Synchronized Multimedia Integration Language (SMIL) Boston Specification

W3C Working Draft 15 November 1999

This version:
http://www.w3.org/TR/1999/WD-smil-boston-19991115
Latest version:
http://www.w3.org/TR/smil-boston
Previous version:
http://www.w3.org/1999/08/WD-smil-boston-19990820
Editors:
Jeff Ayars (RealNetworks), Aaron Cohen (Intel), Ken Day (Macromedia), Erik Hodge (RealNetworks), Philipp Hoschka (W3C), Rob Lanphier (RealNetworks), Nabil Layaïda (INRIA), Jacco van Ossenbruggen (CWI), Lloyd Rutledge (CWI), Bridie Saccocio (RealNetworks), Patrick Schmitz (Microsoft), Warner ten Kate (Philips), Ted Wugofski (Gateway), Jin Yu (Compaq), Thierry Michel (W3C).

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 second 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.

Available formats

This SMIL Boston specification is also available in the following formats:

as well as a postscript file (single file):
    http://www.w3.org/TR/1999/WD-smil-boston-19991115/smilboston.ps
and a PDF file (single file):
    http://www.w3.org/TR/1999/WD-smil-boston-19991115/smilboston.pdf.

Quick Table of Contents

Full Table of Contents

Copyright Notice

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, with one section per module:

- Section 2 [p.15] presents an overview of the individual modules, and gives example profiles.
- Section 3 [p.21] defines the declarative animation module.
- Section 4 [p.23] presents the content control module, such as the switch and preload elements.
- Section 5 [p.35] describes the SMIL Boston basic layout module.
- Section 6 [p.43] defines the linking module.
- Section 7 [p.55] presents the media object module.
- Section 8 [p.65] defines the metadata module.
- Section 9 [p.69] defines the SMIL Boston structure module including the head, and body elements.
- Section 10 [p.73] defines the SMIL timing and synchronization module.
- Section 11 [p.135] describes the means of integrating SMIL timing into other XML-based languages.
- Section 12 [p.155] presents the transition effects module.
- Finally, Section 13 [p.171] defines the SMIL DOM interfaces for all of the above modules.

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:

- Jeff Ayars, RealNetworks
- Dick Bulterman, Oratrix (Invited Expert)
- Wayne Carr, Intel
- Wo Chang, NIST
- Aaron Cohen, Intel
- Ken Day, Macromedia
- Geoff Freed, WGBH
- Mark Hakkinen, Productivity Works
- Lynda Hardman, CWI
- Masayuki Hiyama, Glocomm
- Erik Hodge, RealNetworks
- Philipp Hoschka, W3C
- Eric Hyche, RealNetworks
- Jack Jansen, Oratrix (Invited Expert)
- Muriel Jourdan, INRIA
- Keisuke Kamimura, Glocomm
- Kenichi Kubota, Panasonic
- Nabil Layaïda, INRIA
- Rob Lanphier, RealNetworks
- Philippe Le Hégaret, W3C
- Pietro Marchisio, CSELT
- Thierry Michel, W3C
- Sjoerd Mullender, Oratrix (Invited Expert)
- Debbie Newman, Microsoft
- Jacco van Ossenbruggen, CWI
- Didier Pillet, France Telecom
- Hanan Rosenthal, Canon
- Lloyd Rutledge, CWI
- Bridie Saccocio, RealNetworks
- Patrick Schmitz, Microsoft
- Warner ten Kate, Philips
- Daniel Weber, Matsushita
- Gary Wiemann, National Security Agency
- Ted Wugofski, Gateway (Invited Expert)
- Jin Yu, Compaq

2. Synchronized Multimedia Integration Language (SMIL) Modules

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

2.1 Introduction

The first W3C Working Group on Synchronized Multimedia (SYMM) developed SMIL, the Synchronized Multimedia Integration Language [SMIL10]. This XML-based language [XML10] is used to express timing relationships among media elements such as audio and video files. SMIL 1.0 documents describe multimedia presentations that can be played in a SMIL-conformant viewer.

Since the publication of SMIL 1.0, 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 exploring how XHTML, the Extensible Markup Language [XHTML10], can be integrated with other languages. Both Working Groups are considering modularization as a strategy for integrating their respective functionality with each other and other XML languages.

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.

