration and deceleration are ignored. Note that while this fallback is defined for implementations, it is much more likely that authors will simply avoid the use of these attributes in the given situations.
- Time containers may not ignore speed modifications, or the autoReverse attribute. It is left to the author to avoid the use of these features in situations that are inappropriate, or in which the results will be undesirable.

@@Need more info on how time containers run backwards. The use of autoReverse is special, as it can generally use the end and begin times from the forward half. Note that time dependencies work backwards, but that event-based timing does not work correctly, and script events may not be fired. This needs more discussion. Need a reasonable means of simplifying the implementation and still allowing backwards play of fully scheduled timegraphs.

@@Examples!

More on the implementation

In the theoretical model, each element has a notion of local time that extends from 0 to some duration (the simple duration). To obtain the local time for an element in the simple case with no pacing control, an arithmetic transform is performed for the element and each ascendant time container up to the document root. This transform allows for the begin offset of the element relative to its parent time container, and for any repeat behavior of the element. The pseudo-code for this transform looks like:

@@Edit to make it produce both active and simple time? The only difference is in how the final element is handled. Also need to have a control over unconstrained versus constrained time. Sync arc calculation uses unconstrained time, but real sampling uses end-constrained time that respects freeze.

@@ Edit to use the most recent parent begin if it is active, rather than the constant repeatDuration

```
// globalToLocalTime()
//
// Recurses up to document root, and then transforms timeIn
// from parent to local time as the recursion unwinds
// Version 1: Ignores fill=freeze, assumes constant pace of time
//             assumes repeat duration is constant,
//             and assumes all times are resolved.
// Inputs:
//        timeIn - the parent simple time
// Outputs:
//        timeOut - the resulting local simple time
//
TimeInstant globalToLocalTime( TimeInstant timeIn )
{
   TimeInstant timeOut;
   Node parent = getParent();

   if( !parent.isDocumentTimeRoot() )
   {
      // We have a parent time container above us.
      // First convert timeIn to simple local time for it,
      // and then adjust for this node.
      timeOut = parent.globalToLocalTime( timeIn );
   }
   else // Our parent is the top time container
      timeOut = timeIn;

   // Adjust for begin offset. Assume "begin" has simplified begin time.
   timeOut = timeOut - this.begin; // Adjust for begin offset
   if( this.repeats() )
   {
       // Adjust for repeating simple duration.
       timeOut = timeOut % this.repeatDuration;
   }
   // timeOut is now converted to simple local time for this node
   return timeOut;
}
```

@@Next version shows how this is extended to support the time manipulations. For autoReverse, remember the edge case for the endpoint. ?Talk about the imprecision around the reverse point: interval math introduces an epsilon inaccuracy at the end, where epsilon is the sample granularity (e.g. milliseconds in our implementation).

@@How to show the update dependencies of repeatDur, segmentDur? Could just show a method to recompute, that is called on changes. Ignore all optimization issues.

Converting between local and global times

@@@ Need to discuss how to convert a time specified as a syncbase-value (and by extension a wallclock-value or an event in document or system time) to a time on the parent time container local timeline. Especially given the wallclock stuff, we need to consider the name "local time".

Define the notion of document global time, and note that it is the normalized timeline used to convert between different timespaces. Operations defined are GlobalToLocal, and LocalToGlobal. To convert from one timespace to another, simply convert the first time from local to global, and then from global to local for the second tiespace.

Basic mechanism for global to local conversion is iterate downward from the document body to the element converting the global time to a time on each time container encountered along the way. This is often implemented as a recursive algorithm, where the recursion moves from the local element up to the document body, and the work is done as the recursion "unwinds". For each time container, subtract the begin offset of the time container and then, use the remainder after dividing by the simple duration (which may vary over time - yuck!) or subtract the offset of the current repeat iteration from the begin time (better when working on begin resolution on a reset). Then, apply any filters for time manipulations (speed, acclerate/decelerate, autoReverse).

Basic mechanism for localToGlobal reverses the above algorithm. If the element is active, then the effective begin time of the current iteration of all (repeating) ancestor time containers is used when adding the begin offset. If the element is not active, then for each ancestor time container that is not active, the earliest begin time is used.

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.

Evaluation of begin and end time lists

In evaluating the list of begin values, one of two questions is asked:

- what is the earliest begin time in the list?
- what is the next begin time after a particular time?

The earliest time is used for example when an element is the target of a hyperlink activation (see also Hyperlinks and timing). The next begin time may be used by a scheduler when an element can begin more than once.

By the same token, a list of end values is evaluated for two cases:

- what is the earliest end time in the list?
- what is the next end time after a particular time?

The earliest time is used for example to calculate the simple duration of a time container defined with `endSync`. The next end time may be used by a scheduler when an element can begin more than once.

Begin and end time lists are considered in tandem. Begin times can be constrained by an end specification, as well as by the parent time container. Note that if an end time is not resolved, or is specified as "indefinite", then it is considered to be "after" al begin times. The constraint rules are:

- If a given begin time occurs after the end of the parent simple duration, the begin time is ignored and will not cause the element to begin. An unresolved or indefinite parent simple duration does not constrain any begin time.
- If at least one end time was specified, but there is no end time that occurs after a given begin time, the begin time is ignored and will not cause the element to begin. Another way to look at this is that if all values in the `end` list are resolved and occur before a begin time, then the element *does not begin*.

Thus, for a given point when an element is not active, the next begin and end time are derived from the `begin` and `end` lists as follows:

1. Find the earliest resolved begin time after the current time
   a) Discard any unresolved times
   b) Discard any times that occur before the current time
   c) Discard any times that occur after the end of the parent simple duration
   d) Take the earliest time of the remaining times (if there are none, this step fails)
2. If Step 1 fails, and if there are unresolved times, then the begin time is considered to be unresolved, and behaves as though "indefinite" had been specified. If Step 1 fails and there are no unresolved times, then this step fails.

If both Steps 1 and 2 fail, then the begin list only specifies resolved times in the past, or after the end of the parent time container. In this case, there is no (valid) begin time after the current time (although for the purposes of a hyperlink that targets the element, even an "invalid" time may be used with additional constraints - see Hyperlinks and timing).

If a begin time is found, then it must be checked against the list of end times. If no `end` attribute is specified, the end is "indefinite", and any begin found above is valid. If any end values are specified, the list is evaluated as follows:

1. Find the earliest resolved end time after the candidate begin time
   a) Discard any unresolved times
   b) Discard any times that occur before the begin time
   c) Take the earliest time of the remaining times (if there are none, this step fails)
2. If Step 1 fails, and if there are unresolved times, then the end time is considered to be unresolved, and behaves as though "indefinite" had been specified. If Step 1 fails and there are no unresolved times, then all end times are resolved and occur before the candidate begin time. In this case the candidate begin time is rejected, and the element does not begin.

If a valid begin and end value are found, the end value found with the rules above is the value used in Computing the Active Duration .

When it is specified that the element *does not begin*, begin and end events will not be raised in the DOM, and time dependents defined relative to the begin or end of this element will not be activated.

In contrast to the rules above, if a `beginElement()` or a `beginElementAt()` call specifies a begin time after the last end time (with no unresolved end times), the active duration is indefinite, as though end had been defined as "indefinite". Note that if `beginElement()` or `beginElementAt()` is called when the parent time container is not active, the method call will have no effect. If `beginElementAt()` is called and specifies a time after the end of the parent simple duration, the method call will have no effect. See also Supported methods .

Hyperlinks and timing

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 indeterminate and interactive timing, nor do they handle author-time restart restrictions. Here we extend SMIL 1.0 semantics for use in presentations using elements with indeterminate timing, interactive timing, and author-time restart restrictions.

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 (current) begin time of the element. If there are multiple begin times, use the begin time that corresponds to the current "begin instance".
2. Else if the target element begin time is resolved (i.e. there is any resolved time in the list of begin times, or if the begin time was forced by an earlier hyperlink or a `beginElement()` method call), seek the document time (forward or back, as needed) to the earliest resolved begin time of 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. element begin time is unresolved), 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 is active, seek the document time back to the (current) begin time of the element. If there are multiple begin times, use the begin time that corresponds to the current "begin instance".
    2. Else if the begin time is resolved, seek the document time (forward or back, as needed) to the earliest resolved begin time of the target element.
    3. 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.

In the above rules, the following additional constraints 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 the parent simple duration, then the seek-to time is *clamped* to the end time of the parent time container simple duration. This constraint is applied recursively for all ascendant time containers.

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.

After seeking a document forward, the document should be in 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). In particular, seeking the presentation time forward should also begin any other elements that have resolved begin times between the current time and the seeked-to time. The elements that are begun in this manner may still be active, may be frozen, or may already have ended at the seeked-to time. If an element has ended, it logically begins and ends during the seek. The associated DOM events are raised, and all time dependents are updated. Also any elements currently active at the time of hyperlinking should "fast-forward" over the seek interval. These elements may also be active, frozen or already ended at the seeked-to time. The net effect is that seeking forward to a presentation time puts the document into a state identical to that as if the document presentation time advanced undisturbed to reach the seek 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 during the seek interval and will turn off any elements that are resolved to begin at a time after the seeked-to time. Note that resolved begin times (e.g. a begin associated with an event) are not cleared or lost by seeking to an earlier time. Note further that 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. Subject to the rules above for hyperlinks that target timed elements, hyperlinking to elements with resolved begin times will function normally, advancing the presentation time forward to the previously resolved time. When the document seeks backwards before a resolved begin for an element time, this does not reset the element

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. 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). For example:

```
<par begin="0">
   <img id="A" begin="10s" .../>
   <img id="B" begin="A.begin+5s" .../>
   <img id="C" begin="click" .../>
   <img id="D" begin="C.begin+5s" .../>
   ...
   <a href="#D">Click here!</a>
</par>
```

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 "C" is clicked at 25 seconds into the presentation. The click on "C" in turn resolves "D" to begin at 30 seconds. From this point on, traversing the hyperlink will cause the presentation time to be seeked to 30 seconds.

If at 60 seconds into the presentation, the user again clicks on "C", "D" will become re-resolved to a presentation time of 65 seconds. Subsequent activation of the hyperlink will result in the seeking the presentation to 65 seconds.

If the time container were defined to repeat, or could restart, then all indeterminate times for children of the time container *are cleared*(reset to "indefinite") when the parent time container repeats or restarts. See also Resetting element state .

Implications of beginElement() and hyperlinking for seq and excl time containers

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. 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 as the intervening elements are activated and deactivated.

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).

@@The above semantic keeps things simple, although it may point up a need for a seekToElement() method in the DOM interface that mimics the hyperlink functionality.

Note that if a hyperlink targets (or if `beginElement()` or `beginElementAt()` is called for) an element **A** defined to begin when another element **B** ends, and the other element **B** has (e.g.) an event-base or syncbase end, the hyperlink or method call will not end element **B**. It will only active 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.

@@ What if the target element is in an excl and is in a priorityclass that is defined to be deferred if it tries to interrupt the current (higher priority) element? Do we overrule the priorityClass rules, and just stop or pause the running element?

Note that the presentation agent need not actually prepare any media for elements that are seeked over, but it does need to propagate the sync behavior to all time dependents so that the effect of the seek is correct.

Propagating changes to times

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 - both the begin and active end time of an element can change when it restarts. Since the begin and active end times of one element can be defined relative to the begin or active end of other elements, any changes to times must be propagated throughout the document.

When an element "foo" has a begin or active end time that specifies a syncbase element (e.g. "bar" as below):

```
<img id="foo" begin="bar.end" .../>
```

we say that "foo" is a *time-dependent* of "bar" - that is, the "foo" begin time depends upon the active end of "bar". Any changes to the active end time of "bar" must be propagated to the begin of "foo". The effect of the changes depends upon the state of "foo" when the change happens. The rest of the section describes the specific rules for propagating changes.

Note that it is possible for the syncbase element "bar" to end *again*, if it is restarted. When "bar" restarts, the a new end time is calculated and all time dependents are notified of the change. For example:

```
<img id="foo" begin="0" end="bar.end" .../>
<img id="bar" begin="btn.click" dur="5s" .../>
```

Element "foo" will end when "bar" ends, however "bar" can restart on another click. When "bar" restarts, a new end is calculated, and "foo" is notified. However, as "foo" will not restart, the change is ignored. A variant on this illustrates a case when the time change does propagate through:

```
<img id="foo" begin="0" end="bar.end+10s" .../>
<img id="bar" begin="btn.click" dur="5s" .../>
```

Element "foo" will end 10 seconds after "bar" ends. If "bar" is restarted within 10 seconds of when it first ended, "foo" will still be active, and the changed end time will propagate through. Using example times, if the user clicks on the "btn" element 8 seconds after the parent time container begins, "bar" begins at 8 seconds and will end at 13 seconds. Element "foo" would then end at 23 seconds. If the users clicks "btn" again 3 seconds after "bar" ends, (i.e. at 16 seconds), the end of "bar" now has the value of 21 seconds. This change propagates to "foo", and "pushes out" the end of "foo" until 31 seconds.

The rule is that once an element has ended its active duration, changes that affect its end time are ignored (within the current simple duration of the parent time container).

When an element restarts (or when an ascendant time container repeats or restarts), all child times are recalculated, and may again become indefinite. For example:

```
<img id="foo" begin="btn.click" end="mouseout" .../>
<img id="bar" begin="btn.click" end="foo.end"  .../>
```

Both elements will start when the "btn" element is first clicked. Element "foo" will end when "mouseout" is raised on the img. At this point, the active duration of "bar" will become defined (resolved), and "bar" will end the active duration. If the user clicks on the target element again, both elements will restart, and "bar" will once again have an indefinite active duration.

@@Add additional example with explanation. Note that if user clicks at 4s, image1 is never seen. The resolution of image end happens and sticks when image1 begin is evaluated.

```
<par>
   <img id="image3" dur = "24s" end="user.click"/>
   <img id="image1" begin ="10s" end="image3.end"/>
</par>
```

Handling negative offsets

The use of negative offsets to define begin times merely defines the synchronization relationship of the element. If does not in any way override the time container constraints upon the element, and it cannot override the constraints of presentation time.

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 begin="-5s" dur="30s" src="movie.mpg" />
</par>
```

The `video` element cannot begin before the `par` begins. The begin is simply defined to occur *"in the past"* when the `par` begins.

A begin or 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 (eventually) resolved, the dependent time that is computed with a negative offset may occur in the past. The computed time defines the *scheduled synchronization relationship* of the element, even if it is not possible to begin or end the element at the computed time.

When a begin time is defined to be in the past, 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 media will actually begin at the time computed according to the following equation:

```
Let o be the offset value, d is the simple duration, AD is the active duration.
If AD is indefinite, it compares greater than any value of o or ABS(o).
REM( x, y ) is defined as x - floor( x/y ).
If y is indefinite, REM( x, y ) is just x.
```

```
If ABS(o) >= AD the element does not begin.

Else the element media begins at REM( ABS(o), d ).
```

If the element repeats, the iteration value of the `repeat` event has the calculated value based upon the computed begin, and not the observed number of repeats. Thus for example:

```
<ref begin="foo.click-8s" dur="3s" repeatCount="10" .../>
```

The element begins when the user clicks on the element "foo". It begins to play at 2 seconds into the 3 second simple duration. Any time dependents are activated relative to the computed begin time, and not the observed begin time. The `begin` event is raised when the element begins, but has a `timeStamp` value that corresponds to the defined begin time, 8 seconds earlier. One second later, the element will repeat, and the associated `repeat` event will have the iteration value set to 3 (it is zero based). The element will end 22 seconds after the click.

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.

Behavior of 0 duration elements

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 alsoEvaluation of begin and end time lists).

Whether or not media is retrieved and/or rendered is implementation dependent.

Resetting element state

When a time container repeats or restarts, all descendent children are "reset" with respect to certain state:

1. Any event-base times that were resolved are reset to unresolved (equivalent to "indefinite").
2. Any syncbase times are reevaluated (i.e. the translation between timespaces must be recalculated - see Converting between local and global times).
3. Any state associated with the interpretation of the `restart` semantics is reset. Thus, for example if an element specifies `restart="never"`, the element can begin again after a reset. The `restart="never"` setting is only defined for the extent of the parent time container simple duration.

When an element restarts, the rules 1 and 2 are also applied to the element itself, although the rule 3 (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 *or paused* children, the pause queue for the <excl> is cleared.

10.3.6 Controlling runtime synchronization behavior

Proposed new support in SMIL Boston 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.

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 for 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 thought the element were defined with scheduled timing.

The syncBehavior attribute is subordinate to any sync relationships defined by time containers, sync arcs, event arcs, etc. The syncBehavior attribute has no bearing on the formation of the time graph, only the enforcement of it.

@@ Need to address how syncBehavior will interact with restart semantics. In particular, do we re-establish the sync relationship when it restarts (this is my first guess, assuming that restart is allowed). syncBehavior is not supposed to define all the behavior of the element, but rather just the behavior when there is some problem with sync, or when the user pauses or seeks an element. E.g. we do not require that children of seq elements be locked, but we do require that the seq semantics be maintained. If restart is allowed, then that should be orthogonal to the syncBehavior. Note that all other aspects of timing (e.g. repeat, parent constraints and event-based timing override the syncBehavior, so I think we have a precedent.

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.

@@ We need to move the general definition of the SMIL Default stuff elsewhere, and just specify the possible arg values here. Ideas where it should go?

The value of the syncBehavior and syncTolerance attributes are not inherited, but it is possible to set default behavior for elements or time containers using the SMILdefault syntax below.

The default value for syncTolerance is implementation dependent, but should be no greater than two seconds.

Sync behavior attributes

syncBehavior
: Defines the runtime synchronization behavior for an element.
: Legal values are:

canSlip
: Allows the associated node to slip with respect to the parent time container. When this value is used, any syncTolerance attribute is ignored.

locked
: Forces the associated node to maintain sync with respect to the parent time container. This can be eased with the use of the syncTolerance attribute.

syncTolerance
: This attribute on timed elements and time containers defines the synchronization tolerance for the associated element. It has an effect only if the element has `syncBehavior="locked"`. This allows a locked sync relationship to ignore a given amount of slew without forcing resynchronization. resolution.
: Legal values are Clock-values .

SMILdefault
: Defines the default value for the runtime synchronization behavior for an element and all descendents, or the default synchronization tolerance for the associated element.
: Legal values are:

`syncBehavior:canSlip`
: Allows the associated node to slip with respect to the parent time container. When this value is used, any syncTolerance attribute is ignored.

`syncBehavior:locked`
: Forces the associated node to maintain sync with respect to the parent time container. This can be eased with the use of the syncTolerance attribute.

`syncTolerance: Clock-value`
: Defines the default value for the synchronization tolerance for an element, and all descendents.

Sync master support

An additional proposed 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 players 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 player 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 player 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.

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. If more than one element within the syncBehavior scope has the syncMaster attribute set to true, and the elements are both active at any moment in time, the behavior will be implementation dependent.

syncMaster
: Boolean attribute on media elements and time containers that forces the time container playback to sync to this element.

  The default value is false.
  The associated property is read-only, and cannot be set by script.

10.3.7 Common syntax DTD definitions

@@ Need to decide whether endSync belongs on media elements (with timed children) or not.

Timing attributes

```
<!ENTITY % timingAttrs
  begin          CDATA   #IMPLIED
  dur            CDATA   #IMPLIED
  end            CDATA   #IMPLIED
  restart        (always | never |
                   whenNotActive)   "always"
  repeatCount    CDATA   #IMPLIED
  repeatDur      CDATA   #IMPLIED
  fill           (remove | freeze | hold) "remove"
>
```

Runtime sync behavior attributes

```
<!ENTITY % runtimeSyncBvrAttrs
  syncBehavior          (locked | canSlip) #IMPLIED
  defaultSyncBehavior   (locked | canSlip) "canSlip"
  syncTolerance         CDATA              #IMPLIED
  defaultSyncTolerance  CDATA              #IMPLIED
  syncMaster            (true | false)     "false"
>
```

Time container elements

```
<!ELEMENT par ???>
<!ATTLIST par
  %timingAttrs
  %runtimeSyncBvrAttrs
  id              ID      #IMPLIED
  endSync         CDATA   #IMPLIED
>

<!ELEMENT seq ???>
<!ATTLIST seq
  %timingAttrs
  %runtimeSyncBvrAttrs
  id              ID      #IMPLIED
>