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 1.0 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 [p.15] :

- Animation Module [p.17]
- Content Control Module [p.17]
- Layout Module [p.17]
- Linking Module [p.17]
- Media Object Module [p.17]
- Metainformation Module [p.17]
- Structure Module [p.17]
- Timing and Synchronization Module [p.17]
- Transition Effects Module [p.18]

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

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.

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.

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.

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.

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.

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.

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.

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 properties including begin, dur, end, repeatCount, repeatDur, et al. These elements and attributes are subject to change.

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

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.

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

Animation	animate, set, animateMotion, animationColor	-	-
Content Control	switch	-	-
Layout	layout, region, root-layout	Stylesheet	style
Linking	a, area	Hypertext	a
		Link	link
		Base	base
		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
Structure	smil, head, body	Structure	html, head, body, title
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.

2.4 Multimedia Profiles

There are a range of possible profiles [p.15] 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 [p.19]
- SMIL 1.0 Profile [p.19]
- XHTML Presentations Profile [p.19]
- Web Enhanced Media Profile [p.20]

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:

- Animation Module
- Timing and Synchronization Module
- Transition Effects 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 SMIL language. This profile might include 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

2.4.3 XHTML Presentations Profile

The XHTML Presentations Profile integrates multimedia, XHTML layout, and CSS positioning. This profile might include the following SMIL modules:

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

This profile would use XHTML modules for structure and layout and SMIL modules for multimedia and timing. The linking functionality may come from the XHTML modules [XMOD] or from the SMIL modules.

2.4.4 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:

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

This profile is similar to the XHTML Presentations Profile with additional support to manage stream events and synchronization of the document’s clock to the primary media.

3. The SMIL-Boston Animation Module

Editors
Patrick Schmitz (pschmitz@microsoft.com), Microsoft
Aaron Cohen (aaron.m.cohen@intel.com), Intel

3.1 Introduction

This section defines the SMIL-Boston animation module. SMIL-Boston 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-Boston animation module when they need to include animation in their language.

The SMIL-Boston animation module will build upon the functionality of the SMIL Animation [SMIL-ANIMATION] module currently being prepared for last call. The timing model included in SMIL Animation is based upon SMIL 1.0 [SMIL10], with some changes and extensions to support interactive (event-based) timing. The extensions are compatible with a core subset of the functionality expected to be included in the SMIL-Boston timing module.

This approach was used in the first version of SMIL Animation in order to facilitate release of an animation module well before SMIL-Boston is ready to go to Recommendation status. The SMIL-Boston animation module will not include its own timing model and will simply use the SMIL-Boston timing module. It will not redefine timing markup specifically for the purpose of animation.

This version of the document will only describe the enhancements expected to be included in the final version of this module. Further definition will follow in a later draft.

3.2 SMIL Animation

Animation is inherently time-based. 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.

SMIL Animation presents a consistent model for document authors and runtime implementers, and introduces a framework for integrating animation with the SMIL timing model. Animation only manipulates attributes of the target elements, and so does not require any specific knowledge of the target element semantics. Host languages may build upon the support in SMIL Animation to define additional and/or more specialized animation elements.

3.3 SMIL-Boston Extensions to SMIL Animation

SMIL animation was written for integration into languages, such as SVG [SVG], that do not have other timing constructs. However, the need for declarative animation extends into languages (such as SMIL 1.0) which already have a great deal of support for timing. Animation in SMIL-Boston will overcome this limitation and extend SMIL Animation in a number of other significant ways.

SMIL-Boston animation will integrate fully with the SMIL-Boston language and and its more powerful timing module. Specific new functionality to be inherited from the SMIL-Boston timing module is expected to include:

- Support for multiple begin conditions and mixing event and scheduled timing on a single animation element.
- Enhanced DOM level 2 support including the generation of DOM timing events on animation elements.

Specific new functionality in the animation module is expected to include:

- A full framework for integration of animation with documents that contain other timing markup such as the elements `<par>`, `<seq>`, and `<excl>`, and attributes such as `begin` and `end` on media elements.
- Specialized DOM interfaces for determining the current state and value of an animation element.
- Removal of the restriction that the DOM `beginElement()` and `endElement()` methods can only be called on animation elements where `begin="indefinite"`.

4. The SMIL Content Control Module

Editors
Jeffrey Ayars (jeffa@real.com), RealNetworks
Dick Bulterman, (Dick.Bulterman@cwi.nl), 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 includes:

- 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 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 e.g. 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 Boston, 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.

SMIL Boston 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 SMIL Boston *camelCase* convention. For these, the deprecated SMIL 1.0 name is shown in parentheses after the SMIL-Boston 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 a 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, implementors 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 a smaller presentation.

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, #

audio channels, codec, HW mpeg decoder

Examples

1) Choosing between content with different bitrate

In a common scenario, implementations may wish to allow for selection via a "systemBitrate" parameter 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 bitrate

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 User Groups

The following is still under development by the SYMM Working Group. The syntax and semantics described here are preliminary and subject to change.

New to SMIL Boston is a mechanism for authors to define a set of test-attributes that enable a presentation to be customized to the needs of an individual viewer. The author defines a set of named states along with their initial value. In the body of the presentation these states are checked by declaring their ID’s to be the value of the "uGroup" test attribute. The user groups can be used to control content presentation or selection just like the system test attributes described above.

`<userAttributes>` element
: This element introduces a section within the SMIL head that contains definitions of each of the user groups.

Attributes:

uGroup
: an author-defined grouping of related media objects.

uState
: this is the default evaluated state of the uGroup.

If the uGroup attribute evaluates to true, the associated element is evaluated, otherwise it and its content is skipped. Note that players are free to implement different mechanisms for setting the state of the user groups. Bringing up a dialog box allowing the user to choose, and evaluating based on data stored in a configuration file, are two of the suggested alternatives.

Example:

```
<smil>
  <head>
    <layout>
      <!-- define projection regions a, b, c & d -->
    </layout>
    <userAttributes>
      <uGroup id="nl_aud" uState="RENDERED" title="Dutch Audio Cap" />
      <uGroup id="uk_aud" uState="NOT_RENDERED" title="English Audio Cap" />
```

```
      <uGroup id="nl_txt" uState="NOT_RENDERED" title="Dutch Text Cap"/>
      <uGroup id="uk_txt" uState="NOT_RENDERED" title="English Text Cap"/>
    </userAttributes>
  </head>
  <body>
    ...
    <par>
      <video src="announcer.rm" region="a"/>
      <text src="news_headline.html" region="b"/>
      <audio src="story_1_nl.rm" uGroup="nl_aud" region="c"/>
      <audio src="story_1_uk.rm" uGroup="uk_aud" region="d"/>
      <text src="story_1_nl.html" uGroup="nl_txt"/>
      <text src="story_1_uk.html" uGroup="uk_txt"/>
    </par>
    ...
  </body>
</smil>
```

{Need to provide description of example}

4.3 Presentation Priority/Grouping

The following is still under development by the SYMM Working Group. The syntax and semantics described here are preliminary and subject to change.

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 syntax and semantics described here are preliminary and subject to change.

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

The following is still under development by the SYMM Working Group. The syntax and semantics described here are preliminary and subject to change.

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 [p.55] , 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. The SMIL Layout Module

Editors
Aaron Cohen (aaron.m.cohen@intel.com), Intel

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 reuse the functionality in the SMIL layout module when they need to include media layout in their language. Conversely, language designers incorporating other SMIL modules do not need to include the layout module if other layout functionality is already present.

Changes with respect to the layout elements and attributes in SMIL 1.0 [SMIL10] are expected to be minor. SMIL 1.0 already provides for using alternative layout models, for example CSS [CSS2], and these can provide much of the additional functionality desired over the SMIL basic layout module. The support for hierarchical regions is one area in which the syntax may be extended.

5.2 Brief review of SMIL 1.0 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, a set of regions can be described using the SMIL 1.0 basic layout syntax, CSS2 syntax, or some other syntax [SMIL-CSS2].

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="15px" left="20px" width="100px" height="50px"/>

</layout>
```

To create the same region using CSS2 syntax:

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

[region="r"] { top: 15px; left: 20px; width: 100px; height:50px; }

</layout>
```

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

```
<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>

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 SMIL basic layout syntax and semantics

5.3.1 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. Typically the `<layout>` element appears in the document head section. If a document contains no `<layout>` element, the positioning of the body elements is implementation-dependent.

A SMIL document can contain multiple alternative layouts by enclosing several `<layout>` elements within a "switch" element. This can be used for example to describe the document's layout using different layout languages.

Element attributes

id
: This value uniquely identifies an 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 all content up until the next "</layout>" tag. The default value of the type attribute is "text/smil-basic-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 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>
      <region id="a" top="5" />
    </layout>
  </head>
  <body>
    <text region="a" src="text.html" dur="10s" />
  </body>
</smil>
```

Element attributes

The "region" element can have the following 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".
If the backgroundColor attribute is absent, the background is transparent.

background-color
: Deprecated. Equivalent to "backgroundColor", which replaces this attribute. Only SMIL document players are required to support "background-color".