<!ELEMENT excl ???>
<!ATTLIST excl
  %timingAttrs
  %runtimeSyncBvrAttrs
  id              ID      #IMPLIED
  endSync         CDATA   #IMPLIED
>
```

10.4 Integrating SMIL Timing and Synchronization into a host language

@@When this settles down, fill in subsections in TOC

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

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.

The host language designer must define what "presenting a document" means. A typical example is that the document is displayed on a screen.

The host language designer must define the *document begin*. Possible definitions are that the document begins when the complete document has been received by a client over a network, or that the document begins when certain document parts have been received.

The host language designer must define the *document end*. This is typically when the associated application exits or switches context to another document.

@@ Check to see if we really have issues with this, e.g. for specifying floating point values. Can we use this in our definitions of offset value?

The host language must specify the formats supported for numeric attribute values. This includes integer values and especially floating point values for attributes such as `keyTimes` and `keySplines`. As a reasonable minimum, host language designers are encouraged to support the format described in [CSS2]. The specific reference within the CSS specification for these data types is 4.3.1 Integers and real numbers.

10.4.2 Required definitions and constraints on element timing

@@ Need to talk about specifying which elements can be timed, and what it means to time them.

The set of elements that may have timing includes ??? elements defined in host languages.

Supported events for event-base timing

The host language must specify which event names are legal in event base values. If the host language defines no allowed event names, event-based timing is effectively precluded for the host language.

Host languages may specify that dynamically created events (as per the [DOM2Events] specification) are legal as event names, and not explicitly list the allowed names.

10.4.3 Error handling semantics

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.

@@Broken link to Handling errors - Do we need a section on this?

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.

10.4.4 SMIL Timing and Synchronization namespace

Language designers can choose to integrate SMIL Timing and Synchronization as an independent namespace, or can integrate SMIL Timing and Synchronization names into a new namespace defined as part of the host language. Language designers that wish to put the SMIL Timing and Synchronization functionality in an isolated namespace should use the following namespace:

@@ URI to be confirmed by W3C webmaster

http://www.w3.org/2000/TR/smil20

10.5 Document object model support

Any XML-based language that integrates SMIL Timing will inherit the basic interfaces defined in DOM [DOM2] (although not all languages may require a DOM implementation). SMIL Timing specifies the interaction of timing functionality and DOM. SMIL Timing also defines constraints upon the basic DOM interfaces, and specific DOM interfaces to support SMIL Timing.

Much of the related SMIL-DOM functionality is proposed in the [SMIL-DOM] section. We may need to go into further detail on the specific semantics of the interfaces - the sections below are placeholders.

10.5.1 Element and attribute manipulation, mutation and constraints

Define rules on element and attribute access (inherit from and point to Core DOM docs for this). Define mutation constraints. This is currently covered in the [SMIL-DOM] section.

10.5.2 Event model

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.

The specific events supported are defined by the host language. If no events are defined by a host language, event-timing is effectively omitted.

The [SMIL-DOM] section defines the initial set of time-related events that have been proposed.

10.5.3 Supported methods

SMIL Timing supports two methods for controlling the timing of elements: `beginElement()` and `endElement()`. These 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()` and `endElement()` 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.

Calling `beginElement()` causes the element to begin in the same way that an element with event-based begin timing begins. The effective begin time is the current presentation time at the time of the DOM method call. Note that `beginElement()` is subject to the `restart` attribute in the same manner that event-based begin timing is. If an element is specified to disallow restarting at a given point, `beginElement()` methods calls must fail. Refer also to the section Restarting elements .

Calling `beginElementAt()` causes the element to begin in the same way that an element begins with event-based begin timing *that includes an offset*.

- If the offset passed to `beginElementAt()` is positive, then the element will be restarted (subject to the `restart` attribute semantics) at the specified offset into the future from the current time. If the specified time is past the end of the parent time container simple duration, the element may be stopped by the restart semantics and yet not restart (due the the time container constraints).
- If the offset passed to `beginElementAt()` is negative, then the element is restarted in the same manner (and subject to the same constraints) as for the `beginElement()` method. However, the element is begun as though it had begun at the earlier offset. For details see Handling negative offsets .

Calling `endElement()` causes an element to end the active duration, just as `end` does. Depending upon the value of the `fill` attribute, the element effect may no longer be applied, or it may be frozen at the current effect. Refer also to the section Freezing elements . If an element is not currently active (i.e. if it has not yet begun or if it is frozen), the `endElement()` method will fail.

Interfaces are currently defined in the [SMIL-DOM] section.

10.6 Glossary

10.6.1 General concepts

The following concepts are the basic terms used to describe the timing model.

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.

Note that this definition is preliminary.

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.

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.

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.

Events and interactive timing

Begin and active end times in SMIL Boston 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.

More information on the supported events and the underlying mechanism is described in the DOM section of this draft [SMIL-DOM].

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.

Note that this definition is preliminary. The name may also change.

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-ref or event name, or logically with special syntax.

Note that this definition is preliminary.

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.

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 Boston 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.

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.

Time manipulations

Time manipulations allow the element's time (within the simple duration) to be filtered or modified. For example the speed of time can be varied to make the element play faster or slower than normal. The filtered time affects all descendents of the element. Several time manipulations are proposed for SMIL Boston. Time manipulation is primarily intended to be used with animation [SMIL-ANIMATION] (W3C members only).

Note that any time manipulation that changes the effective play speed of an element's time may conflict with the basic capabilities of some media players. The use of these manipulations is not recommended with linear media players, or with time containers that contain linear media elements, such as streaming video.

There are a number of unresolved issues with this kind of time manipulation, including issues related to event-based timing and negative play speeds, as well as many media-related issues.

Determinate and indeterminate schedules

Using simple, scheduled timing, a time graph can be described in which all the times have a known, defined sync relationship to the document timeline. We describe this as *determinate* timing.

When timing is specified relative to events or external clocks, the sync relationship is not initially defined. We describe this as *indeterminate* timing.

A time is *resolved* when the sync relationship is defined, and the time can actually be scheduled on the document time graph.

Indeterminate times that are event-based are resolved when the associated event occurs at runtime - this is described more completely in the section Unifying Scheduling and Interactive Timing . Indeterminate times that are defined relative to external clocks are usually resolved when the document playback begins, and the relationship of the document timeline to the external clock reference is defined.

A determinate time may initially be unresolved, e.g. if it is relative to an unknown time such as the end of a streaming MPEG movie (the duration of an MPEG movie is not known until the entire file is downloaded). When the movie finishes, determinate times defined relative to the end of the movie are resolved.

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 Boston 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.

10.7 Appendix A: Annotated examples

10.7.1 Example 1: Simple timing within a Parallel time container

This section includes a set of examples that illustrate both the usage of the SMIL syntax, as well as the semantics of specific constructs. This section is informative.

Note: In the examples below, the additional syntax related to layout and other issues specific to individual document types is omitted for simplicity.

All the children of a <par> begin by default when the <par> begins. For example:

```
<par>
   <img id="i1" dur="5s"  src="img.jpg" />
   <img id="i2" dur="10s" src="img2.jpg" />
   <img id="i3" begin="2s" dur="5s" src="img3.jpg" />
</par>
```

Elements "i1" and "i2" both begin immediately when the par begins, which is the default begin time. The active duration of "i1" ends at 5 seconds into the <par>. The active duration of "i2" ends at 10 seconds into the <par>. The last element "i3" begins at 2 seconds since it has an explicit begin offset, and has a duration of 5 seconds which means its active duration ends 7 seconds after the <par> begins.

An image that illustrated the timeline might be useful here.

10.7.2 Example 2: Simple timing within a Sequence time container

Each child of a <seq> begins by default when the previous element ends. For example:

```
<seq>
   <img id="i1" begin="0s" dur="5s" src="img1.jpg" />
   <img id="i2" dur="10s" src="img2.jpg" />
   <img id="i3" begin="1s" dur="5s" src="img3.jpg" />
</seq>
```

The element "i1" begins immediately, with the start of the <seq>, and ends 5 seconds later. Note: specifying a begin time of 0 seconds is optional since the default begin offset is always 0 seconds. The second element "i2" begins, by default, 0 seconds after the previous element "i1" ends, which is 5 seconds into the <seq>. Element "i2" ends 10 seconds later, at 15 seconds into the <seq>. The last element, "i3", has a begin offset of 1 second specified, so it begins 1 second after the previous element "i2" ends, and has a duration of 5 seconds, so it ends at 21 seconds into the <seq>.

Insert illustration.

10.7.3 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"><img src="Button1.jpg"/></a>
  <a href="p2"><img src="Button2.jpg"/></a>
</par>
```

   This example models jukebox-like behavior. Clicking on the first image 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).

   Shouldn't we say, here, exactly where the elements of the selected par in the excl should begin when a click happens, e.g., if we are 10 seconds into the outer par and we click on button 2, does the MPG video in p2 start 10 seconds into its stream (in-sync), or does it start at its time 0?

2. Exclusive element combined with event-based activation:

   Note that the specific syntax for beginEvent argument values is still under discussion.

```
<par>
  <excl>
    <par begin="btn1.click">
       ...
    </par>
    <par begin="btn2.click">
       ...
    </par>
  </excl>
  <img id="btn1" src=... />
  <img id="btn2" src=... />
</par>
```

   The same jukebox example, using event-based activation.

In these two examples event-based and anchor-based activation look almost identical, maybe we should come up with examples showing the difference and the relative power of each.

3. Exclusive element using determinate declarative 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 determinate syntax is allowed, this is not expected to be a common use-case scenario.

Issue - should we preclude the use of determinate timing on children of excl? Other proposals would declare one child (possibly the first) to begin playing by default. Proposals include an attribute on the <excl> container that indicate one child to begin playing by default.

10.7.4 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 indefinite. This can lead to surprising results, as in this example:

```
<seq>
   <img src="img1.jpg" />
   <video src="vid2.mpg" />
   <video src="vid3.mpg" />
</seq>
```

The default syncbase of a sequence is defined to be the effective active end of the previous element in the sequence, unless the active duration is indefinite in which case the default syncbase is the *begin* of the previous element. In the example, the implicit duration of the image is used to defined 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.

10.7.5 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 specified. The attribute end on the other hand overrides the active duration of the element. If the element does not have repeat specified, the active duration is the same as the simple duration. However, if the element has repeat specified, then the end will override the repeat, but will not affect the simple duration. For example:

```
<seq repeat="10" end="stopBtn.click">
   <img src="img1.jpg" dur="2s" />
   <img src="img2.jpg" dur="2s" />
   <img src="img3.jpg" dur="2s" />
</seq>
```

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.

10.7.6 Example 6: SMIL-DOM-initiated timing

When an implementation supports the SMIL-DOM, 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 SMIL-DOM `beginElement()` method is called.

10.8 Appendix B: Authoring guidelines (to be added)

This is a placeholder for a set of authoring guidelines intended to help authors avoid potential mistakes and confusion, and to suggest best practices as intended by the authors.

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.

11. Integrating SMIL Timing into Other XML-Based Languages

Editors:
Erik Hodge *(ehodge@real.com)* (RealNetworks)
Warner ten Kate *(warner.ten.kate@philips.com)* (Philips Electronics)

11.1 Abstract

This segment of the working draft specifies an architecture for applying timing information to XML documents. It specifies the syntax and semantics of the constructs that provide timing information. This approach builds on SMIL by preserving SMIL's timing model and maintaining the semantics of SMIL constructs.

The two non-In-Line Timing paradigms mentioned in this section of the working draft, namely CSS Timing and Timesheets, have not been given as much consideration by the SYMM Working Group as has In-Line Timing. The Working Group will continue to concentrate on solidifying In-Line Timing before it revisits other possible methods of adding timing such as CSS Timing and Timesheets.

11.2 Introduction

Currently there exists no standardized method for adding timing to elements in any arbitrary XML document. This segment of the working draft defines the mechanisms for doing so.

11.2.1 Background

Prior to SMIL 1.0 becoming a W3C recommendation, a significant number of W3C members expressed interest in integrating SMIL timing functionality with XML-based languages such as [XHTML10].

SMIL 1.0 describes timing relationships between objects, including complete XML documents. SMIL 1.0 can not control the timing of individual elements contained within these documents, e.g., the display of a single [XHTML10] heading before the bulk body text appears, or the sequential display of the items of a group in an [SVG] document. When using SMIL 1.0 for this, a content author is forced to contain each temporal element set in a separate document, leading to very small documents in some cases.

As another example, consider the split up of text that would occur when creating closed captioning from a subtitle track using SMIL 1.0 if the text was in raw-text or HTML form, two standard text data types that do not contain native timing. Using SMIL timing, a text data type could be developed that would handle the presentation of each caption as contained within one file. The SMIL file would then only have to reference that one stream.

The SMIL 1.0 architecture assumes that SMIL documents will be played by a SMIL-based presentation environment. It does not treat the case where timing is an auxiliary component, and the presentation environment is defined by another language, like [XHTML10], a vector-graphics language like [SVG], or any user-defined XML-based language and stylesheet.

This segment of the working draft specifies how SMIL timing can be used in other XML languages, providing a solution to the above cases. This version of this segment of the working draft only concentrates on In-line Timing; future versions may include concepts like CSS Timing, where SMIL timing would be handled using CSS, and possibly other methods of externally adding timing to a document. The work is driven by the following goals:

- *Goal 1:* To provide a solution for integrating timing into XML languages. The XML language can be user defined.
- *Goal 2:* To base that solution on SMIL. In case SYMM settles on a syntax different from SMIL, e.g., a CSS-based solution, the semantics, names, values, functionality and the timing model should be preserved as used by SMIL.
- *Goal 3:* To strive for a single solution, rather than multiple alternatives. Duplication of functionality is allowed only if it is well-justified. One example of a well-justified deviation from this goal is the inclusion of some form of external timing which would be necessary in such cases where a document to which the timing is being added can not be altered due to copyright or other restrictions. This would necessitate the existence of two methods for adding timing if in-line timing is allowed.

11.2.2 Use cases

The following cases require the application of timing. These use cases are not listed in any particular order:

1. **Change media presentation over time.**
   Various media objects contained in or referenced in an XML-based document are made to appear and disappear over time. Note: the media can be any element that models content: a video, a paragraph, a database record, etc. An example is a series of images and captions timed to appear as a slide show.
2. **Schedule the presentation of elements of a Web page.**
   E.g., an HTML[*] page could be made to change over time by controlling the display of its elements.

   [*] Note: This assumes that the HTML document is a valid XML 1.0 [XML10] document.
   - **Use case 2A:** Add in-line timing to an `<H1>` element of an [XHTML10] document to schedule the display of that header's text.
   - **Use case 2B:** Add timing to the existing structure of an [XHTML10] document in order to schedule the presentation of the elements of a list or of sections of the document. There are many ways that elements of a list or the sections of a document could be presented. A common way might be to

have the elements' content appear one after the other with the prior element's content remaining displayed at least until the list is completely displayed. Another way might be to have each list item display for a period of time and then be replaced, spatially, by the next.
Consider the script of a play where each line of dialog is within a <P>...</P> container. Such a document could be turned into a textual performance of the play by adding the timing necessary to sequentially present each of the child <P> elements of the <BODY> of the document.
- **Use case 2C:** Add timing to a document where the timing is independent of the structure of that document. Consider the following example:
Assume a human body display language. In this example different parts appear and disappear in different combinations at different times regardless of the content structuring, i.e., regardless of the order of the data in the document body. The document DTD uses the human structure: human = { face, torso, 2 arms, 2 legs }. A leg has a thigh, knee, calf and foot. Etc. The document merely describes the structure of the human form.

3. **Add timing to an immutable document.**
Without modifying the original content document due to copyright and/or other issues, apply an external timing document to that content document. In some cases, timing will be applied externally to elements based on the names of their XML mark-up tags, while in other cases timing will be applied externally to elements of certain classes or to individual elements based on their unique IDs. For example, The Daisy Consortium's "talking book" applications use HTML documents containing the text of a book whose pages are marked with <SPAN> elements containing unique IDs. An external timing document could then be used to apply unique timing to each of these <SPAN> elements.
4. **Add timing to links.**
Links could be made to be active only during certain periods.
Note: this can already be done within a SMIL 1.0 document.
5. **Change the appearance of graphical objects over time.**
For example, add timing to elements of a graphical display so that individual graphical elements appear, disappear, and move in front of and behind each other over time.
6. **Change the style, as opposed to the visibility, of textual data over time.**
For example, make something appear red for 5 seconds and then yellow for the remainder of its duration.

11.2.3 Assumptions

1. The XML language to which the timing is applied can be of any type. The language can be:
    - presentation agnostic
    - presentation oriented
2. For CSS or other stylesheet-specified timing, the XML language must be able to cooperate with a stylesheet. The style language used is assumed to be CSS or another style language like the Extensible Style Language [XSL].

Assumptions that may need further refinement

1. The XML language of the document can cooperate with the document's Document Object Model (DOM), if one exists.
2. If the full document, made up of the content document plus any external (non-in-line) timing, is exposed to the content document's DOM, that DOM models the data along the tree as spanned in the body.

11.2.4 Requirements

1. Should follow the SMIL time model as it evolves.
2. Should be compatible/interoperable with SMIL as specified in the goal number two .
3. Should be possible to apply the timing model to any XML language.
4. Should enable timing of styles as specified in the stylesheet accompanying the XML document.
5. Should cooperate with events as specified by the content document's DOM.
6. Any (allowed) mutations should be reflected appropriately in the time model, in a dynamic manner. For example, if a media element's begin time is based on the end time of another media element that has ended early, the former should begin right away rather than wait until its originally-scheduled begin time is reached.
7. These requirements only apply to Timing methods other than In-Line Timing:
    1. Should enable authoring across documents, e.g., temporal specification may be separated from the content document.
    2. Should enable construction of temporal templates, such that timing styles can be developed and taken as an authoring basis for further refinement. The precedence rules are the same as for CSS.

11.3 Framework

This section outlines the conceptual approach to adding timing to XML languages. The Specification section specifies the constructs used. There are several methods of adding this timing, but this version of this segment of the working draft considers only method number 1, below, in any detail. Note that the second and third methods will require considerable refinement and are only mentioned in this document to show their potential for adding timing in cases where doing so using the first method is either not possible or is less efficient:

1. Through In-line Timing . In-line timing is simply the addition of timing syntax into a content document to schedule the presentation of its objects. This does *not* include any timing done through CSS or other style sheet timing methods even if the style sheet is "in-line", i.e., exists within the XML document to which it applies. "In-line" in this section of the working draft refers only to adding SMIL timing attributes to XML elements as well as adding SMIL time container elements to the body of the content XML document.

2. Future Methods Under Consideration:
   - Through Cascading Style Sheet (CSS) Timing . CSS Timing would treat timing as style attributes and would allow the application of these timing style attributes to elements of the content of a document in the same way that [CSS1] and [CSS2] currently allow other styles (e.g., color, spacing) to be applied to the content. CSS Timing may be added to the content document or may be contained in an external document that is referenced by the content document.
   - Through Timesheets . Timesheets are a new concept under development that apply timing to elements in the content document. The order of the items in a timesheet determines the order of presentation of the referenced content elements. Unlike CSS Timing, a Timesheet separates timing from the content document's structure. A timesheet may be added to the content document, may be contained in an external document that is referenced by the content document, or may be a document that references the content document external to itself.

How to ensure that In-line Timing cooperates uniformly with CSS Timing or Timesheets is still under consideration.

In cases where SMIL timing is placed within an XML document, a hybrid DTD may be needed containing the DTD for the SMIL Timing and Synchronization module as well as the DTD for the XML language in which the original content document was written.

11.3.1 Framework: In-line Timing

In some cases In-line Timing will make authoring easier, especially in cases where the author wants the timing to flow with the structure of the content. In other cases, CSS Timing or Timesheets may be needed.

The semantics of In-line Timing are the same as that of SMIL Boston timing, but the syntax can differ. This module defines two ways to add In-line Timing to XML content. These two methods may be used in combination:

1. The first is to add SMIL time container elements `<par>...</par>`, `<seq>...</seq>`, and `<excl>...</excl>` to create time blocks that apply timing to all child elements. Legal SMIL Boston timing attributes, such as `begin` , `end`, and `dur`, could be added to these elements as well as to the resultant child elements. For instance, an author could place a `<seq>` element as a parent of a list of items, then add `dur="5s"` to each list item element, and consequently make those list items display one after the other for five seconds each.
2. The second way is to add timing container functionality to some of the existing XML mark-up elements. This has the advantage of not requiring the use of an additional element that might make it harder to manage the layout and other behavior of the document.
   Essentially, and element is made to act as a parent `par`, `seq`, or `excl`, and

may also contain optional SMIL timing attributes like duration, begin time (relative to that of any parent element), and end time, to name a few. In order to declare that an element should act as a time container, a new attribute, "timeContainer", is defined. This attribute is only legal within grouping elements in XML documents, and specifically cannot be applied to any of the time container elements including par, seq and excl. The use of this attribute in a document does not preclude the use of par, seq, and excl elements within that same document. A language designer may place additional constraints on the elements that can support the "timeContainer" attribute. Children of an element with this attribute have the same semantics as children of the respective time container elements as specified in the SMIL Timing module of this specification.