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
- 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 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 to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2).

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 "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 have the same restrictions as the attribute values of the "height" attribute.
The default value is zero.

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".

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 have the same restrictions as the attribute values of the "height" attribute.
The default value is zero.

width

The use and definition of this attribute are identical to the "width" property in the CSS2 specification.
Attribute values have the same restrictions as the attribute values of the "height" attribute.

z-index

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

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

1. 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).
2. 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).

Element content

"region" 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 viewport, e.g. the window in which the SMIL presentation is rendered.

If a document contains more than one "root-layout" element, this is an error, and the document should not be displayed.

Element attributes

The "root-layout" element can have the following attributes:

backgroundColor
Defined in backgroundColor [p.37] under the `<region>` element.
background-color
Defined in background-color [p.37] under the `<region>` element.
height
Sets the height of the root element. Only length values are allowed.
id
Defined in id [p.38] under the `<region>` element.
skip-content
Defined in skip-content [p.38] under the `<region>` element.
title
Defined in title [p.39] under the `<region>` element.
width
Sets the width of the root element. Only length values are allowed.

Element content

"root-layout" is an empty element.

5.3.2 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.

SMIL basic layout only controls the layout of media object elements. It is illegal to use SMIL basic layout for other SMIL elements.

The type identifier for SMIL basic layout is "text/smil-basic-layout".

Fixed property values

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}
animation   {display: block;
             position: absolute}
body        {display: block}
head        {display: none}
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

SMIL basic layout defines default values for all layout-related attributes. These are consistent with the initial values of the corresponding properties in CSS2.

If the author wants to select the default layout values for *all* media object elements in a document, the document must contain an empty layout element of type "text/smil-basic-layout" such as:

```
<layout type="text/smil-basic-layout"></layout>
```

5.4 Extensions to SMIL 1.0 basic layout

The current SMIL 1.0 basic layout syntax is useful for its intended purpose as a simple, lightweight means of laying out a multimedia presentation that is available in all SMIL players. However, the need has arisen for additional layout functionality. It is expected that this additional functionality can be added at low implementation and runtime cost and not violate the original lightweight design goals of SMIL basic layout.

Additional functionality expected to be provided via new syntax includes:

- 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.

Additional functionality that may be provided via new syntax includes:

- Support for a hierarchical layout model; regions with parent/child relationships.

Additional functionality expected to be provided by support for additional CSS2/CSS3 properties includes:

- Alternative placement methods, e.g. bottom/right.
- Control of shape of cursor when over an anchor.
- Control of region transparency and opaqueness.

- Provide for control of acoustic rendering: audio levels, mixing, and placement in space.

5.5 Open Issues

Even given the straightforward extensions of the SMIL layout model discussed above, a number of open issues remain:

- Which CSS2/CSS3 properties do we support on regions? Do we define a set that all applications must support? Is support for other properties allowed?

  Ed: I think that we need to define a supported set of properties for visual and audio regions and make this set as complete as possible, supporting any attributes that have reasonable utility. If we do not define the list of properties, we run the risk of having layout be non-interoperable.
- Do we require support of the CSS syntax as well as the SMIL 1.0 syntax?

  Ed: This depends upon the cost of including a CSS parser in the runtimes. If the cost is high, the current strategy allows for a direct mapping between CSS and SMIL 1.0 layout syntax and requiring only the SMIL 1.0 is simpler. If the cost is low, it would be advantageous to align with other W3C efforts and require support for the CSS syntax.
- Do we deprecate the SMIL 1.0 syntax?

  Ed: This again depends upon the cost of including a CSS parser in the runtimes. If the cost is low, it would be advantageous to deprecate the SMIL 1.0 syntax and encourage people to use the more powerful CSS syntax. I don't think that we should disallow the SMIL 1.0 syntax altogether in this release.
- Do we extend the SMIL 1.0 syntax to include the newly supported CSS properties?