This example adds timing to an [XHTML10] <DIV> element so that it acts as a <par> SMIL time container and has a duration of display of 10 seconds: `<div timeContainer="par" dur="10s">`.

Besides adding timing to the display of objects within an XML document, varying styles like color and location over time may also be desired. This can be done two ways:

1. By using a new attribute called "timeAction". This attribute is the action associated with the timing. This attribute would allow the author to specify how the element's timing should be applied, e.g., to the display of its content or to style attributes like the color of its content. In SMIL 1.0, the begin, end, duration, and other times specified in elements are always used to place the element on its parent element's time line. This new attribute, timeAction, was created to allow alternate application of the specified time values, e.g., the begin time could be applied to a style like the color of an element without affecting the true begin time of the element. For example, the following would make an [XHTML10] paragraph appear red for 5 seconds and then black for the remainder of its duration. This example assumes that CSS class "redText" is defined for the document:

```
<span class="redText"
   timeAction="class" dur="5s">This text will be red for 5 seconds and then
  black thereafter</span>
```

   The legal values for timeAction must be specified by the host language.
2. By using SMIL Animation [SMIL-ANIMATION]. Like timeAction, this allows timing to be applied to elements' attributes such as style. Unlike timeAction, SMIL animation allows for timing to be applied to multiple attributes of an element. For example, making the contents of an [XHTML10] paragraph be black for five seconds and then red for five seconds (at times relative to the parent's time line) while at the same time setting the duration of the display of that paragraph to 20 seconds, could be done as follows. This example assumes that CSS classes "blackText" and "redText" are defined for the document. Note that class takes a string, thus a calcMode of discrete applies. The animation will set the fontStyle to "blackText" for 5 seconds (half the simple duration) and

then set the fontStyle to "redText" for the remaining 5 seconds of the animate element's duration:

```
<p class="blackText" dur="20s">
  <animate attributeName="class" from="blackText"
          to="redText" dur="10s"/>
  This text appears in black for five seconds, then changes to
  red for five more seconds.  It changes back to black when the
  animate element's duration is reached at 10 seconds because
  the default fill is "remove" for the animation.  The dur of
  the p element applies to the display of the paragraph, not to
  the style.
</p>
```

Here is another, more-detailed example of In-line Timing being used to schedule the application of color style attributes as specified in the XML document's style sheet: Consider the playback of a music album where the audio track plays in concert with a list of the songs. Timing is added to the list so that the song that is currently playing is colored differently from the others. A `<set>` element from SMIL Animation in this example is used to set the style of the class "playing" (only) to the text during the time specified. Note that, in this example, the text of the paragraphs, namely "song 1", "song 2", and "song 3", all appear throughout the entire presentation; it is only their color that has been modified over time using (in-line) timing:

```
<head>
  <style>
    .stopped { color: black; }
    .playing { color: red; }
  </style>
</head>
<body timeContainer="par">
  <seq>
      <audio id="song1" src="song1.au" />
      <audio id="song2" src="song2.au" />
      <audio id="song3" src="song3.au" />
  </seq>
  <p class="stopped">
    <set begin="song1.begin" end="song1.end"
         attributeName="class" to="playing" />
    song 1
  </p>
  <p class="stopped">
    <set begin="song2.begin" end="song2.end"
         attributeName="class" to="playing" />
    song 2
  </p>
  <p class="stopped">
    <set begin="song3.begin" end="song3.end"
         attributeName="class" to="playing" />
    song 3
  </p>
</body>
```

11.3.2 Framework: Future Frameworks Under Consideration

Future Framework: Cascading Style Sheet Timing

See Appendix B: Future Framework: Cascading Style Sheet Timing for one possible method of applying timing that may be considered by the SYMM Working Group after In-Line Timing is defined.

Future Framework: Timesheets

See Appendix C: Future Framework: Timesheets for another possible method of applying timing that may be considered by the SYMM Working Group after In-Line Timing is defined.

11.4 Specification

This section will precisely define the syntax and semantics of each method of integrating SMIL timing into XML-based documents.

11.4.1 Specification: In-line Timing

This section specifies In-line timing syntax.

Time Container elements:

All time container elements defined in the SMIL Timing module may be used, along with their respective legal attributes. These elements are *predefined* time container elements.

The "timeContainer" attribute:

XML elements other than existing time container elements may be made into time container elements through the "`timeContainer`" attribute. These elements become *declared* time container elements. An XML language designer may place additional constraints on the elements that can support the "`timeContainer`" attribute. The syntax is:

```
timeContainer="t"
```

where "t" is any valid time container defined in the SMIL Timing module , including "`par`", "`seq`", and "`excl`", or "`none`":

Legal values are:

`par`
: Defines a parallel time container with the same timing and synchronization semantics as a par element.

`seq`
: Defines a sequence time container with the same timing and synchronization semantics as a seq element.

`excl`
: Defines an exclusive time container with the same timing and synchronization semantics as an excl element.

`none`
: Default value. Defines the current element to *not* have time container behavior (i.e. to behave as a simple time leaf).

Timing Attributes for Child Elements of Time Container Elements:

All child elements of both predefined and declared time container elements may contain any legal timing attributes defined for media elements as specified in the SMIL Timing module .

Any document using in-line timing markup that is not within a predefined or declared time container behaves as if the document's body is wrapped in a <par>/</par>.

The timeAction attribute:

The legal values for `timeAction` are as follows. Each host language must specify which of the following are allowed and must define the intrinsic behavior of each element to which `timeAction` may be applied. Note: a host language may not expand on the following list. These values are:

1. "intrinsic": control the intrinsic behavior of the element, as defined on an element-by-element basis. This is the default value for all elements. For SMIL media elements like video, "intrinsic" is synonymous with "display", as described below.
2. "display": make an element appear and disappear on its parent's time line.
3. "visibility": make an element appear and disappear visibly without affecting its presentation space. Note that this is the same as "display" for non-visual media.
4. "style": apply the timing to the inline style of the element.
5. "class:*classname*": add the specified classname to the value of the "class" attribute of the element when the associated timing is active.
6. "none": apply no action when active. This is generally only useful for time container elements as a means of indicating that no presentational control is applied when active and only the timing semantic is applied.

Additional timeAction rules:

- In host languages that do not support CSS, the `timeAction` values "style" and "class:*classname*" must be treated as if "intrinsic" were specified as the `timeAction` for that element.
- Media elements (i.e. the 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`.

- Time Container elements (`par`, `seq`, and `excl`) have the intrinsic behavior of the scheduling, as is true for SMIL media elements. By default, time containers should *not* have their visibility controlled as part of the `timeAction`.
- Phrasal, presentation, and style-like elements (e.g., XHTML's `b`, `em`, `strong`, ...etc.) must have an intrinsic timeAction behavior that applies their effect. When `timeAction` is set to any other value (besides intrinsic), the intrinsic presentation behavior will be controlled *in addition to* the specified `timeAction`.
- "Content" elements (e.g., XHTML's `p`, `div`, `span`) have an intrinsic behavior equivalent to "visibility", If set to any value other than "intrinsic", visibility will not be controlled in addition to the specified `timeAction`.
- Certain special elements have specific intrinsic semantics. For example, elements like `a` and `area` have an intrinsic (default) `timeAction` that controls these elements' sensitivity to actuation by the user. Based upon the linking work in SMIL, timing may actually force the actuation of links; this is still being defined. Note that in XHTML, making these elements insensitive also has the effect that the default styling that is applied to clickable links is removed when the element is not active.
- Host language designers should carefully consider and define the behavior associated with the activation and deactivation of each element. For example, `script` elements could execute when the timing is set to begin or they could simply have an "intrinsic" `timeAction` equivalent to "none". Similarly, `link` elements could apply a linked stylesheet when the timing begins or they too could have an "intrinsic" `timeAction` behavior equivalent to "none".

Examples:

Note: the In-line Timing Framework section contains several examples using SMIL timing .

11.4.2 Specification: Future Specifications Under Consideration

Future Specification: CSS Timing

See Appendix D: Future Specification: CSS Timing .

Future Specification: Timesheets

See Appendix E: Future Specification: Timesheets .

Cascading Rules

In the case where in-line timing and another method are active simultaneously, in-line timing takes precedence if a conflict arises. This enables the creation of CSS Timing or Timesheets to be used as templates whose rules can be easily modified locally by in-line constructs. The only exception to this rule is the ability for something like a user-stylesheet to be applied, such that !important rules are not overridden by inline timing. This would allow special stylesheets to control the timing in accessibility cases as well as other cases where user-specific timing may be desired. Thus, as is true for SMIL Animation as well as CSS, a user-stylesheet !important rule is always on top.

Integrating SMIL Timing into a host XML language

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

Required host language definitions

The host language designer must define some basic concepts in the context of the host language to which timing will be integrated.

The host language designer must define what "presenting a document" means. A typical example is that the document is displayed on a computer screen.

The host language designer must explicitly define the begin time of a document, i.e, does the document begin when the complete document has been received by a client (possibly over a network), does the document begin when certain document parts have been received, ...etc. This is important so that different applications that play these documents will provide the same end-user experience under the same conditions.

The host language designer must define the end time of the document. This is typically when the associated application exits or switches context to another document. The language designer may want to specify that an explicit `"end"` attribute be defined for the body element of each document, or that the body element has an indefinite duration.

The host language designer must specify which elements can be made into time containers, i.e., which elements support the "`timeContainer`" attribute, which support other timing attributes such as `"begin"` and `"dur"`, and then what the behavior of the remaining timing-free elements is under different parent element timing conditions.

The host language designer must specify when an element can be considered made active and made inactive. For example, an XHTML "b" element becoming active will only change the bold quality of text, which is something very different from the activation of a "div" element which causes a block of text to appear. How the element acts based on its activation and deactivation must be specified for each element for the host language.

Error handling semantics

The host language designer may impose stricter constraints upon the error handling semantics, but may not relax them. 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 of stricter constraints would be to stop all processing of the document, and to halt playback of the document if it had begun before the erroneous code was received by the parser. If a supported SMIL module states that certain conditions should result in an error message, the host language must display an error message under those conditions.

SMIL Timing namespace

A namespace for the "timeContainer" and "timeAction" attributes will be located at http://www.w3.org/TR/1999/smil-boston-integration.

11.5 DTD

This section provides the formal specification for the inline-specific timing markup. Refer to the SMIL Boston timing module for specification of the generic set of timing elements and attributes. Other timing markup methods to be defined will also include their DTD definitions here.

In-line Timing Syntax DTD definitions:

```
<!ENTITY % integrateInlineTimingAttrs
  timeContainer   (par | seq | excl | none) "none"
  timeAction      CDATA              #IMPLIED
>
```

11.6 Appendix A. In-Line Method Examples

1. Consider an XML-based image-list language. Each document contains a list of references to JPEG images. Timing of the images relative to one another is done in line. Here is an example of such a document, where each image in the list exists on the time line for its specified duration and is then replaced, both spatially as well as on the time line, by the next image. The final image will be active on the time line for only 8 of its 10-second duration because the parent is explicitly specified to end at that time. Note: the presentation of the elements is implied in this example.

```
<imagelist timeContainer="seq" end="28s">
  <image dur="5s" src="image1.jpg" />
  <image dur="3s" src="image2.jpg" />
  <image dur="12s" src="image3.jpg" />
  <image dur="10s" src="image4.jpg" />
</imagelist>
```

11.7 Appendix B. Future Framework: Cascading Style Sheet Timing

Reminder: the various syntaxes specified in this segment of the specification are likely to change prior to the finalization of the working draft.

Still under discussion is whether the timing attributes are XML attributes or CSS properties, i.e., whether CSS style rules will be used to apply timing properties to XML elements, or whether the timing is an actual style property. For this version of this segment of the working draft, we assume the latter but may switch to the former after further debate:

CSS Timing is the use of SMIL timing within a style sheet, where timing may be a style property, just like, for example, color and font-weight in CSS, that is applied to elements in the content document. The resultant timing structure is based on and depends on the structure of the content document. In some cases, in-line timing may be inefficient, difficult, or impossible to add particular timing. In these cases, either CSS Timing or Timesheets may be needed. Some possible cases where CSS Timing will provide a better solution than in-line timing are:

- where adding the same timing attributes to all elements of a class is needed, e.g., making all list items in the document display for 3 seconds.
- where reuse of the CSS Timing is desired for use with other content documents.
- where the content document can not be altered due to copyright or other restrictions.
- where it is desired to have two possible presentations of the same content, one a static (non-timed) presentation, and the other a timed one. This is possible when the timing is in a separate document.

The same attributes mentioned in the In-Line Timing Framework section, above, will be needed. "timeContainer" is needed to be able to declare that an element should act as a time container. The "animate" element and/or the "timeAction" attribute is needed to be able to apply timing to a style applied to the object(s).

How to ensure that CSS timing and in-line timing cooperate uniformly is still under consideration.

Here is a simple example containing one possible syntax for integrating timing using CSS. In this example, the list will play in sequence as dictated by the style sheet in the HEAD section of the document. Note: the style sheet, like any CSS, could alternatively exist as a separate document. Also, note that the timing applies, by default, to the display of the elements as opposed to the style of the elements:

```
</HEAD>
  <STYLE>
    UL { timeContainer: seq; }
    LI { font-weight: bold; dur: 5s; }
  </STYLE>
</HEAD>
<BODY>
```

```
    <UL>
      <LI>This list item will appear at 0 seconds
          and last until 5 seconds.
      </LI>
      <LI>This list item will appear after the prior
          one ends and last until 10 seconds.
      </LI>
    <UL>
  </BODY>
```

11.8 Appendix C. Future Framework: Timesheets

Timesheets refer to both the conceptual model along which timing, including the structure of the timing, is integrated into an XML document, as well as one possible syntax implementation. This approach provides a solution where time can be brought to any XML document regardless of its syntax and semantics.

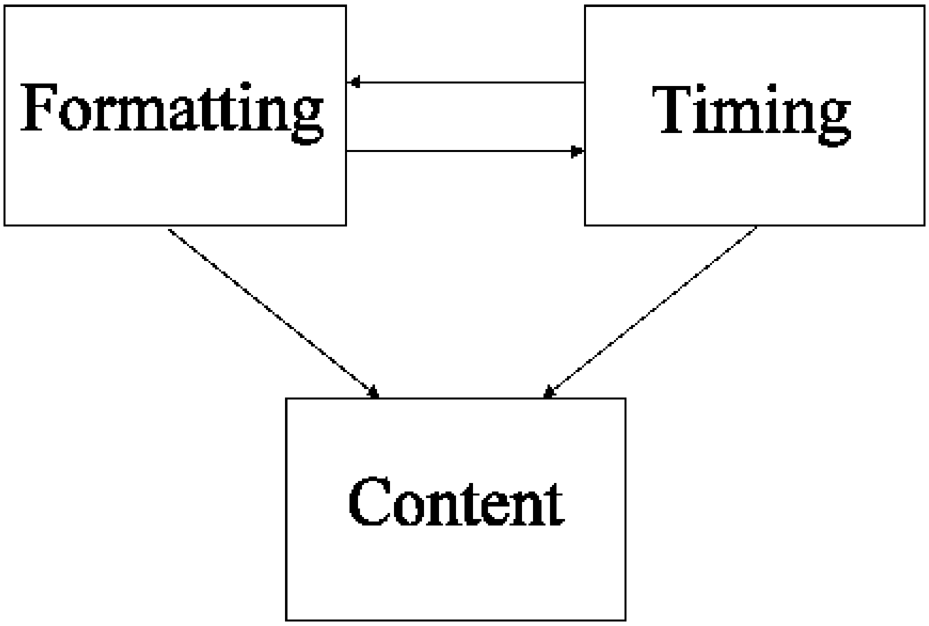
A Timesheet uses SMIL timing within a separate document or separate section of the content document and imposes that timing onto elements within the content document. The resultant timing structure is not necessarily related to the structure of the content document. Some possible cases where a Timesheet will provide a better solution than in-line timing are a superset of such CSS Timing cases (which are included in the list below):

- where the timing structure doesn't match the structure of the content, e.g., making the elements in a list appear out of order
- where adding the same timing attributes to all elements of a class is needed, e.g., making all list items in the document display for 3 seconds.
- where reuse of the Timesheet is desired for use with other content documents.
- where the content document can not be altered due to copyright or other restrictions.
- where it is desired to have two possible presentations of the same content, one a static (non-timed) presentation, and the other a timed one. This is possible when the timing is in a separate document.

11.8.1 Three document sections

Timesheets assume an XML document conceptually composed of three presentation related sections:

1. the content part.
2. the formatting part.
3. the timing part.



The first section, *content,* relates to the particular XML document. It conforms to a DTD written for an XML language. The content part describes the media and its structure.

The second section, *formatting,* provides control of the properties of the elements in the content section. It conforms to a style language, which, for the purpose of this discussion, we assume to be CSS. The style section describes the style and (spatial) layout of presenting the content. "Formatting" might include matters like routing of audio signals to loudspeakers.

The third section, *timing,* provides control of the temporal relations between the elements in the content section. It conforms to SMIL's timing model . The time section describes the time at which content is presented as well as the time at which style is applied. The time section contains the information to prepare a presentation schedule.

Sections two and three provide presentation information to the content: the stylesheet on style and positional layout, the timesheet on temporal layout. The stylesheet and timesheet may influence each other, but there should be no circular dependencies.

The idea is that each section operates independent from and compliant with the others.

11.8.2 Principles

1. The temporal structure is not necessarily implied by the content structure. Here is an example .
2. A timesheet may not be sufficient to build a time graph to provide a timing structure. A timesheet can consist of independent *rules* (time relations), which, together with the content, build the timing graph. For example, a selector in a timesheet may apply to multiple items in the content.
3. Unspecified timing may be left to the implementation to fill in. For example, items in a list can be declared to appear sequentially, while the temporal relations between lists and other content remain unspecified. When the author does not supply these, the template is still to be obeyed.
4. A timesheet may over-specify time relations. Unused rules are ignored. Conflicting time relations which concern the same element are either resolved using the timesheet cascading rules (to be specified, e.g. in-line overrides a template) or are an error (also to be made explicit). For example, when the timesheet declares sequential presentation of list items, while there are none of them in the document, the rule is simply ignored. Another example is where two rules select list items specifying different durations, e.g., all list item elements have a duration of 5 seconds except the first in each list has a duration of 8 seconds.

Here is a simple example where a timesheet exists, but in-line timing is also specified and overrides the timing imposed by the timesheet:

This example has a timesheet that specifies that each `"li"` element will have a begin time of 10 seconds and a duration of 15 seconds. However, the in-line timing in the second `"li"` element has presidence over the timesheet and thus the second line item ends up having a begin time of 0 seconds and a duration of 5 seconds. Note: this example could have been done just as easily using CSS Timing ; the added power of Timesheets will be made clearer in the next example.

```
<time>
 <par>
  li { begin=10s dur=15s }
 </par>
</time>
<body>
 <ul>
  <li>This first line will begin at 10 sec and run for 15 sec.</li>
  <li begin="0s" dur="5s">This second line’s timing is dictated
     by the in-line timing which overrides the timesheet timing
         for each child "<li>" element.  It will thus
         begin at 0 seconds and last 5 seconds.</li>
 </ul>
</body>
```

Following is an example showing some HTML extended with timing via a Timesheet. As with the CSS example , the Timesheet could just as well have been contained in a separate document and applied externally. CSS selector syntax

[CSS-selectors] has been used. The use of CSS selectors here should not be confused with CSS Timing, proposed in the prior section of this segment of the specification.
The expected presentation of this would be to have the two Headings appear together followed by the first list item in each list, namely Point A1 and Point B1, appearing at 3 seconds followed thereafter by the second list item in each list, namely Points A2 and B2, appearing at 6 seconds. All items would disappear at 10 seconds which is the duration of the outer `<par>`.

```
<html>
  <head>
    <time>
      <par dur="10">
        <par>
          h1 {}
        </par>
        <par begin="3">
          <!-- Selects the first LI in each list:         -->
          OL > LI:first-child { }
        </par>
        <par begin="6">
          <!-- Selects the second LI in each list:         -->
          OL > LI:first-child + LI { }
        </par>
      </par>
    </time>
  </head>
  <body>
    <h1>Heading A</h1>
    <ol>
      <li id="PA1">Point A1</li>
      <li id="PA2">Point A2</li>
    </ol>
    <h1>Heading B</h1>
    <ol>
      <li id="PB1">Point B1</li>
      <li id="PB2">Point B2</li>
    </ol>
  </body>
</html>
```

Note: the property fields {.} could contain duration and syncarc relations if the author wished to add more complex timing.

Here is another example as mentioned in Use Case 2C . Assume a human body display language. In this example different parts appear and disappear in different combinations at different times regardless of the content structuring, i.e., regardless of the order of the data in the document body. The document DTD uses the human structure: human = { face, torso, 2 arms, 2 legs }. A leg has a thigh, knee, calf and foot. Etc. The document merely describes the structure of the human form. Here is an example of such a document:

```
<human>
    <face id="face" ...>
       <eye id="leftEye" color="green" .../>
       <eye id="rightEye" color="blue" .../>
       ...
    </face>
    ...
    <torso>
       ...
    </torso>
    <arm id="leftArm" ...>
       ...
       <hand id="leftHand" .../>
    </arm>
    ...
    <leg id="leftLeg" ...>
       <thigh id="leftThigh" .../>
       <knee id="leftKnee" .../>
       <calf id="leftCalf" .../>
       <foot id="leftFoot" .../>
    </leg>
    ...
</human>
```

Both of the following examples are possible by applying a different timesheet in each case to the same XML document. For these examples, we use the XML "human" document, above. Note: these examples demonstrate the timesheet's ability to allow a content element to be displayed as if its parent were but with the parent not displayed, in other words the child element is displayed in the same place, spatially, as if the parent was displayed. "These examples presume that the XML language allows a content element to be displayed as if the full document was, but with some parents not displayed. In other words the child element is displayed in the same place, spatially, as if the entire document was displayed. Not all XML languages support this."

- One example is to first display hands, then add feet, then each calf and forearm, then knees and elbows, etc., building the whole human form up from the extremities. The (abbreviated) timesheet might look like this:

```
<time>
  <par dur="60s">
    <par>
          #leftHand { }
          #rightHand { }
    </par>
    <par begin="10s">
          #leftFoot { }
          #rightFoot { }
    </par>
    <par begin="20s">
          #leftCalf { }
          #rightCalf { }
          #leftForearm { }
          #righForearm { }
```

```
    </par>
    ...
  </par>
</time>
```

- A second example is to combine <seq>s and <par>s in the time section. In sequence, show a finger, the face, and a thigh. In parallel with that, accumulate the foot, calf and knee of the same leg as the thigh. The inner <par> elements are not necessary but are there to help delineate the two separate but parallel accumulations of human body parts. The (abbreviated) timesheet might look like this:

```
<time>
  <par dur="60s">
    <par>
      #rightIndexFinger { }
      #face { begin: 5s }
      #rightThigh { begin: 10s }
    </par>
    <par>
      #rightFoot { }
      #rightCalf { begin: 5s }
      #rightKnee { begin: 10s }
    </seq>
  </par>
</time>
```

11.9 Appendix D. Future Specification: CSS Timing

CSS timing syntax has not been specified, but several possibilities are under consideration.

The exact specification of CSS Timing selectors is still being considered. Selector algebra will most likely be that defined by CSS2 [CSS-selectors].

The CSS Timing Framework section contains an example using SMIL timing .

11.9.1 Timing style

In addition to selecting elements, style rules should be selectable. This enables changing style properties over time, just as we saw in the In-Line Timing color style example .

11.10 Appendix E. Future Specification: Timesheets

Timesheet syntax has not been specified, but several possibilities are under consideration. The Timesheets Framework section contains several examples (1 , 2) using SMIL timing .

11.10.1 Structure copying

The structure of the body may be used to impose temporal semantics, where a time property is assigned to an element. It is important to realize that time relations are imposed between the elements selected. For instance, when selecting a <ol> in a <seq> relation, it means that the ordered list is going to be displayed after or before some other element. It does not mean that the list items contained by the ordered list are to be presented in a sequence.

In order to provide a syntax for denoting temporal relations in line with the body structure, a new type of selectors is added to those already available from CSS.

CSS has the notion of class selectors. These selectors imply that the rule (time relation) they are part of should be applied for each element in the body that is a member of that class.

Timesheets add a new type of class selectors, henceforth to be called **structure selectors**. These selectors imply that the time relation they are part of applies to the result of expanding the structure selector into id selectors of all elements in the body that are members of that structure class. The id selectors have to appear in the order in which the elements lexically appear in the body. In this way, by selecting the class of descendants, the structure of the body section can be copied into the time section, such that the copied structure receives the temporal semantics required.

11.10.2 Structure ownership

Another form of using the structure in the XML body is called **ownership**. Ownership dictates whether a temporal relationship imposed on an element applies to all of its descendants or only on the element itself. Ownership applies for example in the sequenced <ol> case when child <li> element(s) contain further markup. By specifying that ownership is on, the children of <li> element(s) will also take on the same temporal relationship as their parents.

11.10.3 Timesheet selectors

As discussed earlier, in timesheets there are two ways to expand class selectors:

1. *Class selector*. The timesheet's rule applies per member in the class. This is the traditional CSS meaning; the timesheet's rule is repeated per element in that class.
2. *Structure selector*. The timesheet's rule applies to the set elements resulting from expanding the class selector into all its elements. For example, a structure selector is used to create a `<seq>` of `<li>` without identifying all these <li> individually.

The exact specification of timesheet selectors is still being considered. Selector algebra will most likely be that defined by CSS2 [CSS-selectors] with some additional algebra defined as necessary.

11.11 Appendix F. CSS Timing, Timesheet, and other non-In-Line Examples

1. This example uses CSS Timing to cause an otherwise static [XHTML10] list to grow over time, where each list item shows up below the prior item, 10 seconds after the prior item began its display. Because the UL becomes a "par" time container, its list items do not disappear until the UL's end time is reached.

```
/*   style sheet document "growlist.css": */
.seqtimecontainer { timeContainer:
    seq; dur: 30s} LI { dur: 10s; }

<!-- HTML document (which happens to be well-formed XML): -->
<HTML>
  <HEAD>
   <LINK rel="stylesheet" type="text/css" href="growlist.css" />>
  </HEAD>
  <BODY>
    <UL class="seqtimecontainer">
          <LI>This is item 1.  It appears from 0 to 30 seconds.
          </LI>
          <LI>This is item 2.  It appears from 10 to 30 seconds.
          </LI>
          <LI>This is item 3.  It appears from 20 to 30 seconds.
          </LI>
        </UL>
  </BODY>
</HTML>
```

2. Consider a document written in some graphics language where three big squares are layed out inside a rectangle, and each square contains a smaller square. We should be able to create a timesheet that can schedule the appearance of each square at different times from the others. Note: the presentation of the elements is implied in this example.

```
<rectangle id="window" geometry="..." fill="...">
  <square id="b1" ... >
    <square id="s1" ... / >
  </square>
  <square id="b2" ... >
    <square id="s2" ... / >
  </square>
  <square id="b3" ... >
    <square id="s3" ... / >
  </square>
</rectangle>
```

In order to time the presentation of the elements so that the big squares pop up one after the other, followed by the simultaneous appearance of the small ones, the timesheet might look like this:

```
<time>
  <seq>
    <par>
      #b1 { dur: 2s }
      #b2 { dur: 2s; begin: 2s; }
      #b3 { dur: 2s; begin: 4s; }
    </par>
    <par>
      #s1 { }
      #s2 { }
      #s3 { }
    </par>
  </seq>
</time>
```

Note: the outer "window" rectangle has not been given any explicit timing. for this example, we assume that the lack of timing implies a begin time of zero and an indefinite duration if the element does not have an implicit duration.

12. The SMIL Transition Effects Module

Editors
Eric Hyche (ehyche@real.com), (RealNetworks)

12.1 Introduction

In most public descriptions of SMIL, the language is described as "allowing authors to bring TV-like content to the Web." However, one aspect of presentations commonly seen on television has been noticeably absent from SMIL: transitions such as fades and wipes. In SMIL 1.0, any representation of transitions had to be "baked into" the media itself and there was no method of coordinating transitions across multiple media regions and timelines. The purpose of this document is to specify the semantics and syntax for describing transitions within SMIL and other XML-based documents. Also, this specification describes a taxonomy of transitions based on SMPTE 258M-1993 [SMPTE] as well as a compact set of parameters which can be used to express this set of transitions. Although the majority of transitions described in this document are *visual* transitions, a number of transitions have *audio* equivalents and are equally applicable.

Any XML language that wants to make use of transitions must have:

1. *A Layout Language*. Transitions operate on media elements which are associated with layout elements. If transitions are coordinated across multiple media elements, then it is necessary to be able to access properties of the layout region in which that media is playing. Therefore, any language which would use transitions must have the ability to express the concept of playback regions. CSS2 and SMIL 1.0 are examples of languages with layout language capabilities.
2. *A Timing Model*. In this context, transitions are time-based, client-side effects between media. Since transitions occur over time and are applied to media at a certain point in time, then the host XML language must have ability to specify a timeline. SMIL 1.0 and HTML+TIME are examples of languages with a time model.

For example, consider a simple still image slideshow of four images, each displayed for 5 seconds. In SMIL 1.0 this might look like:

```
<smil>
    <head>
        <layout>
            <root-layout width="256" height="256" background-color="#000000"/>
            <region id="whole" left="32" top="32" width="192" height="192"/>
        </layout>
    </head>
    <body>
        <seq>
            <img src="butterfly.jpg" region="whole" dur="5s"/>
```

```
            <img src="eagle.jpg"     region="whole" dur="5s"/>
            <img src="wolf.jpg"      region="whole" dur="5s"/>
            <img src="seal.jpg"      region="whole" dur="5s"/>
        </seq>
    </body>
</smil>
```

and the corresponding presentation in HTML+TIME (for the timing model) and CSS2 (for the layout language):

```
<HTML>
    <HEAD>
        <XML:NAMESPACE PREFIX="t"/>
        <STYLE>
            DIV      { position:         absolute;
                       left:             0px;
                       top:              0px;
                       width:            256px;
                       height:           256px;
                       background-color: #000000 }
            .whole   { position:         absolute;
                       left:             32px;
                       top:              32px;
                       width:            192px;
                       height:           192px }
        </STYLE>

    </HEAD>
    <BODY>
        <DIV STYLE="behavior:url(#default#time);" t:TIMELINE="seq">
            <t:IMG CLASS="whole" STYLE="behavior:url(#default#time);" t:SRC="butterfly.jpg"   t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole" STYLE="behavior:url(#default#time);" t:SRC="eagle.jpg"       t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole" STYLE="behavior:url(#default#time);" t:SRC="wolf.jpg"        t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole" STYLE="behavior:url(#default#time);" t:SRC="seal.jpg"        t:DUR="5" t:TIMEACTION="display"/>
        </DIV>
    </BODY>
</HTML>
```

Currently when these presentations play, we see a straight "cut" from one image to another, as shown in this animated image. However, what we would like to see are three wipes in between the four images: in between butterfly.jpg and eagle.jpg at 5 seconds, in between eagle.jpg and wolf.jpg at 10 seconds, and in between wolf.jpg and seal.jpg at 15 seconds. Therefore, we must define the following to our presentations:

1. The *class* of transition we wish to apply. For instance, if we want all three transitions to be 1-second wipes, then we can define 1-second wipes as a transition class and apply this class to the media elements. For purposes of introductory illustration, we will use several transition parameters without first defining them. However, for complete detail about transition parameters, see the Transition Taxonomy and Parameters section.
2. The *media elements* to which we wish to apply this transition class. For more detail about applying transition classes to media elements, see the Applying Transitions to Media Elements section.

Adding these two definitions to the previous SMIL 1.0 slideshow example would make the presentation now look like:

```
<smil>
    <head>
        <layout>
            <root-layout width="256" height="256" background-color="#000000"/>
            <region id="whole" left="32" top="32" width="192" height="192"/>
        </layout>
        <transition id="wipe1" type="wipe" subtype="slideHorizontal" dur="1s"/>
```

```
    </head>
    <body>
        <seq>
            <img src="butterfly.jpg" region="whole" dur="5s"/>
            <img src="eagle.jpg"     region="whole" dur="5s" transition="wipe1"/>
            <img src="wolf.jpg"      region="whole" dur="5s" transition="wipe1"/>
            <img src="seal.jpg"      region="whole" dur="5s" transition="wipe1"/>
        </seq>
    </body>
</smil>
```

and the presentation in HTML+TIME and CSS2 would now look like:

```
<HTML>
    <HEAD>
        <XML:NAMESPACE PREFIX="t"/>
        <STYLE>
            DIV      { position:         absolute;
                       left:             0px;
                       top:              0px;
                       width:            256px;
                       height:           256px;
                       background-color: #000000 }
            .whole   { position:         absolute;
                       left:             32px;
                       top:              32px;
                       width:            192px;
                       height:           192px }
            .wipe1   { transitionType:    wipe;
                       transitionSubType: slideHorizontal;
                       transitionDur:     1s }
        </STYLE>
    </HEAD>
    <BODY>
        <DIV STYLE="behavior:url(#default#time);" t:TIMELINE="seq">
            <t:IMG CLASS="whole"       STYLE="behavior:url(#default#time);" t:SRC="butterfly.jpg"   t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole;wipe1" STYLE="behavior:url(#default#time);" t:SRC="eagle.jpg"       t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole;wipe1" STYLE="behavior:url(#default#time);" t:SRC="wolf.jpg"        t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole;wipe1" STYLE="behavior:url(#default#time);" t:SRC="seal.jpg"        t:DUR="5" t:TIMEACTION="display"/>
        </DIV>
    </BODY>
</HTML>
```

Now the presentations play as follows. First, at 0 seconds, we cut directly to butterfly.jpg. Next, at 5 seconds we begin a 1-second wipe into eagle.jpg. Therefore, at 6 seconds, eagle.jpg is fully displayed and remains displayed for 4 more seconds until 10 seconds. At this time, we begin a another 1-second wipe from eagle.jpg to wolf.jpg. At 11 seconds, wolf.jpg is fully displayed until 15 seconds, when we begin another 1-second transition to seal.jpg. At 16 seconds, seal.jpg is fully displayed until 20 seconds at which time the presentation ends. When the presentation ends, there is an immediate cut to black due to the default fill="remove" behavior of SMIL and the TIMEACTION="display" behavior of HTML+TIME. This is visually illustrated by this animated image. Notice that these transitions occur *during* the timeline each of the images and do not add or subtract from their host timeline. In this case, the transition occurs (by default) at the beginning of the timeline, although we will discuss later a method of placing the transition at the end of a media element's timeline.

This document is structured as follows. In the Taxonomy section, we define a taxonomy of transitions and describe the families of transitions. Next in the Parameters section, we define a set of parameters which can fully describe all the transitions in our taxonomy. Next, in the Applying Transitions to Media Elements section, we describe the semantics of applying a transition class to a media element. Next, in the Multiple-Element Transitions section, we describe how to apply single transitions across multiple media elements.

12.2 Transition Taxonomy

Using CSS, making text appear in a certain font face and size involves defining a style and then using selectors to apply that style to the appropriate elements. The entire set of possible font faces are grouped into broad font families with specialization within each family. In a similar manner, we define in this section several broad families of transitions and describe the distinguishing characteristics of each family. In the next section, we will define a parameter set which can fully specify all the transitions in each family.

In all of the examples of specific transitions mentioned in this document, we will refer to the following model: we refer to the element being transitioned *from* as element A (or just A) and we refer to the element being transitioned *to* as element B (or just B). We define the following eight families (or *types*) of transitions:

edgeWipe
: B is "under" A and is uncovered by combinations of edges. SMPTE Wipe Codes 1-74 are members of this family. For example, in SMPTE Wipe Code 1, a vertical line moves left to right across A. B is revealed on the left side of the line, while A remains on the right side. SMPTE Wipe Code 1 is illustrated by this animated image.

irisWipe
: B is "under" A and is uncovered by an expanding shape. SMPTE Wipe Codes 101-131 are members of this family. For example, in SMPTE Wipe Code 102, B is gradually revealed by an expanding diamond shape. SMPTE Wipe Code 102 is illustrated by this animated image.

radialWipe
: B is "under" A and is uncovered by one or more radial sweeps. SMPTE Wipe Codes 201-264 are members of this family. For example, in SMPTE Wipe Code 201, B is revealed by a clockwise sweep, as shown in this animated image.

matrixWipe
: B is "under" A and is uncovered by one or more block traversals. SMPTE Wipe Codes 301-353 are members of this family. For example, in SMPTE Wipe Code 301, B is revealed by a block which alternates moving left to right then right to left as it moves down A. This transition is illustrated by this animated image.

pushWipe
: A is "pushed" out of view by B. An example of this family would be a transition where B moves in from the left, while pushing A out of view. This transition is illustrated by this animated image.

slideWipe
: B "slides over" A. An example of this family would be a transition where B moves in from the left, and slides *over* A, as illustrated by this animated image.

fade
: additive blend between A and B, A and a color, or B and a color. An example of this family would be a crossfade between A and B, as illustrated by this animated image.

warp
: A or B is spatially distorted until only B remains. An example of this family would be when B zooms in on top of A, as illustrated by this animated image.

Each of these transition "types" are further divided into many "subtypes". The table below lists the possible subtypes for each type. Also the table lists the mapping between the assigned name and the SMPTE Wipe Code (where applicable).

Transition type	**Transition subtypes** (SMPTE Wipe Codes in parentheses)
edgeWipe	"slideHorizontal" (1) [default], "slideVertical" (2), "topLeft" (3), "topRight" (4), "bottomRight" (5), "bottomLeft" (6), "fourCorner" (7), "fourBox" (8), "barnVertical" (21), "barnHorizontal" (22), "topCenter" (23), "rightCenter" (24), "bottomCenter" (25), "leftCenter" (26), "diagonalLeftDown" (41), "diagonalRightDown" (42), "verticalBowTie" (43), "horizontalBowTie" (44), "diagonalLeftOut" (45), "diagonalRightOut" (46), "diagonalCross" (47), "diagonalBox" (48), "filledVUp" (61), "filledVRight" (62), "filledVBottom" (63), "filledVLeft" (64), "hollowVUp" (65), "hollowVRight" (66), "hollowVBottom" (67), "hollowVLeft" (68), "verticalZigZag" (71), "horizontalZigZag" (72), "verticalBarnZigZag" (73), "horizontalBarnZigZag" (74)
irisWipe	"rectangle" (101) [default], "diamond" (102), "triangleUp" (103), "triangleRight" (104), "triangleDown" (105), "triangleLeft" (106), "arrowheadUp" (107), "arrowheadRight" (108), "arrowheadDown" (109), "arrowheadLeft" (110), "pentagonUp" (111), "pentagonDown" (112), "hexagon" (113), "hexagonSide" (114), "cicle" (119), "oval" (120), "ovalSide" (121), "catEye" (122), "catEyeSide" (123), "roundRect" (124), "roundRectSide" (125), "star4pt" (127), "star5pt" (128), "star6pt" (129), "heart" (130), "keyhole" (131)
radialWipe	"top" (201) [default], "right" (202), "bottom" (203), "left" (204), "topBottom" (205), "leftRight" (206), "quadrant" (207), "topBottom180" (211), "rightLeft180" (212), "topBottom90" (213), "rightLeft90" (214), "top180" (221), "right180" (222), "bottom180" (223), "left180" (224), "counterTopBottom" (225), "counterLeftRight" (226), "doubleTopBottom" (227), "doubleLeftRight" (228), "vOpenTop" (231), "vOpenRight" (232), "vOpenBottom" (233), "vOpenLeft" (234), "vOpenTopBottom" (235), "vOpenLeftRight" (236), "topLeft" (241), "bottomLeft" (242), "bottomRight" (243), "topRight" (244), "topLeftBottomRight" (245), "bottomLeftTopRight" (246), "topLeftRight" (251), "leftTopBottom" (252), "bottomLeftRight" (253), "rightTopBottom" (254), "doubleCenterRight" (261), "doubleCenterTop" (262), "doubleCenterTopBottom" (263), "doubleCenterLeftRight" (264)

matrixWipe	"horizontal" (301) [default], "vertical" (302), "topLeftDiagonal" (303), "topRightDiagonal" (304), "bottomRightDiagonal" (305), "bottomLeftDiagonal" (306), "cwTopLeft" (310), "cwTopRight" (311), "cwBottomRight" (312), "cwBottomLeft" (313), "ccwTopLeft" (314), "ccwTopRight" (315), "ccwBottomRight" (316), "ccwBottomLeft" (317), "verticalStartTop" (320), "verticalStartBottom" (321), "verticalStartTopOpposite" (322), "verticalStartBottomOpposite" (323), "verticalStartLeft" (324), "verticalStartRight" (325), "verticalStartLeftOpposite" (326), "verticalStartRightOpposite" (327), "doubleDiagonalTopRight" (328), "doubleDiagonalBottomRight" (329), "doubleSpiralTop" (340), "doubleSpiralBottom" (341), "doubleSpiralLeft" (342), "doubleSpiralRight" (343), "quadSpiralVertical" (344), "quadSpiralHorizontal" (345), "verticalWaterfallLeft" (350), "verticalWaterfallRight" (351), "horizontalWaterfallLeft" (352), "horizontalWaterfallRight" (353)
pushWipe	"fromTop", "fromRight", "fromBottom", "fromLeft" [default]
slideWipe	"fromTop", "fromRight", "fromBottom", "fromLeft" [default], "angular"
fade	"crossfade" [default], "fadeToColor", "fadeFromColor"
warp	"explode", "implode", "zoomOver" [default], "zoomBoth"

For each of the types, the first subtype is labeled as the "default" subtype. The purpose of this is to allow for a default transition for this transition family, if either the transition subtype is not specified or not implemented. This is a similar idea to CSS's font-family property, where the value is a comma-separated list of font faces of families. If the first font in the list is not available, then the browser tries the second. Usually, the last font in the list will be very generic, so that all browsers can support it.