  Ed: If we do not deprecate or remove the SMIL 1.0 syntax, we should extend the attribute list to include the supported CSS properties.
- There is a strong dependency on the content control work to extend the use of test attributes both for accessibility and platform support.
- The transition module will likely depend upon a hierarchical layout model, and so we may need to include one here. We must do this in a way that is compatible with CSS2 layout.

6. The SMIL Linking Module

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

6.1 Introduction

The SMIL linking module defines the user-initiated hyperlink elements that can be used in a SMIL document. It describes:

- How other documents can use XPointer to link into SMIL documents
- How XPointer is used to reference document component referencing in certain attributes in SMIL

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.

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 will change when these two formats change.

6.2 XPointer Support

6.2.1 Linking into SMIL documents

SMIL 1.0 allowed authors to playing back a SMIL presentation at a particular element rather than at the beginning by using a URI with a fragment identifier, e.g. "doc#test", where "test" was the value of an element identifier in the SMIL document "doc". This meant that only elements with an "id" attribute could be the target of a link.

The SMIL Linking module defined in this specification allows using any element in a SMIL document as target of a link. In addition, we are considering supporting the use of XPointers for fragment identifiers in URIs pointing into SMIL documents. This would enable linking to elements that do not have a specified "id" attribute. For example:

The following URI selects the 4th par element of an element called "bar":

```
http://www.w3.org/foo.smil#id("bar").child(4,par)
```

Note that XPointer only allows navigating in the XML document tree, i.e. it does not actually understand the time structure of a SMIL document.

Error handling

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

When a link into a SMIL document contains an XPointer 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. The result of the link traversal is thus to play the "switch" element child that passes the usual switch child selection process.

6.2.2 Use of XPointer in SMIL attributes

The use of XPointer is not restricted to XLink attributes. Any attribute specifying a URI can use an XPointer (unless, of course, prohibited for that attribute’s document set).

XPointer can be used in various SMIL attributes which refer to XML components in the same SMIL document or in external XML documents. These include:

- Hyperlink attributes referring to link resource, through the "href" attribute, if the media object referenced in the link supports XPointer
- The media that get integrated into the presentation, through the "src" attribute, if the media object referenced in the link supports XPointer

6.3 Link Elements

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

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).

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 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 released in future versions of SMIL.

Linking to SMIL Fragments

A locator that points to a SMIL document may contain a fragment part (e.g. `http://www.w3.org/test.smi#par1`). The fragment part is an id value that identifies one of the elements within the referenced document. 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-Boston Timing 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. See the section on Error handling [p.44] .
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.

The <a> Element

The <a> element has the same syntax and semantics as the SMIL 1.0 <a> element. The <a> element has SMIL-only attributes as well. For synchronization purposes, the <a> element is transparent, i.e. it does not influence the synchronization of its child elements. <a> elements may not be nested. An <a> element must have an "href" attribute.

Attributes

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

id
: Standard XML ID attribute, for referential use.

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.
- "stop": When the link is traversed, the presentation containing the link stops, i.e. 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.

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": Semantics as defined in SMIL 1.0. In SMIL 1.0, the presentation containing the link continues playing when a link with show="new" is followed. As a refinement to SMIL 1.0, if both the presentation containing the link and the remote resource contain audio media, both are played in parallel.
  This specification allows authors to change the SMIL 1.0 behavior by setting the "sourcePlaystate" attribute to a value other than "play". To model the SMIL 1.0 behavior, the default value of "sourcePlaystate" is "play" when the "show" has the value "new".
- "pause": Semantics as defined in SMIL 1.0. Use of this value is deprecated in favor of the "sourcePlaystate" attribute.
  To achieve SMIL 1.0 behavior, the "show" attribute should be set to "new", and the "sourcePlaystate" attribute should be set to "pause".
- "replace": Semantics as defined in SMIL 1.0. In SMIL 1.0, the presentation containing the link pauses when a link with show="replace" is followed.
  This specification allows authors to change the SMIL 1.0 behavior by setting the "sourcePlaystate" attribute to a value other than "play". To model the SMIL 1.0 behavior, the default value of "sourcePlaystate" is "pause" when the "show" has the value "replace".

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

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 in which existing display environment the link should be opened (e.g. a SMIL region, an HTML frame or another named window), or triggers opening a new display environment. 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.

title
: This attribute provides human-readable text describing the link. It has the same significance as in SMIL 1.0.

Element Content

The "a" element can 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>
      ...
```

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 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 to spatial portions of a visual object, using the "href" attribute (in contrast, the <a> element only allows associating a link with a complete media object).
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 SMIL presentation.

The SMIL 1.0 <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:

- 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:

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

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

coords
: This attribute is extended to be identical with the "coords" attribute in HTML 4.0, i.e. 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.

  The value of the "fragment" attribute is a fragment. It can be a named anchor (for integrated HTML displays) an ID, an XPointer, or a locator in some other fragment scheme. For an XML file, it is the part that would come after the '#' in the URL. 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. For example, if the fragment is an XPointer, the media object must be an XML document.

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 (i.e. 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 (i.e., click on "link one" and 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.

Element Attributes

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 SMIL Boston 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 the old and the new version of a clipping attribute, the the attribute that occurs later in the text is ignored.

Example:

```
<audio src="radio.wav" clip-begin="5s" clipBegin="10s" />
```

The clip begins at second 5 of the audio, and not at second 10, since the "clipBegin" attribute is ignored. See Changes to SMIL 1.0 Attributes [p.62] 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 SMIL Boston 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 the old and the new version of a clipping attribute, the the attribute that occurs later in the text is ignored.

See Changes to SMIL 1.0 Attributes [p.62] for more discussion on this topic.

copyright
: The copyright notice of the content contained in the element.

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.

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.

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.

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 (defined below) 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.

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. The standard value for this field is "RTP/AVP", but alternate values may be defined by IANA. 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]. Only applies if media object is transferred via RTP.

@@ 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.

type

MIME type of the media object referenced by the "src" attribute.

xml:lang

Used to identify the natural or formal language for the element. For a complete description, see the XML specification, section 2.12 [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 [p.23] for details)

Element Content

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 below

7.3 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>
```

7.4 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.

7.4.1 Appendix A: Changes to SMIL 1.0 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.

7.4.2 Appendix B: Element Content

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

rtpmap
: Defined above

animate, set, animateColor, animateMotion
: Defined in Animation Module

7.4.3 Appendix C: New sections

The `rtpmap` element

A new section describing the "rtpmap" element is provided which 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.

7.4.4 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.

8.3 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.) 66

8.3.1 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 and activities 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, 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. 66

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: 66

```
<?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. The 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 term "profile family" needs definition. This can be done in Section B [p.??] . In this document a link to that definition is to be inserted.

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, Events	(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)

Events
: onBegin, onEnd

The Events collection is only defined when the Event Module (Section E) is selected in the profile.

@@ The enumeration of Events needs extension, e.g. with onRepeat, onClick, onMouseover, etc., or a reference to another specification is required.

@@ references to Section E need be checked in final version, plus an addition of a hyperlink to that section.

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

Schedule
: par, seq, excl [Section L]

MediaContent
: ref, audio, video, img, animation, text, and textstream [Section I]

MediaControl
: switch [Section D]

LinkAnchor
: a, area [Section H]

@@ references to other sections need be checked in final version, plus an addition of a hyperlink to those sections.

@@ Do we allow area as child of body?

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

The `body` element associates with the `root` in the Media Object Model [Section I]. Its default schedule semantics equals that of the `seq` element.

@@ references to Section I need be checked in final version, plus an addition of a hyperlink to that section.

@@ The following is a proposal

Whereas the <smil> element represents the document (DOM) root, the <body> element represents the schedule/media root. This has the consequence that upon nesting of SMIL documents the tree downwards from the <body> of the nested document is included in the schedule of the parent SMIL document. Navigation calls as performed by <area>, XPointer, and the XML-DOM and SMIL-DOM, smoothly extend along this tree.

9.3 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.

```
<!-- ============================================================= -->
<!-- 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*)?">
<!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;
>


<!-- 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*.

@@ Do we still need the next paragraph?

In the process of extending SMIL 1.0 for modularization and use in other XML languages, some alternate syntaxes have been defined. If a document would otherwise be SMIL 1.0 compatible except for use of alternate syntax, the use of the SMIL 1.0 syntax is recommended so the document will be playable by SMIL 1.0 as well as later document players.

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 "application/smil" documents (@@ or however we denote SMIL documents as distinct from integration documents).

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.