In the same way, authors can specify a type and subtype for a transition class. If this transition class is not available or not implemented by the user agent, then the user agent should fall back on the default subtype for that transition family. The side effect of this is that all renderers are required to support a minimum of 8 transitions (the default transition for each of the transition families).

12.3 Transition Parameters

Now that we have a taxonomy of transition types and subtypes defined, now we must define a set of parameters which can span the entire space of transitions. In the following list, not all the parameters apply to every transition type. However, there is enough commonality between parameters for each family that it is not useful to have a separate parameter set for each transition family.

We also present the `<transition>` element for SMIL. In SMIL, this element defines a single transition class. If the transition class is expressed in a stylesheet language such as CSS, then each of these parameters are properties defined in CSS syntax within the <STYLE> element. In order not to be distinguished from other CSS properties, the prefix "transition" should be prepended to each of the parameter names to create the CSS property name, using camelCase to mark the separation between words. For example, the transition parameter "dur" would translate directly to "dur" as a SMIL attribute but would translate to "transitionDur" as a CSS property. We will reference an Integration section here when that section is complete.

12.3.1 The `<transition>` element

The `<transition>` element defines a single transition class within a SMIL document. This element should appear in the `<head>` section of the document. Since there may be multiple transition classes used in a SMIL document, then there may be multiple `<transition>` elements in the `<head>` section of the SMIL document.

Element attributes

id
: This attribute uniquely identifies the transition element within the SMIL document. Its value is an XML identifier.

type
: This is the type or family of transition. The "type" attribute is required and must be one of the transition families listed in the Taxonomy section.

subtype
: This is the subtype of the transition. This parameter is optional and if specified, must be one of the transition subtypes appropriate for the specified type as listed in the table of subtypes in the Taxonomy section. If this parameter is not specified, it defaults to the subtype listed as the default subtype for the specified transition type.

dur
: This is the duration of the transition. The value of this attribute must be a clock-value as defined by the SMIL Timing and Synchronization Module . This parameter has a default of 1 second.

base
: This defines whether the transition is applied to beginning or the end of the host timeline. There are two possible values for base:
  begin
  : The transition occurs during the time [0,dur] in the host timeline. This is the default value.
  end
  : The transition occurs during the time [D-dur,D] in the host timeline, where D is the duration of the host timeline, which may be possibly unknown at authoring time.

startPercent

This is the percentage through the transition at which to begin execution. Legal values are integers in the range [0,100]. For instance, we may want to begin a crossfade with the destination image being already 30% faded in.
The default value is zero.

endPercent

This is the percentage through the transition at which to end execution. Legal values are integers in the range [0,100] and the value of this attribute must be greater than or equal to the value of the "startPercent" attribute.
The default value is 100.

horzRepeat

This attribute specifies how many times to repeat the wipe pattern along the horizontal axis.
The default value is 0 (the pattern is not repeated horizontally).

vertRepeat

This attribute specifies how many times to repeat the wipe pattern along the vertical axis.
The default value is 0 (the pattern is not repeated vertically).

startX

This attribute specifies the distance from the left side of the element region at which to start the warp transition. This parameter is a floating point number in the range [-2.0, 2.0], where 0.0 is the center of the region, -1.0 is the left edge, and 1.0 is the right edge. Therefore, half of the width of the region is defined as a unit measure. So -2.0 is two units to the left of the center of the element region, and 2.0 is two units to the right of the center of the element region.
The default value is 0.

startY

This attribute specifies the distance from the top side of the element at which to start the warp transition. This parameter is a floating point number in the range [-2.0, 2.0], where 0.0 is the center of the region, -1.0 is the left edge, and 1.0 is the right edge. Therefore, half of the height of the region is defined as a unit measure. So -2.0 is two units above the center of the element region, and 2.0 is two units below the center of the element region.
The default value is 0.

endX

This attribute specifies the distance from the left side of the element at which to end the warp transition. This parameter is a floating point number in the range [-2.0, 2.0], where 0.0 is the center of the region, -1.0 is the left edge, and 1.0 is the right edge. Therefore, half of the width of the region is defined as a unit measure. So -2.0 is two units to the left of the center of the element region, and 2.0 is two units to the right of the center of the element region.
The default value is 0.

endY

This attribute specifies the distance from the top side of the element at which to start the warp transition. This parameter is a floating point number in the range [-2.0, 2.0], where 0.0 is the center of the element, -1.0 is the left edge, and 1.0 is

the right edge. Therefore, half of the height of the region is defined as a unit measure. So -2.0 is two units above the center of the element region, and 2.0 is two units below the center of the element region.
The default value is 0.

borderWidth
: This attribute specifies the width of a generated border along a wipe edge. Legal values are integers greater than or equal to 0. If borderWidth is equal to 0, then this implies no generated border along the wipe edge.
The default value is 0.

color
: If the value of the "type" attribute is "wipe", "iris", "radial", "matrix", "push", "slide", or "warp", then this attribute specifies the content of the generated border along a wipe or warp edge. This attribute can either be a color (as specified by the "background-color" property of the CSS2 specification) *or* the string "blend". If the value of this attribute is a color, then the generated border along the wipe or warp edge is filled with this color. If the value of this attribute is "blend", then generated border along the wipe blend is an additive blend (or blur). If the value of the "type" attribute is "fade" and the value of the "subtype" attribute is "fadeToColor" or "fadeFromColor", then this color specifies the starting or ending color of the fade.
The default value is "black".

Element content

The `<transition>` element is an empty element.

Examples of the `<transition>` element.

For example, suppose we wanted to define two transition classes: a simple 2-second fade-to-black and a 5-second keyhole-shaped iris wipe. In SMIL, our definition would look like:

```
...
<head>
    ...
    <transition id="ftb2" type="fade"     subtype="fadeToColor"
                   dur="2s" color="#000000" />
    <transition id="key5" type="irisWipe" subtype="keyhole"
                   dur="5s" />
    ...
</head>
...
```

and in a CSS-like syntax our definition would look like:

```
...
<HEAD>
    ...
    <STYLE>
        .ftb2 { transitionType:    fade;
                transitionSubtype: fadeToColor;
                transitionDur:     2s;
```

```
                  transitionColor:   #000000 }
          .key5 { transitionType:    irisWipe;
                  transitionSubtype: keyhole;
                  transitionDur:     5s }
      </STYLE>
      ...
  </HEAD>
  ...
```

Note that in SMIL, the "id" attribute is necessary to identify the transition class. In CSS, the transition class name is implicit in the CSS class selector notation and thus an "id" property is unnecessary.

12.3.2 Handling Parameter Errors

Transitions parameters can be specified incorrectly in many different ways with varying levels of severity. Therefore, the following errors should be handled with the specified action:

1. *Transitions subtype is not valid for specified transition type.* The specified transition subtype should be ignored and the default subtype for the specified transition type should be performed.
2. *Transition duration is not specified.* The default duration of 1 second should be assumed.
3. *Transition parameter is outside the legal range.* If a transition parameter is specified outside of the legal range, then the default value of the parameter should be assumed. For instance, the startX parameter has a legal range of [-2.0,2.0] and a default value of 0.0. If startX were to be specified as 3.0, then the specified value should be ignored and a default value of 0.0 should be assumed.
4. *Transition parameter does not apply to this transition type.* Since not all transition parameters apply to all transition types, then a common error could be to specify a transition parameter which does not apply to the specified transition type. These irrelevant parameters should be ignored. For instance, the "startX" parameter only applies to the "warp" transition type. If "startX" were to be specified for the "edgeWipe" transition type, then it should be ignored.
5. *Transition duration is longer than the duration of the media object itself.* In this case, the entire transition should be ignored and not performed.
6. *Two transitions are specified as active on the same region at the same time.* For example, image B follows image A in a region and A has an base="end" transition and B has a base="begin" transition. In conflicts involving a begin versus an end transition, the begin transition is used and the end transition is ignored.

12.4 Applying Transitions to Media Elements

Once a transition class has been defined in the head of a document, then a transition instance can be created by applying the transition class to the timeline of a media object element. For languages which support CSS style, the class selector is used to apply a transition class to a media element. The value of the class attribute is defined in the class selector of the transition definition. For SMIL, a "transition" attribute is added to all media object elements.

12.4.1 The "transition" attribute.

The "transition" attribute is added to all media object elements. The default value of the transition attribute is an empty string, which indicates that no transition should be performed. The value of the "transition" attribute should be the same as the value of the "id" attribute of one of the `<transition>` elements defined in the <head> of the document. If the value of the "transition" attribute does not correspond to the value of the "id" attribute of any one of the `<transition>` elements in the <head> of the document, then this is an error. In this case, the value of the "transition" attribute should be considered to be the empty string and therefore no transition should be performed.

In SMIL, this attribute may be applied to any media object element. These elements are listed in the SMIL Media Object Module . In other languages, this attribute may be applied to the appropriate elements which reference media objects. Also this element may be applied to other non-media elements for which transitions are desired (such as the `<DIV>` element in HTML.

Examples of applying the "transition" attribute.

Consider the slideshow example in the Introduction of the document with two additions: a fade-from-black is applied to butterfly.jpg and a fade-to-black is applied to seal.jpg. In SMIL this would look like:

```
<smil>
    <head>
        <layout>
            <root-layout width="256" height="256" background-color="#000000"/>
            <region id="whole" left="32" top="32" width="192" height="192"/>
        </layout>
        <transition id="xfade1s"    type="fade" subtype="crossfade"     dur="1s"/>
        <transition id="fromblack1" type="fade" subtype="fadeFromColor" dur="1s"/>
        <transition id="toblack1"   type="fade" subtype="fadeToColor"   dur="1s" base="end" />
    </head>
    <body>
       <seq>
            <img src="butterfly.jpg" region="whole" dur="5s" transition="fromblack1"/>
            <img src="eagle.jpg"     region="whole" dur="5s" transition="xfade1s"/>
            <img src="wolf.jpg"      region="whole" dur="5s" transition="xfade1s"/>
            <img src="seal.jpg"      region="whole" dur="5s" transition="xfade1s,toblack1"/>
       </seq>
    </body>
</smil>
```

and

```
<HTML>
    <HEAD>
        <XML:NAMESPACE PREFIX="t"/>
        <STYLE>
            DIV      { position:         absolute;
                       left:             0px;
                       top:              0px;
                       width:            256px;
                       height:           256px;
                       background-color: #000000 }
            .whole   { position:         absolute;
                       left:             32px;
                       top:              32px;
                       width:            192px;
                       height:           192px }
            .xfade1s { transitionType:    fade;
                       transitionSubType: crossfade;
                       transitionDur:     1s }
            .fromblack1 { transitionType:    fade;
                          transitionSubType: fadeFromColor;
                          transitionDur:     1s;
                          transitionColor:   #000000 }
            .toblack1 { transitionType:    fade;
                        transitionSubType: fadeToColor;
                        transitionDur:     1s;
                        transitionColor:   #000000;
                        transitionBase:    end }
        </STYLE>
    </HEAD>
    <BODY>
        <DIV STYLE="behavior:url(#default#time);" t:TIMELINE="seq">
            <t:IMG CLASS="whole,fromblack1"       STYLE="behavior:url(#default#time);" t:SRC="butterfly.jpg"   t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole,xfade1s"          STYLE="behavior:url(#default#time);" t:SRC="eagle.jpg"       t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole,xfade1s"          STYLE="behavior:url(#default#time);" t:SRC="wolf.jpg"        t:DUR="5" t:TIMEACTION="display"/>
            <t:IMG CLASS="whole,xfade1s,toblack1" STYLE="behavior:url(#default#time);" t:SRC="seal.jpg"        t:DUR="5" t:TIMEACTION="display"/>
        </DIV>
    </BODY>
</HTML>
```

We will use this example to illustrate the following rules for applying transitions to media elements:

1. Since the purpose of transitions is "transitioning" from one media object to another, then transitions must be applied to either the beginning or end (or both) of some media object. However, the visual effect may appear to be applying this transition in the middle of an element's timeline. Consider the following SMIL snippet:

   ```
   ...
   <par>
       <img src="butterfly.jpg" dur="10s" region="foo"/>
       <img src="eagle.jpg" begin="3s" dur="4s" region="bar" />
   </par>
   ...
   ```

   Assuming that region "bar" is z-ordered on top of region "foo", then transitions applied to both the beginning and end of eagle.jpg would have the visual appearance of being applied during the timeline of butterfly.jpg. However, from the authoring perspective, they are still applied at the beginning and end of eagle.jpg.
2. Applying a transition to the beginning or end of an element's timeline does not affect the duration of the element. For instance, in the example above, applying a 1-second transition at the beginning of eagle.jpg does not add or subtract from the timeline of eagle.jpg - it is still displayed from 5-10 seconds in the presentation. Applying a 1-second transition at the beginning of eagle.jpg makes the transition take place from [5,6] seconds and applying a 2-second transition at the end of eagle.jpg would make the transition happen from [8,10] seconds.
3. Transitions which occur at the end of a media object's timeline must respect the object's fill behavior. In other words, a transition intended for the end of a media

object's timeline actually take place at the *effective* end of that element's timeline. For instance, in the following presentation:

```
...
<transition id="toblack1s" type="fade" subType="fadeToColor"
               color="#000000" base="end" dur="1s"/>
...
<par>
    <img   ... dur="10s" transition="toblack1s" fill="freeze"/>
    <video ... dur="30s" transition="toblack1s"/>
</par>
```

the effective end of the <img> element is 30s. Therefore both elements fade to black together at 29s. However, in the following:

```
...

<transition id="toblack1s"   type="fade" subType="fadeToColor"
               color="#000000" base="end" dur="1s"/>
...
<par>
    <img   ... dur="10s" transition="toblack1s" fill="remove"/>
    <video ... dur="30s" transition="toblack1s"/>
</par>
```

the effective end of the <img> element is 10s. Therefore, in this case the <img> element fades to black starting at 9s and the <video> element fades to black starting at 29s.

4. The timeline for the media element we are transitioning *to* must either overlap or immediately follow the timeline for the media element we are transitioning *from*. In the slideshow example, the timelines for each media object we are transitioning to immediately follow the end of the timeline of the objects we are transitioning from. In these cases (where the timelines immediately follow but do not overlap), the transition is effectively between the frozen last frame of the previous ("from") media and active frames of the current ("to") media. In cases where the timelines overlap (and hence the regions being played to have different z-orders), the transition is between active frames of both media. For instance, in this transition:

```
...
<seq>
    <video src="foo1.mpg" region=<reg1> ... />
    <video src="foo2.mpg" region=<reg1> transition="xfade1s" ... />
</seq>
...
```

the timelines do not overlap and therefore we are doing a crossfade between the last frame of foo1.mpg and active frames of foo2.mpg. In the following presentation, however:

```
...
<transition id="xfadebeg" type="fade" subtype="crossfade" dur="1s" />
<transition id="xfadeend" type="fade" subtype="crossfade" dur="1s" base="end" />
...
<par>
    <video src="foo1.mpg" dur="30s" region="reg1" />
    <video src="foo2.mpg" begin="10s" dur="10s" region="reg2" transition="xfadebeg,xfadeend" />
</par>
...
```

crossfades both at the beginning and end of foo2.mpg are between active frames of both foo1.mpg and foo2.mpg.

5. If the timelines for the media objects involved in the transition do not overlap, then the background color for the missing regions should be used. For example,

```
...
<transition id="awipe" type="wipe" dur="1s" ... />
...
<par>
    <img src="img1.jpg" region="whole" dur="2s" transition="awipe" base="end"/>
    <img src="img2.jpg" region="whole" begin="5s" dur="2s"/>
</par>
...
```

In this example, the timelines for img1.jpg and img2.jpg do not overlap. Therefore, img1.jpg will transition to the background color of the region.

6. The fill behavior of an element instructs the user agent when it can remove the media object. For instance, in the following example *not* using transitions,

```
...
<seq>
    <img src="img1.jpg" region="whole" dur="10s" />
    <img src="img2.jpg" region="whole" dur="10s" />
</par>
...
```

the implementation knows that it can remove the object representing img1.jpg after 10 seconds. However, if we were using a transition between img1.jpg and img2.jpg, then we need the object for img1.jpg to remain until after the transition is completed and then it may be removed. This is a new kind of fill behavior and is specified by a new value for the fill attribute called "transition". In the above example,

```
...
<seq>
    <img src="img1.jpg" region="whole" dur="10s" fill="transition" transition="awipe"/>
    <img src="img2.jpg" region="whole" dur="10s" />
</par>
...
```

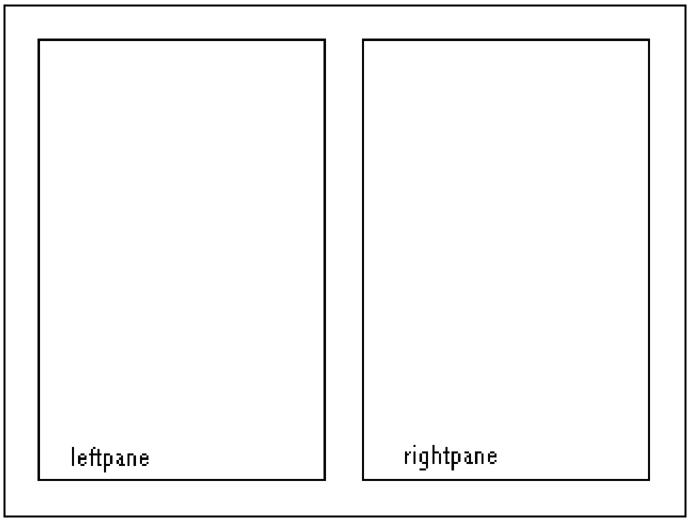
the implementation is instructed to keep the object for img1.jpg around long enough to complete the transition between img1.jpg and img2.jpg in the region named "whole".

12.5 Multi-Element Transitions

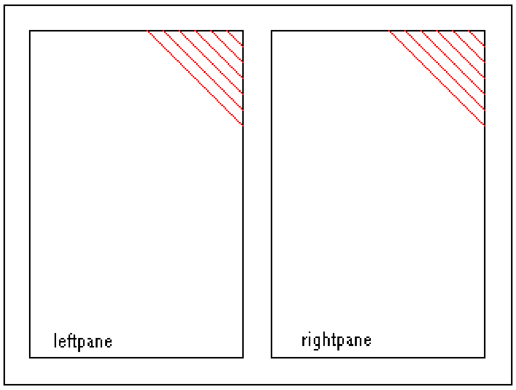
Up until this point in the discussion, we have applied transitions to single media object elements. However, it is common practice to apply transitions across several different media at once. Consider the following example:

```
<smil>
    <head>
        <layout>
            <root-layout width="320" height="240" background-color="#000000"/>
            <region id="whole"     left="0"   top="0"  width="320" height="240" z-index="0"/>
            <region id="leftpane"  left="16"  top="16" width="136" height="208" z-index="1"/>
            <region id="rightpane" left="168" top="16" width="136" height="208" z-index="1"/>
        </layout>
    </head>
    <body>
        <seq>
            <par>
                <img src="back1.jpg"  dur="10s" region="whole"/>
                <img src="left1.jpg"  dur="10s" region="leftpane"/>
                <img src="right1.jpg" dur="10s" region="rightpane"/>
            </par>
            <par>
                <img src="back2.jpg"  dur="10s" region="whole"/>
                <img src="left2.jpg"  dur="10s" region="leftpane"/>
                <img src="right2.jpg" dur="10s" region="rightpane"/>
            </par>
        </seq>
    </body>
</smil>
```

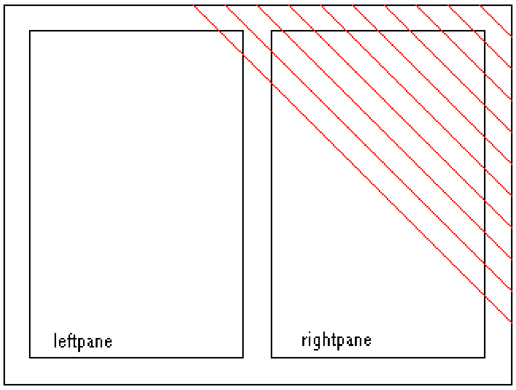
where the regions of this presentation look like:



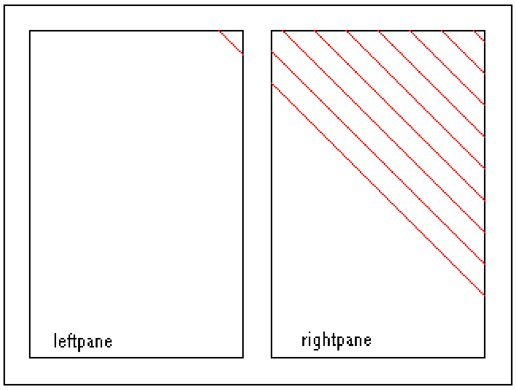
Suppose that we had defined a transition class called "diagwipe" to be a 1-second diagonal wipe from upper right to lower left. In this example, we consider 4 possible different cases of how we might want to apply this transition to this presentation:



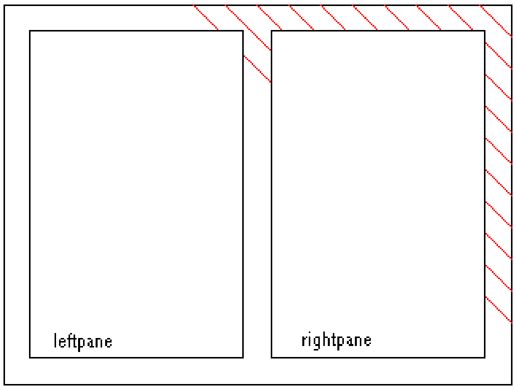
Case 1



Case 2



Case 3



Case 4

Cases 1 and 4 are fairly straightforward, since they are applying individual transitions to individual media elements, which we discussed in the previous section. The SMIL for Case 1 would look like:

```
<smil>
    <head>
        <layout>
            <root-layout width="320" height="240" background-color="#000000"/>
            <region id="whole"     left="0"   top="0"  width="320" height="240" z-index="0"/>
            <region id="leftpane"  left="16"  top="16" width="136" height="208" z-index="1"/>
            <region id="rightpane" left="168" top="16" width="136" height="208" z-index="1"/>
        </layout>
        <transition id="diagwipe" type="wipe" subtype="diagonalRightDown" dur="1s"/>
    </head>
    <body>
        <seq>
            <par>
                <img src="back1.jpg"  dur="10s" region="whole"/>
                <img src="left1.jpg"  dur="10s" region="leftpane"/>
                <img src="right1.jpg" dur="10s" region="rightpane"/>
            </par>
            <par>
                <img src="back2.jpg"  dur="10s" region="whole"/>
                <img src="left2.jpg"  dur="10s" region="leftpane"  transition="diagwipe"/>
                <img src="right2.jpg" dur="10s" region="rightpane" transition="diagwipe"/>
```

```
            </par>
        </seq>
    </body>
</smil>
```

and the SMIL for Case 4 would look like:

```
<smil>
    <head>
        <layout>
            <root-layout width="320" height="240" background-color="#000000"/>
            <region id="whole"     left="0"   top="0"  width="320" height="240" z-index="0"/>
            <region id="leftpane"  left="16"  top="16" width="136" height="208" z-index="1"/>
            <region id="rightpane" left="168" top="16" width="136" height="208" z-index="1"/>
        </layout>
        <transition id="diagwipe" type="wipe" subtype="diagonalRightDown" dur="1s"/>
    </head>
    <body>
        <seq>
            <par>
                <img src="back1.jpg"  dur="10s" region="whole"/>
                <img src="left1.jpg"  dur="10s" region="leftpane"/>
                <img src="right1.jpg" dur="10s" region="rightpane"/>
            </par>
            <par>
                <img src="back2.jpg"  dur="10s" region="whole" transition="diagwipe"/>
                <img src="left2.jpg"  dur="10s" region="leftpane"/>
                <img src="right2.jpg" dur="10s" region="rightpane"/>
            </par>
        </seq>
    </body>
</smil>
```

In Cases 2 and 3, however, we want to apply the transition to the "whole" region and either have the "leftpane" and "rightpane" regions clip (Case 3) or not clip (Case 2) the transition. In order to express Cases 2 and 3, there are three additional syntactical concepts which need to be added to SMIL:

1. *Parent regions*. How do we do this in CSS? As defined in the Layout Module, the <region> element may be a structural child of another <region> element, as in the following SMIL fragment:

   ```
   <region id="parent" ... >
       <region id="child1" ... />
       <region id="child2" ... />
       ...
   </region>
   ```

2. *Allow child regions to clip their parent*. How do we do this in CSS? In SMIL, we will accomplish this by allowing adding a "childrenClip" attribute to the <region> element. If childrenClip="true", then operations on the parent region (such as transitions) will be clipped by the child regions. By default, childrenClip would be "false".
3. *Lightweight media object element*. We introduce a new media object element called <brush>. This element can be used to place lightweight media such as solid fills or pattern fills in regions at points along the timeline. For our purposes, we will use it to place transitions on the timeline *for transitions which include multiple media elements*.

The reason for introducing parent regions and a lightweight media object element is to maintain consistency with single-media-element transitions. In single-media-element transitions, we associate a transition with a media object element which in turn is associated with a playback region. This, by inference, makes a one-to-one mapping between transition and playback region. Therefore, in order to have transitions which incorporate multiple media objects (and thus multiple regions), we associate a transition with a lightweight media object which is then associated with a parent region.

A secondary purpose of the parent region is to define the bounding rectangle for transitions which will involve multiple media objects. An alternative would be to assume that the bounding rectangle is defined by the convex hull (the minimum bounding rectangle) of the set of all the regions involved. However, formally defining a parent region is simpler and more flexible.

12.5.1 The `<brush>` element

The `<brush>` element is a lightweight media object element, meaning that the media object is completely procedural. Since this media object is procedural, then all the information about the media object is specified in the attributes of the element itself. Therefore does not have to specify a "src" attribute.

Element attributes

color
: The use and definition of this attribute are identical to the "background-color" property in the CSS2 specification, except that the `<brush>` element does not require support for "system colors".

transition
: This attribute is the same as the transition attribute which was added to all media object elements and defined previously.

The `<brush>` element also supports all timing and synchronization attributes which all other media objects support (see the SMIL Timing and Synchronization Module and the SMIL Media Object Module for a list of these attributes).

Element content

The `<brush>` element is an empty element.

Now, armed with these new syntactical constructs, we can express Cases 2 and 3 in SMIL. First, Case 2:

```
<smil>
    <head>
        <layout>
            <root-layout width="320" height="240" background-color="#000000"/>
            <region id="whole" left="0" top="0" width="320" height="240" z-index="0">
                <region id="leftpane"  left="16"  top="16" width="136" height="208" z-index="1"/>
                <region id="rightpane" left="168" top="16" width="136" height="208" z-index="1"/>
            </region>
        </layout>
        <transition id="diagwipe" type="wipe" subtype="diagonalRightDown" dur="1s"/>
    </head>
    <body>
```

```
        <seq>
            <par>
                <img src="back1.jpg"  dur="10s" region="whole"/>
                <img src="left1.jpg"  dur="10s" region="leftpane"/>
                <img src="right1.jpg" dur="10s" region="rightpane"/>
            </par>
            <par>
                <brush                 dur="10s" region="whole" transition="diagwipe"/>
                <img  src="back2.jpg"  dur="10s" region="whole"/>
                <img  src="left2.jpg"  dur="10s" region="leftpane"/>
                <img  src="right2.jpg" dur="10s" region="rightpane"/>
            </par>
        </seq>
    </body>
</smil>
```

Note the following changes from other cases. First, we have made the "leftpane" and "rightpane" regions to be *children* of the "whole" region. Second, we have placed the <brush> element on the timeline and associated the transition with it. Now, Case 3 is a trivial change from Case 2:

```
<smil>
    <head>
        <layout>
            <root-layout width="320" height="240" background-color="#000000"/>
            <region id="whole" left="0" top="0" width="320" height="240" z-index="0" childrenClip="true">
            <region id="leftpane"  left="16"  top="16" width="136" height="208" z-index="1"/>
            <region id="rightpane" left="168" top="16" width="136" height="208" z-index="1"/>
            </region>
        </layout>
        <transition id="diagwipe" type="wipe" subtype="diagonalRightDown" dur="1s"/>
    </head>
    <body>
        <seq>
            <par>
                <img src="back1.jpg"  dur="10s" region="whole"/>
                <img src="left1.jpg"  dur="10s" region="leftpane"/>
                <img src="right1.jpg" dur="10s" region="rightpane"/>
            </par>
            <par>
                <brush                 dur="10s" region="whole" transition="diagwipe"/>
                <img  src="back2.jpg"  dur="10s" region="whole"/>
                <img  src="left2.jpg"  dur="10s" region="leftpane"/>
                <img  src="right2.jpg" dur="10s" region="rightpane"/>
            </par>
        </seq>
    </body>
</smil>
```

where all we have done is added childrenClip="true" to the declaration of the parent "whole" region.

12.6 Appendix A: Open Issues

1. Transitions vs. Effects - should we also be able to specify non-time-based effects? These are not transitions between two different media types, but do fit nicely into the idea of style. In other words, an "emboss" effect might just be a style on a particular element. It doesn't have a duration, but it could have a start.
2. How do we make the set of transitions and transition parameters extensible?
3. Is is possible to come up with a language for describing *what* a transition is, rather than just enumerating a list of types?
4. Not all transition parameters apply to all transition subtypes - how to we enforce this?
5. Some transitions (like fading to a color) really only make sense at the end of a

timeline. How do we enforce this?

6. How do we express the notion of parent regions in CSS?
7. Implementing all of the transitions in this document is a daunting task. Do we need to define some "baseline" transition subset which would be required for compliance?
8. Do the child regions have to be completely geometrically contained by the parent? If they do not have to be, then what is the behavior of the transition across the regions (or partial regions) which are structural children but not geometric children?
9. We implicitly assume that the lightweight media object will be placed by the author at the appropriate place in the timeline. What happens if the timeline of the lightweight media object doesn't sync with a timeline of any media objects?

13. The SMIL Document Object Model Module

Editors:
Philippe Le Hégaret, *W3C*
Patrick Schmitz, *Microsoft*

13.1 Abstract

This specification defines the Document Object Model (DOM) specification for synchronized multimedia functionality [SMIL-DOM]. It is part of work in the Synchronized Multimedia Working Group (SYMM) towards a next version of the SMIL language and SMIL modules. Related documents describe the specific application of this SMIL DOM for SMIL documents and for HTML and XML documents that integrate SMIL functionality. The SMIL DOM builds upon the DOM Core functionality, adding support for timing and synchronization, media integration and other extensions to support synchronized multimedia documents.

14. SMIL Boston Language Profile

Editors:
Nabil Layaida (Nabil.Layaida@inrialpes.fr), INRIA
Jacco.van.Ossenbruggen (Jacco.van.Ossenbruggen@cwi.nl), CWI

14.1 Open issues

Should we remove the current profile definitions from the SMIL modules draft, or should we integrate the SMIL language profile into SMIL modules?
Should the profile define a minimal list/recommended of media types? see: Baseline formats .

Should we allow subsetting/splitting of modules (basic/Boston layout module)?

Does the profile require support for some or all features of SMIL Boston? If it requires some, what features are not required If it requires all, is it realistic to expect that someone will really implement the full Boston Language Profile in the near future (including DOM, transitions, animation)

Should we define:

- level of DOM support
- level of XPointer support
- transfer protocols?

See in-line for more remarks.

14.2 Abstract

The SMIL Boston profile describes the SMIL modules that are included and details how this modules are integrated. It contains all of the SMIL Boston features including animation, content control, layout, linking, media object, meta-information, structure, timing and transition effects modules. It is designed for Web clients that support direct SMIL Boston markup such as standalone multimedia players.

14.3 SMIL Boston Profile

This section is *informative.*

The SMIL Boston Profile is defined as a markup language. The syntax of this language is formally described with a document type definition or Schema which are based on SMIL modules as defined in "Modularization of SMIL" [SMIL-MOD]

The SMIL Boston Profile design requirements are:

1. Ensure that the profile is completely backward compatible with SMIL 1.0. (check this)
2. Ensure that all the modules' semantics maintain compatibility with SMIL semantics (this includes content and timing).
3. Adopt new W3C recommendations when appropriate and not in conflict with other requirements. (check against both Schemas and CC/PP, align with XHTML)
4. Specify how the modules support the document object model. (Define specific level of DOM support)

14.4 Normative Definition of SMIL Boston

This section is *normative.*

14.4.1 Document Conformance

A *conforming* SMIL Boston document is a document that requires only the facilities described as mandatory in this specification. Such a document must meet all of the following criteria:

1. It must validate against the DTD (or Schema?) found in Appendix A
2. The root element of the document must be `<smil>`.
3. The name of the default namespace on the root element must be the SMIL Boston namespace name, (TBD) `http://www.w3.org/2000/smil`
4. There must be a DOCTYPE declaration in the document prior to the root element. The public identifier included in the DOCTYPE declaration must reference the DTD ot schema (TBD) found in Appendix A using its Formal Public Identifier. The system identifier may be modified appropriately.

```
<!DOCTYPE SMIL-Boston PUBLIC "-//W3C//DTD SMIL Boston //EN"
          "smil-boston.dtd">
```

14.4.2 User Agent Conformance

The user agent must conform to the following user agent rules :

@fill in here requirements.

14.4.3 SMIL-Boston Profile

The SMIL-Boston Profile supports the timeline-centric multimedia features found in SMIL language. This profile includes the following SMIL modules:

- Animation Module
- Content Control Module
- Layout Module

- Linking Module
- Media Object Module
- Metainformation Module
- Structure Module
- Timing and Synchronization Module
- Transition Effects Module

Is it realistic to expect that someone will really implement this in the near future (including full transitions, animation, DOM)? Check this with implementers. Chairman: Yes, we should require what people will actually implement. If the group wants to make certain features option, that is up for discussion.

14.4.4 Animation Module

The Animation Module provides a framework for incorporating animation onto a timeline (a timing model) and a mechanism for composing the effects of multiple animations (a composition model). The Animation Module defines semantics for the animate, set, animateMotion, and animateColor elements:

Elements	Attributes	Minimal Content Model
animate	TBD	TBD
set	TBD	TBD
animateMotion	TBD	TBD
animateColor	TBD	TBD

This module adds the animate, set, animateMotion, and animateColor elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the content model of the body element of the Structure Module.

Integration issues with animation

We need to think about how animation applies to SMIL. It should be possible to animate regions, and so animation will apply to the elements of layout. Animating the time containers is interesting, but likely beyond what we want to do here. What properties of media elements are interesting to animate? How about the URL's of media objects? There is much up for discussion here.

14.4.5 Content Control Module

The Content Control Module provides a framework for selecting content based on a set of test attributes. The Content Control Module defines semantics for the switch element.

Elements	Attributes	Minimal Content Model
switch	Common, Timing	TBD

This module adds the switch, element to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds this element to the content model of the body element of the Structure Module. It also adds this element to the content model of the a element of the Linking Module. It also adds this element to the content model of the head element of the Structure Module.

The Content Control Module defines the Attribute set "Test".

Collection Name	Attributes in Collection
Test	systemBitrate (Number), systemCaption (on|off), systemLanguage (CDATA), systemOverdubOrCaption (caption|overdub), systemRequired (URI), systemScreenSize (CDATA), systemScreenDepth (CDATA), systemOverdubOrSubtitle (overdub|subtitle), systemAudioDesc (on|off), systemComponent (CDATA),

We also want to include the test-attributes, which can be on elements within or outside of a switch, the usergroups, and the prefetch element.

14.4.6 Layout Module

The Layout Module provides a framework for spatial layout of visual components. The Layout Module defines semantics for the layout, root-layout, and region elements.

We may want to split layout up, but this will not be done for this draft. This shouldn’t affect this profile, since all of the layout will likely be included in the full profile (as opposed to the Basic profile).

Elements	Attributes	Minimal Content Model
region	backgroundColor,bottom, fit (fill | hidden | meet | scroll | slice), width, height, left, right, title, top, volume, z-index,	TBD
root-layout	backgroundColor, width, height, skip-content, title	None
top-layout(*)	backgroundColor, width, height, skip-content, title	region, None
layout	TBD**	root-layout, region, top-layout

(*) If the type attribute of the "layout" element has the value "text/smil-basic-layout", it can contain the "region" and the "root-layout" elements. If the type attribute of the layout element has the value "text/smil-extended-layout", in addition to the "layout" and "root-layout" elements it can contain the "top-layout" element.

(**) The "background-color" attribute of SMIL1.0 is deprecated in favor of "backgroundColor".

This module adds the layout element to the content model of the head element of the Structure Module. It also adds this element to the content model of the switch element of the Content Control Module.

Probably need more explanation here as to how modules add to each other through the integration profile. Any suggestions for a good format? Maybe define in both sections: briefly note in the section adding functionality, and fully describe in the section having functionality added.

14.4.7 Linking Module

The Linking Module provides a framework for relating documents to content, documents and document fragments. The Linking Module defines semantics for the a and area elements.

Both the a and area elements have an "href" attribute, whose value should be a valid URI. Support for URI's using http:// and file:/ access protocols is required. Support for other protocols is optional.

Make support for RT(S)P required? Chairman: How about if RTP/RTSP is supported by the implementation, then the markup must be supported. If not, then the rtsp attributes/elements are ignored. This is the kind of thing that the profile has to nail down).

Support for URI's with XPointer fragment identifier syntax is not required.

Elements	Attributes	Minimal Content Model
a	href, sourceVolume, destinationVolume, sourcePlaystate (play | pause | stop), destinationPlaystate, show (new | replace), accesskey, tabindex , target, actuate, Common, Timing, Test	Media Objects, Time Container Elements,
area	coords, sourceVolume , destinationVolume, sourcePlaystate, destinationPlaystate, show, accesskey, tabindex, target, Common, Timing,Test	Empty

This module adds the area and a elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the content model of the body element of the Structure Module.

SMIL 1: The <anchor> element is deprecated in favor of <area>.

SMIL 1: The show attribute value "pause" is deprecated in favor of setting the the "show" attribute to "new" and the "sourcePlaystate" attribute to "pause".

Chairman: need to define what "adding to the content model" means. This is not fully descriptive, since the time containers can be children of media elements, etc.

14.4.8 Media Object Module

The Media Object Module provides a framework for declaring media. The Media Object Module defines semantics for the ref, animation, audio, img, video, text, and textstream elements.

Should the profile define a minimal list/recommended of media types?
see: Baseline formats .

In the SMIL Boston Language Profile, media object elements can have the following attributes, in addition to the attributes defined in the SMIL Media Object Module:

dur
: Defined in the SMIL Timing Module

end
: Defined in the SMIL Timing Module

fill
: For a definition of the semantics of this attribute, see SMIL Timing Module. The attribute can have the values "remove" and "freeze".

id
: This attribute uniquely identifies an element within a document. Its value is an XML identifier.

region
: This attribute specifies an abstract rendering surface (either visual or acoustic) defined within the layout section of the document. Its value must be an XML identifier. If no rendering surface with this id is defined in the layout section, the values of the formatting properties of this element are determined by the default layout.

In the SMIL Boston Language Profile, media object elements can contain the following elements:

anchor
: Defined in Linking Module

area
: Defined in Linking Module

par
: Defined in Timing Module

seq
: Defined in Timing Module

excl
: Defined in Timing Module

animate
: Defined in Animation Module

set
: Defined in Animation Module

animateColor
: Defined in Animation Module

animateMotion
: Defined in Animation Module

rtpmap
: Defined in the Media Object Module

param
: defined in the Media Object Module

Can this be moved to an appendix?

Changes from SMIL 1.0

SMIL 1.0 only allowed "anchor" as a child element of a media element. In addition to "anchor", the following elements are now allowed as children of a SMIL media object:

area, anchor
: Defined in Linking Module

par, seq, excl
: Defined in Timing Module

param, rtpmap
: Defined in Media Object Module

animate, set, animateColor, animateMotion
Defined in Animation Module

Elements	Attributes	Minimal Content Model
ref	TBD	TBD
img, text	TBD	TBD
audio, video, animation, textstream	TBD	TBD

This module adds the ref, animation, audio, img, video, text, and textstream elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the content model of the body element of the Structure Module. It also adds these elements to the content model of the a element of the Linking Module.

14.4.9 Metainformation Module

The Metainformation Module provides a framework for describing a document, either to inform the human user or to assist in automation. The Metainformation Module defines semantics for the meta and metadata elements.

Elements	Attributes	Minimal Content Model
meta (TBD)	base, pics-label (or PICS-Label), title, xml:lang, http-equiv, scheme	None
metadata		RDF

This module adds the meta element to the content model of the head element of the Structure Module.

14.4.10 Structure Module

The Structure Module provides a framework for structuring a SMIL document. The Structure Module defines semantics for the smil, head, and body elements.

Elements	Attributes	Minimal Content Model
smil	Core, Accessibility, xmlns	head?, body?, metadata?
head	Core, Accessibility, profile	meta*, (switch | layout)?
body	Core, Accessibility	(Schedule | MediaContent | MediaControl | LinkAnchor)*

The Attribute collections in this table are defined as follows

Core
id (ID),
class (NMTOKEN)

Accessibility
xml:lang (NMTOKEN),
title (CDATA)

The collections in the table from the Content Model of the body element are defined as follows

Schedule
par, seq, excl

MediaContent
ref, audio, video, img, animation, text, and textstream

MediaControl
switch

LinkAnchor
a, area

The body element acts as the root element to span the timing tree. The body element has the schedule semantics of a time container equal to that of the "seq" element from the timing and synchronization module. This module is a mandatory part in any profile family labeled "SMIL".

14.4.11 Timing and Synchronization Module

The Timing and Synchronization Module provides a framework for describing timing structure, timing control properties, and temporal relationships between elements. The Timing and Synchronization Module defines semantics for par, seq, and excl elements. In addition, this module defines semantics for attributes including begin, dur, end, repeatCount, repeatDur, etc.

Elements	Attributes	Minimal Content Model
par, seq, excl	TBD	TBD
	begin, end, dur, repeatCount, repeatDur, TBD	TBD

This module is mandatory in any profile incorporating SMIL modules.

14.4.12 Transition Effects Module

Elements	Attributes	Minimal Content Model
TBD	TBD	TBD

@TBD This module is used, and it adds the TBD element to the content model of the layout element of the Layout Module.

14.5 Document Type Definition

This section is *normative*.

The SMIL Boston document type is defined as a set of SMIL Boston modules. All SMIL Boston modules are integrated according to the guidelines in the "*Modularization of SMIL Boston*" specification [SMIL-MOD], and defined within their respective module sections.

14.6 Appendix A: Document Type Definition or XML Schema

This section is *normative*.

TBD. May instead be an XML Schema.

15. HTML+SMIL Language Profile

Editor:
Patrick Schmitz (pschmitz@microsoft.com), Microsoft

15.1 Abstract

The HTML+SMIL profile integrates a subset of the SMIL Boston specification with HTML. It includes the SMIL Boston modules supporting animation, content control, linking, media objects, timing an synchronization, and transition effects. The SMIL Boston features are integrated directly with HTML and CSS, and can be used to manipulate HTML and CSS features. It is designed for Web clients that support HTML+SMIL markup.

The document type definition or Schema is implemented using SMIL modules as defined in "Modularization of SMIL" [SMIL-MOD].

15.2 Introduction

This section is *informative.*

This profile describes the SMIL modules that are included, and details the integration issues. The language integration includes the complete set of XHTML 1.1 modules. @@ We really need aspects of XHTML 2.0, but that is not very far along yet).

Throughout the document, where reference is made to "HTML" functionality and elements, this should be understood to refer to XHTML modules and elements.

15.2.1 Motivation and applications

Some notes on why we are doing this.

15.2.2 Design Rationale

This section explains why certain modules of SMIL Boston are not included. The general philosophy is to use XHTML modules where appropriate.

Layout

The SMIL Boston layout module is not included, as HTML and CSS provide layout functionality. Authors are already familiar with the HTML/ CSS layout model, and it provides the tools authors need.

Structure

The SMIL Boston structure module is not included, as the HTML document is defined to be the host language, and so provides the equivalent elements and semantics.

Meta information

The SMIL Boston meta information module is not included, as XHTML provides the equivalent elements and semantics.

15.3 Normative Definition of SMIL Boston

This section is *normative.*

15.3.1 Document Conformance

A *conforming* HTML+SMIL document is a document that requires only the facilities described as mandatory in this specification. Such a document must meet all of the following criteria:

1. It must validate against the DTD (in future the Schema) found in Appendix A
2. The root element of the document must be <html>.
3. The name of the default namespace on the root element must be the HTML+SMIL namespace name, http://www.w3.org/2000/htmlplussmil (This must still be verified)
4. There must be a DOCTYPE declaration in the document prior to the root element. If present, the public identifier included in the DOCTYPE declaration must reference the DTD found in Appendix A using its Formal Public Identifier. The system identifier may be modified appropriately.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML+SMIL //EN"
          "htmlplussmil.dtd">
```

15.3.2 User Agent Conformance

The user agent must conform to the "User Agent Conformance" section of the XHTML specification ([XHTML10], section 3.2).

The user agent must conform to the following user agent rules :

@other requirements?

15.3.3 HTML+SMIL Profile

The HTML functionality in the HTML+SMIL document type is based upon XHTML 1.1 modules and associated document type. The XHTML 1.1 document type is made up of the following abstract modules, as defined in XHTMLMOD [XMOD], and the Ruby Annotation module as defined in [RUBY]. The formal definition of the modules is not repeated here, but only the extensions introduced with timing. The notation, terms and document conventions used here are borrowed from [XMOD].

Structure Module*
: body, head, html, title

Basic Text Module*
: abbr, acronym, address, blockquote, br, cite, code, dfn, div, em, h1, h2, h3, h4, h5, h6, kbd, p, pre, q, samp, span, strong, var

Hypertext Module*
: a

List Module*
: dl, dt, dd, ol, ul, li

Applet Module
: applet, param

Presentation Module
: b, big, hr, i, small, sub, sup, tt

Edit Module
: del, ins

BDO Module
: bdo

Forms Module
: form, input, select, option, textarea, button, fieldset, label, legend, optgroup

Tables Module
: caption, table, td, th, tr, col, colgroup, tbody, thead, tfoot

Image Module
: img

Client-side Image Map Module
: area, map, (attribute extensions)

Server-side Image Map Module
: (attribute extensions)

Object Module
: object, param

Frames Module
: frameset, frame, noframes, (attribute extensions)

Iframe Module
: iframe

Intrinsic Events Module
: (attribute extensions)

Meta Information Module
: meta

Scripting Module
: noscript, script

Stylesheet Module
: style

Link Module
: link

Base Module
    base
Legacy Module
    font, s, strike, u, (attribute extensions)

(*) = This module is a required [XHTMLFamily] module.

@@We will also include the new Events module [XHTMLEvents] as it is completed.

In addition, the HTML+SMIL document type supports the timeline-centric multimedia features defined by SMIL Boston. The profile includes the following SMIL Boston modules:

- Animation Module
- Content Control Module
- Linking Module
- Media Object Module
- Timing and Synchronization Module
- Transition Effects Module

15.3.4 Animation Module

The Animation Module provides a framework for incorporating animation onto a timeline (a timing model) and a mechanism for composing the effects of multiple animations (a composition model). The Animation Module defines semantics for the animate, set, animateMotion, and animateColor elements:

Elements	Attributes	Minimal Content Model
animate	Common, Timing, attributeName, attributeType, additive, accumulate, calcMode, values, keyTimes, keySplines, from, to, by	EMPTY
set	Common, Timing, attributeName, attributeType, to	EMPTY
animateMotion	Common, Timing, additive, accumulate, calcMode, values, keyTimes, keySplines, from, to, by, path, origin	EMPTY
animateColor	Common, Timing, attributeName, attributeType, additive, accumulate, calcMode, values, keyTimes, keySplines, from, to, by	EMPTY

This module adds the animate, set, animateMotion, and animateColor elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It adds the animate, set, and animateColor elements to the content model of the elements of the Basic Text, Hypertext, List, Applet, Presentation, Edit, Tables, Image, Client-side Image Map, Server-side Image Map, Object and Legacy modules. It adds the animateMotionelement to the content model of the elements of the Basic Text and Image modules (and possibly a few others - need to nail this down).

This module defines the following content sets:

BaseAnimation
: animate | set | animateColor

AllAnimation
: BaseAnimation | animateMotion

Additional integration issues with animation

- Need to talk about the legal attributes that can be targetted and the associated values.
- Need to talk about the positioning model and constraints for animateMotion.
- Need to talk about the legal attributes for animateColor
- Need to talk about the numeric forms for things like keyTimes and keySplines (i.e. specific floating point syntax supported).

See also the Integration requirements from the SMIL Boston Animation module.

15.3.5 Content Control Module

The Content Control Module provides a framework for selecting content based on a set of test attributes. The Content Control Module defines semantics for the switch element.

Elements	Attributes	Minimal Content Model
switch	Common, Timing	Flow

This module adds the switch element to the Flow content set of the Basic Text module. It also adds the Test attributes set to the elements in the Flow content set of the Basic Text Module (as modified by all included modules).

The Content Control Module defines the Attribute set "Test".

Collection Name	Attributes in Collection
Test	systemBitrate (Number), systemCaption (on|off), systemLanguage (CDATA), systemOverdubOrCaption (caption|overdub), systemRequired (URI), systemScreenSize (CDATA), systemScreenDepth (CDATA), systemOverdubOrSubtitle (overdub|subtitle), systemAudioDesc (on|off), systemComponent (CDATA),

15.3.6 Linking Module

The SMIL Boston linking module is isomorphic to functionality defined in the XHTML modules. However, it adds some additional attributes and semantics to the a and area elements in XHTML. Rather than describing the elements that are added, the additional functionality defined on the HTML elements a and area is described.

Elements	Attributes	Minimal Content Model
a&	sourceVolume, destinationVolume, sourcePlaystate, destinationPlaystate, actuate, show, Test, Timing	n/a
area&	sourceVolume, destinationVolume, sourcePlaystate, destinationPlaystate, actuate, show, Test, Timing	n/a

This module adds the area element to the content model for the elements in the Media module.

15.3.7 Media Object Module

The Media Object Module provides a framework for declaring media. The Media Object Module defines semantics for the ref, animation, audio, img, video, text, and textstream elements. XHTML defines an img element, and so the integration of the Media module extends the semantics and content model of this element.

Elements	Attributes	Minimal Content Model
ref	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param
animation	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param
audio	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param
img&	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param
video	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param
text	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param
textstream	Common, Test, Timing, timeContainer	AllAnimation, TimeContainers, area, param

This module adds the ref, animation, audio, img, video, text, and textstream elements to the content model of the par, seq, and excl elements of the Timing and Synchronization Module. It also adds these elements to the Inline content set of the Basic Text Module.

15.3.8 Timing and Synchronization Module

The Timing and Synchronization Module provides a framework for describing timing structure, timing control properties, and temporal relationships between elements.

In addition to the data types defined by XHTML Modularization, The HTML+SMIL profile defines the TimeActions data type and its semantics, described in the following table:

Data type	Description
TimeActions	Authors may use the following recognized time actions, listed here with their interpretations. **none** Specifies that no action is performed on the element when it is active. This is only legal for time containers, and allows the author to introduce a new time space without affecting the document presentation. **visibility** Specifies that the CSS "visibility" property is to be manipulated over time. When the element is neither active nor frozen, the property is set to "hidden". When it is active, the original value is used. If the original value is "hidden", the time action will have no effect (i.e. the element will remain hidden). See also [CSS2] **display** Specifies that the CSS "display" property is to be manipulated over time. When the element is neither active nor frozen, the property is set to "none". When it is active, the original value is used. If the original value is "none", the time action will have no effect (i.e. the element will remain out of view and layout). See also [CSS2] **style** Specifies that the CSS inline style attribute "style" is to be removed and applied over time. When the element is neither active nor frozen, the inline stylesheet is cleared (so no style modification is made). When it is active, the original value (i.e. the string attribute value specified for the "style" attribute) is used. If there is no inline style attribute on the element, the time action will have no effect. See also [CSS2] **class:*classname*** Specifies that the name "*[classname]*" will be *removed from* and *added to* the value of the "class" attribute over time. When the element is neither active nor frozen, the specified string (i.e. whatever the author specifies for "*classname*") is removed from the value of the class attribute (if it was included). When the element is active, the specified string is added to the value of the class attribute. Note that any other values specified in the class attribute are not affected. Note also that the application of any associated style must be specified in a stylesheets for the document. Note finally that the application of styles depends not upon the order of names in the class attribute, but rather on the order of the rules in the stylesheets for the document. See also [CSS2].

The Timing and Synchronization Module defines the Attribute sets "Timing" and "RuntimeSync".

Collection Name	Attributes in Collection
Timing	begin (CDATA), dur (CDATA), repeatCount (CDATA), repeatDur (CDATA), end (CDATA), fill (CDATA), restart (CDATA), timeAction (TimeActions), onBegin (Script), onEnd (Script), onRepeat (Script)
RuntimeSync	syncBehavior (locked|canSlip), syncTolerance (CDATA), syncMaster (true|false),

The Timing and Synchronization Module adds the Timing and RuntimeSync attribute set to the elements in the Media Module, and adds the Timing attribute set to the Flow content set of the Basic Text Module (as modified by all included modules).

The Timing and Synchronization Module defines the elements par, seq, and excl.

Elements	Attributes	Minimal Content Model
par	Common, Test, Timing, RuntimeSync, timeAction (CDATA)	par, seq, excl, Flow
seq	Common, Test, Timing, RuntimeSync, timeAction (CDATA)	par, seq, excl, Flow
excl	Common, Test, Timing, RuntimeSync, timeAction (CDATA)	par, seq, excl, Flow

This module adds the par, seq, and excl elements to the Inline content set of the Basic Text, Hypertext and Tables Modules.

As part of the integration of timing and synchronization functionality with HTML, two additional attributes are defined for many of the HTML elements: timeContainer and timeAction. This module adds the timeContainer attribute to the elements of the Flow content set of the Basic Text Module (as modified by all included modules).

The timeAction attribute defines the behavior that is controlled by the timing model. The default depends upon the type of element. A special value "none" is reserved for use with the time container elements and with elements that have been set to be a time container (using "`timeContainer=`[par,seq,excl]").

The following table presents the default time actions. Those modules and elements that are not included do not have a defined time behavior, and cannot legally support timing attributes, or participate in the time model.

Certain elements have a reasonable notion of intrinsic behavior that can be controlled over time. This is generally some presentation or behavioral effect, such as the font style controls of the `<b>` and `<strong>` elements, and the click sensitivity of the `<a>` and `<area>` elements. One way to logically model the control of intrinsic behavior is to convert the element to a `<span>` when it is neither active nor frozen, and to use the original element when it is active or frozen.

Many other elements simply contain content and so default to controlling the "visibility" property for the element. In some cases, an element may have a presentational effect (e.g. the Ruby module elements), but be modeled as a content element. The decision is based upon the usefulness in common authoring scenarios of controlling the presentational behavior in isolation.

For those elements that default to controlling "visibility", setting timeAction to any other value *overrides* this, and will *only control* the specified timeAction (and not the visibility). For all other elements, the timeAction will control the default (intrinsic) behavior *as well as* the indicated timeAction behavior.

In addition, for those elements that default to "visibility", when they are children of a sequence time container `<seq>` or an element with "`timeContainer=seq`", the default timeAction is "display". This more closely matches the expected behavior of the SMIL Language profile.

Module	Elements	Default time action
Structure	body	"none"
Media	*(all)*	*schedule and render*
Timing	*(all)*	"none"
Text	em	*intrinsic effect*
Text	kbd	*intrinsic effect*
Text	strong	*intrinsic effect*
Text	var	*intrinsic effect*
Text	*(all others)*	"visibility"
Hypertext	a	*link sensitivity*
Lists	*(all)*	"visibility"
Applet	applet	"visibility"
Presentational	*(all)*	*intrinsic effect*
Edit	*(all)*	*intrinsic effect*
Forms	*(all)*	"visibility"
Tables	*(all)*	"visibility"
Image Map	area	*link sensitivity*
Object	object	"visibility"
Iframe	iframe	"visibility"
Ruby	*(all)*	"visibility"
Legacy	*(all)*	*intrinsic effect*

All modules not listed in the table, and all Structure module elements except body do not support timing.

Additional integration issues with Timing

- Define document begin and end
- Define document presentation

15.3.9 Transition Effects Module

The Transition Effects Module defines a taxonomy of transition effects as well as semantics and syntax for integrating these effects into XML documents

Elements	Attributes	Minimal Content Model
TBD	TBD	TBD

This module is used, it adds the TBD element to the content model of the layout element of the Layout Module.

15.4 Appendix A: Document Type Definition

This section is *normative*.

TBD.

16. Requirements for a SMIL Basic Profile

Editors
Kenichi Kubota (kuboken@Research.Panasonic.COM), Panasonic
Aaron Cohen (aaron.m.cohen@intel.com), Intel

16.1 Abstract

This document describes the requirements for a SMIL Basic profile, which is intended to meet the needs of low power devices such as mobile phones and other information appliances. It includes comments and discussion about the need for modules that only include the basic syntax and semantics of the SMIL language profile.

16.2 Introduction

16.2.1 SMIL and Modularization

The SMIL language [SMIL10][SMIL-BOSTON] includes powerful functionality for multimedia services not only on desktops but also for information appliances. SMIL content authors may wish their works to be available on a widespread variety of web clients, such as desktops, television sets, PDA's, mobile phones, car navigation systems and voice user agents. Each of these platforms has its specific capabilities and may require its own profile. The SMIL Modularization draft [SMIL-MOD] provides a solution to create subsets and extensions of the full SMIL language profile, in addition to providing the means to integrate SMIL functionality into other languages.

The HTML group has demonstrated the effectiveness of modularization in working on module-based XHTML [XHTML11], [XMOD], [MODMOD]. They also shed light on the path towards greater interoperability of content among various user agents, by making the requirement that the document profile of the content act as a basis for interoperability guarantees [XHTML-PROF-REQ], by issuing guidelines for HTML content [MOBILE-GUIDE], and by providing a minimum subset for portability and conformance [XHTML-BASIC].

16.2.2 Neeed for a SMIL Basic profile

Mobile phones with web browsers have come on to the Internet providing seamless connectivity with fairly broad bandwidth, enabling multimedia services. They might become the smallest SMIL compliant devices. Internet access with mobile phones, however, has some characteristics specific to the wireless environment and its hardware and software constraints. Therefore it seems necessary for the SYMM Working Group to consider a lightweight SMIL profile for mobile phones and to provide a basis for interoperability guarantees between the cutting-edge full SMIL profile and the mobile profile.

The CC/PP [CC/PP] mechanism provides a way to notify device capabilities and user preferences to an origin server and/or intermediaries such as a gateway or proxy, that allows us to generate or select or transform tailored contents. Thus CC/PP can be used with transformation of available documents between client and server. However, as for clients, it is unclear which kind of profile they should support?

Content authors wish their SMIL documents to be delivered to as wide an audience as possible. Client users expect to enjoy SMIL presentations with various kinds of devices, which may have different profiles. Therefore, some consensus between these profiles is useful. The SMIL Basic profile would provide that minimal profile.

"XHTML Document Profile Requirements" [XHTML-PROF-REQ] describes a framework for content negotiation, and requirements for document profiles from the viewpoint of content developers and designers of different kinds of web user agents.

The SMIL Basic profile would consist of a reduced set of modules among the variety of SMIL modules in terms of semantics and syntax, and assures conformance by using a subset of the full SMIL specification. It also describes appropriate behavior of a conforming user agent. A profile for mobile devices can be tailored according to SMIL Basic profile with or without extensions for mobile specific features, and entitled "SMIL Basic". The SMIL Basic profile will provide the same benefits for other devices such as PDA's and TV sets. It should serve as a means of providing SMIL across a wide range of mobile and, further, personal electronic devices.

The Basic Profile does not propose to restrict extension, but aims at a baseline of conformance between the full SMIL and one appropriate for mobile services.

16.3 Requirements for SMIL Basic Profile

16.3.1 Target Devices

SMIL Basic profile must be a client profile which can be supported by wide variety of SMIL players, even those running on small mobile phones. Mobile devices share some common characteristics:

1. **Small display**, Display can render texts, images, audio and stream data in a small area.
2. **Simple input method**, Input devices are numeric keys, 4-way arrow keys, and a select key. Some may have a pointing cursor. A few may have a voice interpreter.
3. **Real-time embedded OS**, Resources for calculation is limited by priority order of each task. So, in a SMIL player, the use of timers should be restricted in number and frequency, and memory should be used sparingly.
4. **Wireless network transaction**, Network transactions should be reduced as much as possible.

@@ This is written to promote reader's understanding of mobile environment.

The following media could be supported:

- text -- plain text, HTML
- img -- GIF, JPEG, PNG
- audio -- voice, MIDI
- video -- MPEG4

This section states requirements for the SMIL Basic profile starting from these points.

16.3.2 Generic requirements

The generic requirements to be considered are that the SMIL Basic profile should:

1. Consist of a comfortably small set of modules chosen from the full set of SMIL modules in terms of required semantics and syntax.
2. Assure conformance to the cutting edge SMIL.
3. Describe appropriate behavior for a conforming user agent according to SMIL syntax and semantics.

16.3.3 User Interface

On a SMIL player window, the user would handle arrow keys to move focus on objects and anchors, and select the target which activates playback or linking. A pointing cursor "mouse-like" device might not be supported. So "move focus and select" is a simple user interface for communication with the SMIL player.

Requirement

- Users can enjoy SMIL Basic presentations with just focus and selection control.
- Users can enjoy SMIL Basic presentations with numeric keys.

Suggestion

- The "`accesskey`" attribute and "`keypress`" events could be supported.
- Event actions between hyperlink and playback control should not conflict.

@@These suggestions are not a complete solution however.

16.3.4 Timing and Synchronization

SMIL Timing and Synchronization presents dynamic and interactive multimedia according to a timeline. The SMIL timing model is expressed in a structured language effectively. The timeline normally needs to be calculated with limited resources of memory and the processor. For example, recursive function calls, caused by nesting elements, or memory allocation, say by additional timelines, at any moment should be restricted. The timeline should be a simple and single sequence of media objects. To achieve these, the single time container (use of par or seq) and single timing attribute value may be preferable. In a list of timing attribute values, events can be activated with "`click`" and also with numeric keys familiar in phones or TV controllers. For the simplest documents, if timing of "`begin`", "`dur`", or "`end`" is not specified, discrete media in a time container should be allowed to be shown if the region related with each of them has only one object of each, that means, "`fill=freeze`" is default even when unspecified.

Requirement
- Calculation of timeline is simple and lightweight with limited resources of memory and processor.

Suggestion
- Single time container.
- Single timing attribute value.
- Do not allow negative offset.
- Allow "`keypress`" events to control presentations.
- Allow displaying discrete media without specified timing or repeating if the region related with each of them has only one object.

16.3.5 Layout

Layout Module presents spatial layout related with objects in a screen. Presentation on a small display has some difficulty in rendering objects. Layout of objects may not be adjusted flexibly comparing their specified regions. So, the layout should be simple and effective.

Requirement
- Simple and effective presentation should be rendered in small area.

Suggestion
- Objects should fit their regions, or they may be rendered in no stretch, just simple upper-left hand rendering.
- "`root-layout`" may be full screen of the display.
- Nesting of layout regions should not be allowed.
- Objects could be moved by users in their positions and layers for better visibility.

@@ This last does not seem lightweight.

16.3.6 Media Object

Media Object Module presents description of media which constitute contents, such as text, image, audio and video.

Requirement
- Players should reduce network transactions to minimize delay time as much as possible.

Suggestion

- Media data may be packed in one format, such as MIME, MPEG4, and ASF, and may not need extra run-time transactions burdening bandwidth.
- Player may not care about fetching time.

16.3.7 Linking

Linking Module presents a hyperlink to relate objects with contents on a request by users. "`a`" and "`area`" links could be activated with focus and selection control and also with "`accesskey`". These linking elements might also be within single time container.

Requirement

- Need to minimize processing complexity for traversing a link.
- Should avoid complex timeline seeking behavior.

Suggestion

- Linking elements within single time container may be supported.

16.3.8 Structure

Structure module describes a structure of SMIL Basic documents.

Requirement

- Should be compatible with structure of full SMIL document.

Suggestion

- Elements of structure module should have, as minimal content model, elements defined in SMIL Basic.

16.4 Use of SMIL Basic Profile

SMIL Basic profile, can be used as is, is intended to be appropriate for small information appliances like mobile phones, and PDAs. Its simple design could serve as a baseline profile for extension for specific purpose. As for HTML+SMIL, another use of SMIL Basic may be to integrate SMIL Basic timing and media functionality with the HTML mobile profile.

17. Baseline Media Formats

Editor
Philipp Hoschka (ph@w3.org), (W3C)

17.1 Introduction

This Section defines a number of baseline media formats that the members of the W3C SYMM Working Group believe will be widely supported by SMIL players. Authors are encouraged to encode media objects in these formats to ensure that their SMIL documents can be played back by a wide range of SMIL implementations. Often, for audio and video formats, the baseline formats will be used as fallbacks when a player cannot render a more efficient, but less widely supported format. This can be achieved by using a `switch` element as shown in the following example:

```
<switch>
  <audio src="non-baseline-format-file" />
  <audio src="baseline-format-file" />
</switch>
```

Note that this Section is non-normative, and that thus implementation of the baseline formats is not a precondition for conformance to this specification.

For selecting the baseline formats, the following criteria were used:

- Implementing the format should not reqire a license fee OR
- The format should be implemented on a wide variety of platforms, indicating that licenses are available on reasonable and non-discriminatory terms. The platforms reviewed were the Linux operating system, the Java Media framework, Apple Quicktime, RealNetworks RealMedia and Microsoft Windows Mediaplayer multimedia platforms, and the Netscape and Internet Explorer browsers (trademarks are poperty of their respective owners).

17.2 Audio Formats

- audio/basic [MIME-2]
  @@ defined as follows: *The content of the "audio/basic" subtype is single channel audio encoded using 8bit ISDN mu-law [PCM] at a sample rate of 8000 Hz.* Note that this is probably a subset only of .au - need a spec for .au.

  Bandwidth: 64 Kbit/s

Editor's note: The Working Group encourages information on other audio formats that have lower bandwidth requirements than audio/basic and also do not require a license fee.

@@@ all the following "support" lists will not go in the final version

Support:

- Linux: yes (reference ?)
- Java Media: yes
- Quicktime: yes
- RealMedia: yes for .au (@@ confirm .au is a superset of audio/basic)
- MS Windows Media: yes for au (@@ but only if "local")
- Internet Explorer: yes
- Netscape: yes

@@ efforts to come up with license-free audio codecs

- preliminary effort to develop IP free audio compression format
- Ogg Vorbis CODEC project

17.3 Image Formats

- image/png ([PNG-MIME], [PNG-REC])
  Support:
  - Linux: yes
  - Java Media:no (but maybe in Java ?)
  - Quicktime: yes
  - RealMedia: yes (reference ?)
  - MS Windows Media: no
  - Internet Explorer: yes (reference ?)
  - Netscape: yes (reference ?)
- image/jpeg ([MIME-2], [JFIF])
  Support:
  - Linux: yes
  - JavaMedia: yes, supported in Java without Java Media Framework
  - Quicktime: yes
  - RealMedia: yes (baseline only, no support for progressive JPEG and grayscale)
  - MS Windows Media: no
  - Internet Explorer: yes
  - Netscape: yes

17.4 Video Formats

- video/mpeg [MIME-2]
  @@ MIME type definition is unclear, supposedly, this is MPEG-1 video - is there a spec for a file format for this ?

@@ are there licensing requirements for MPEG-1 ? The availability of Linux code would tend to suggest there aren't.

Support:

- Linux: yes
- Java Media: yes
- Quicktime: yes (Mac only, not on Windows)
- RealMedia: yes, via plugin
- MS Windows Media: yes
- Internet Explorer: @@
- Netscape: @@

- video/quicktime ([QT-MIME], [QT])

  @@ should restrict to one track and one codec, e.g motion JPEG or Cinepak

  Support:

  - Linux: yes (?? *Supported video is MJPA, JPEG Photo, PNG, RGB, YUV 4:2:2, and YUV 4:2:0 compression)*
  - Java Media: yes (Cinepak, H.261, H.263, Indeo (iv31 and iv32), JPEG (411, 422, 111), RGB)
  - Quicktime: yes
  - RealMedia: G2 on Windows supports CVID, Cinepak, and Indeo codec compression, provided those codecs are installed on the machine already.
  - MS Windows Media: yes, supports version 2 and lower (@@ check which codecs)
  - Internet Explorer: @@
  - Netscape: @@

Editor's note: The Working Group encourages information on other video formats that have lower bandwidth requirements that do not require a license fee.

17.5 Text Formats

- text/plain [MIME-2]

  For profiles supporting SMIL layout, players must choose a font size so that the constraints expressed by the `fit` attribute of the `region` element associated with the media object are fulfilled. If there is no associated `region` element, @@@ all bets are off, since text has no intrinsic size.

  @@ need to provide interpretation for "fit" attribute for objects without intrinsic size
  @@ suggestion to use CSS proposal for "copyfitting" (@@ pointer ?)

  @@ suggestion to use generic CSS font names, see http://www.w3.org/TR/REC-CSS2/fonts.html#generic-font-families

@@ suggestion to require CSS support for text/plain

- text/html @@ with appropriate parameter ? XHTML WG needs to decide about MIME type for XHTML profiles
  XHTML profile consisting only of the basic modules [XMOD] "Structure" (contains e.g. `html` and `body` element), "Basic Text" (contains e.g. `p` and `h1` element), "Hyptertext" (contains `a` element) and "List" (contains e.g. `ul` and `li` elements). The elements in this profile should be formatted using the "Mosaic-style" formatting described in CSS2 (@@ this means that the formatting is mostly fixed, and that there is no need to support CSS).

@@ Auxiliary links

- Supported file formats in Quicktime
- Supported content formats Windows media player
- RealNetworks authoring guide (contains list of supported formats)
- Supported media formats Java Media Framework 2.0 reference implementation

Appendix A. References

[CC/PP]
"CC/PP", technology of the W3C Mobile Interest Group. W3C Note 27 July 1999,
Available at http://www.w3.org/TR/NOTE-CCPP/

[CSS1]
"Cascading Style Sheets, level 1", Håkon Wium Lie, Bert Bos. W3C Recommendation 17 Dec 1996, revised 11 Jan 1999,
Available at http://www.w3.org/TR/REC-CSS1

[CSS2]
"Cascading Style Sheets, level 2", Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs. W3C Recommendation 12 May 1998,
Available at http://www.w3.org/TR/REC-CSS2

[CSS-selectors]
Cascading Style Sheets, level 2, Specification, chapter 5,
Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs. W3C Recommendation 12 May 1998,
Available at http://www.w3.org/TR/REC-CSS2/selector.html#q1

[DATETIME]
"Date and Time Formats", M. Wolf, C. Wicksteed. W3C Note 27 August 1998,
Available at: http://www.w3.org/TR/NOTE-datetime

[DC]
"Dublin Core Metadata Initiative", a Simple Content Description Model for Electronic Resources.
Available at http://purl.org/DC/

[DOM1]
"Document Object Model (DOM) Level 1 Specification", W3C Recommendation 1 October 1998,
Available at http://www.w3.org/TR/REC-DOM-Level-1

[DOM2Events]
"W3C Document Object Model Level 2 Events", T. Pixley, W3C Candidate Recommendation,
Available at http://www.w3.org/TR/DOM-Level-2/events.html

[DOM2]
"W3C (World Wide Web Consortium) Document Object Model (DOM) Level 2 Specification. W3C Candidate Recommendation 10 December, 1999
Available at http://www.w3.org/TR/DOM-Level-2

[draft-ietf-avt-rtp-mime-00]
"MIME Type Registration of RTP Payload Formats", Steve Casner and Philipp Hoschka, June 1999.

[Foley]
Computer Graphics: Principles and Practice, Second Edition by James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips, Addison-Wesley.

[HTML40]
"HTML 4.0 Specification" D. Raggett, A. Le Hors, I. Jacobs. W3C Recommendation 24 December 1999,
Available at http://www.w3.org/TR/REC-html40

[ISO8601]
"Data elements and interchange formats - Information interchange - Representation of dates and times", International Organization for Standardization, 1998.

[IETF]
"IETF - content negotiation working group".
Available at http://www.ietf.org/html.charters/conneg-charter.html

[JFIF]
"JPEG File Interchange Format, Version 1.02"; Eric Hamilton, September 1992.
Available at http://www.w3.org/Graphics/JPEG/jfif.txt

[MIME-2]
"RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types"; N. Freed, N. Borenstein, November 1996.
Available at ftp://ftp.isi.edu/in-notes/rfc2046.txt

[MOBILE-GUIDE]
"HTML4.0 Guidelines for Mobile Access", T. Kamada, Takuya Asada, Masayasu Ishikawa, Shin'ichi Matsui. W3C Note 15 March 1999
Available at http://www.w3.org/TR/NOTE-html40-mobile/.

[MODMOD]
"Building XHTML [tm] Modules" Murray Altheim, Shane McCarron. W3C Working Draft, work in progress,
Available at http://www.w3.org/TR/xhtml-building/

[NAMESPACES]
"Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman. W3C Recommendation 14 January 1999, ,
Available at http://www.w3.org/TR/REC-xml-names

[PICS]
"PICS 1.1 Label Distribution - Label Syntax and Communication Protocols", T. Krauskopf, J. Miller, P. Resnick and W. Trees. W3C Recommendation 31 October 1996,
Available at http://www.w3.org/TR/REC-PICS-labels

[PNG-MIME]
"Registration of new Media Type image/png"; Glenn Randers-Pehrson, Thomas Boutell, 27 July 1996.
Available at ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/image/png

[PNG-REC]
"PNG (Portable Network Graphics) Specification Version 1.0"; Thomas Boutell (Ed.).
Available at http://www.w3.org/TR/REC-png

[QT]
"QuickTime Movie File Format Specification", May 1996.
Available at

http://developer.apple.com/techpubs/quicktime/qtdevdocs/REF/refFileFormat96.htm
[QT-MIME]
"Registration of new MIME content-type/subtype"; Paul Lindner, 1993.
Available at
http://www.isi.edu/in-notes/iana/assignments/media-types/video/quicktime
[RDFsyntax]
"Resource Description Framework (RDF) Model and Syntax Specification", Ora Lassila and Ralph R. Swick. W3C Recommendation 22 February 1999,
Available at http://www.w3.org/TR/REC-rdf-syntax/
[RDFschema]
"Resource Description Framework (RDF) Schema Specification", Dan Brickley and R.V. Guha.W3C Proposed Recommendation 03 March 1999,
Available at http://www.w3.org/TR/PR-rdf-schema/
[RFC1766]
"Tags for the Identification of Languages", H. Alvestrand, March 1995.
Available at ftp://ftp.isi.edu/in-notes/rfc1766.txt
[RFC1889]
"RTP : A Transport Protocol for Real-Time Applications", Henning Schulzrinne, Steve Casner, Ron Frederick and Van Jacobson, January 1996.
Available at ftp://ftp.isi.edu/in-notes/rfc1889.txt
[RFC1890]
" RTP Profile for Audio and Video Conferences with Minimal Control" Henning Schulzrinne, January 1996.
Available at ftp://ftp.isi.edu/in-notes/rfc1890.txt
[RFC2326]
"Real Time Streaming Protocol (RTSP)" Henning Schulzrinne, Anup Rao and Rob Lanphier, April 1998.
Available at ftp://ftp.isi.edu/in-notes/rfc2326.txt
[RFC2327]
"SDP: Session Description Protocol" M. Handley and V. Jacobson, April 1998.
Available at ftp://ftp.isi.edu/in-notes/rfc2327.txt
[RUBY]
"Ruby Annotation", Michel Suignard, Masayasu Ishikawa, Martin Dürst. W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/ruby/
[SMIL10]
"Synchronized Multimedia Integration Language (SMIL) 1.0" P. Hoschka. W3C Recommendation 15 June 1998,
Available at http://www.w3.org/TR/REC-smil.
[SMIL-ANIMATION]
"SMIL Animation Module" Patrick Schmitz, Aaron Cohen and Philipp Hoschka. W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/smil-animation/
[SMIL-BOSTON]
"Synchronized Multimedia Integration Language (SMIL) Boston Specification". W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/smil-boston/

[SMIL-CSS2]
"Displaying SMIL Basic Layout with a CSS2 Rendering Engine". W3C Note 20 July 1998,
Available at: http://www.w3.org/TR/NOTE-CSS-smil.html

[SMIL-DOM]
"Synchronized Multimedia Integration Language Document Object Model (DOM)". W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/smil-boston-dom/

[SMIL-MOD]
"Synchronized Multimedia Modules based upon SMIL 1.0", Patrick Schmitz, Ted Wugofski and Warner ten Kate. W3C Note 23 February 1999,
Available at http://www.w3.org/TR/NOTE-SYMM-modules

[SMPTE]
"Transfer of Edit Decision Lists", ANSI/SMPTE 258M/1993

[SVG]
"Scalable Vector Graphics (SVG) 1.0 Specification". W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/SVG

[XHTML10]
"The Extensible HyperText Markup Language: A Reformulation of HTML 4.0 in XML 1.0". W3C Recommendation 26 January 2000,
Available at http://www.w3.org/TR/xhtml1/

[XHTML11]
"XHTML 1.1 - Module-based XHTML", Murray Altheim, Shane McCarron,.W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/xhtml11.

[XHTML-BASIC]
"XHTML, Basic", Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Toshihiko Yamakami. W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/xhtml-basic/

[XHTMLEvents]
"XHTML, Events Module" An updated events syntax for XML-based markup languages, Ted Wugofski,Patrick Schmitz,Shane P. McCarron
Available at http://www.w3.org/TR/xhtml-events/

[XHTML-PROF-REQ]
"XHTML Document Profile Requirements, Document profiles - a basis for interoperability guarantees", Dave Ragget, Peter Stark, Ted Wugofski. W3C Working Draft.
Available at http://www.w3.org/TR/xhtml-prof-req/.

[XINCL]
"XML Inclusion Proposal (XInclude)", Jonathan Marsh and David Orchard. W3C Note 23 November 1999,
Available at http://www.w3.org/TR/xinclude

[XLINK]
"XML Linking Language (XLink)", S. DeRose, D. Orchard and B. Trafford. W3C Working Draft, work in progress.

Available at http://www.w3.org/TR/xlink

[XML10]

"Extensible Markup Language (XML) 1.0" T. Bray, J. Paoli and C.M. Sperberg-McQueen. W3C Recommendation 10 February 1998 ,
Available at http://www.w3.org/TR/REC-xml

[XML-NS]

"Namespaces in XML" T. Bray, D. Hollander and A. Layman. W3C Recommendation 14 January 1999,
Available at http:/www.w3.org/TR/REC-xml-names

[XMOD]

"Modularization of XHTML", Shane McCarron, Murray Altheim, et al. W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/xhtml-modularization

[XPROF]

"Building Document Profiles", Work in Progress, HTML & CCPP Working Groups *(W3C Members only)*
Refer to http://www.w3.org/MarkUp/Group/ and
http://www.w3.org/Mobile/CCPP/Group

[XPTR]

"XML Pointer Language (XPointer)" Steve DeRose and Ron Daniel Jr. W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/WD-xptr

[XSL]

"Extensible Stylesheet Language (XSL) Specification", Stephen Deach. W3C Working Draft, work in progress.
Available at http://www.w3.org/TR/xsl/