@@ Need a simple bar-illustration with labels for the simple and active durations and the frozen state.

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.

@@ The last sentence above is imprecise, but vague enough ("based upon") not to be incorrect. This section is not normative, and it is easier to gloss over details to make it flow.

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.

@@ ??Need a second bar-illustration that shows how a par cuts off children?

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 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 next paragraph and list should be moved to a "differences form SMIL 1" section

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

The complete syntax is described here, including syntax that is unchanged from SMIL 1.0.

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 like a mouse-click happens. The simple duration of an element is specified as a simple time value. The attribute syntax is described below.

@@ ?Need to introduce and better define the notion of syncbase and event base.

begin and dur Attributes

begin : smil-1.0-syncbase-value [p.81] * | begin-value-list [p.79] | "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 [p.81] * : "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 should not support this syntax.

begin-value-list [p.79] : begin-value (";" begin-value-list)?

A semi-colon separated list of begin values.

"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 [p.126] section.

Hyperlink-based timing is described in the Hyperlinks and Timing [p.117] section.

begin-value : (offset-value | syncbase-value | syncToPrev-value | event-value | media-marker-value | wallclock-sync-value)

Describes the element begin.

offset-value [p.81] : ("+" | "-")? clock-value [p.80]

Describes the element begin as an offset from the default syncbase. The definition of the default syncbase depends upon the element’s parent time container. The offset is measured in local time on the parent time container.

syncbase-value [p.82] : (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 [p.83] : ("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 element begin is defined relative to the begin or active end of the previous sibling element. This is equivalent to the default syncbase for children of a `<seq>` time container.

event-value [p.83] : (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 [p.115] .

media-marker-value [p.84] : id-ref ".marker(" marker-name ")"
Describes the element begin as a marker time in a media element.
wallclock-sync-value [p.85] : "wallclock(" wallclock-value ")"
Describes the element begin as a real-world clock time. The wallclock time is based upon syntax defined in [ISO8601].

dur
Specifies the simple duration.
The attribute value can be either of the following:
clock-value [p.80]
Specifies the length of the simple duration, measured in local time.
"indefinite"
Specifies the simple duration as indefinite.

If no `begin` is specified, the element begins at the time of the default syncbase. 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.

If there is any error in the argument value syntax for `dur` , the attribute will be ignored (as though it were not specified).

@@@ Need to Formalize rules for handling lists of times. Move some of this to Restarting elements. Basic rule is to take earliest resolved, valid time.
For begin, all are potentially valid. Earliest begins, and then others begin or not subject to restart rules.

@@@ Need to discuss, or at least mention how time container constraints interact with specified times. Note that the begin time is just a definition of sync, and especially w.r.t. negative begin times, will be be overridden by the time container constraints. Probably best to just mention it, and point to section within discussion of time containers.

@@@ Need to discuss (somewhere, perhaps not here but in a Details subsection) 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". Basic mechanism coming down from root is to subtract the begin of a parent, 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), apply filters for time manipulations (speed, acclerate/decelerate, autoReverse), and then recurse to the parent. Basic mechanism going up reverses all this. Note that the pure conversions do not take into account the clamping of active durations, nor the effects of fill (where time is frozen). An alternate form of the conversion is used when actually sampling the time graph; this clamps times when an element is frozen, to the value of the active end.

Implicit simple duration

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. 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, the implicit duration is typically a function of the media itself - e.g. video and audio files have a defined duration.
- For simple media elements that specify discrete media, the implicit duration is defined to be indefinite. Note the related example @@ example # [p.133] .
- 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 [p.107] .

@@ There are problems with making the implicit simple duration of discrete media be indefinite: we need to put in the qualification in seq that an element following an element with indefinite active duration becomes relative to the previous element **begin**. OTOH, we have to deal with this for other cases as well. But if we make the implicit duration be 0, then we have to add the note that the default for fill magically becomes "freeze". Neither approach is very clean. As an alternative, we could change seq to allow indefinite durations on children, e.g. for event-based end times. With the SMIL 1 semantics we cannot define the children of a sequence to just end on an event - they will all collapse as the default sync-bases get remapped to the previous begin.

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.

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 [p.115]).

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 in Id 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 [p.81] .
2. Else if the value begins with the token "prev", it should be parsed as a syncToPrev-value [p.83] .
3. Else if the value begins with the token "wallclock", it should be parsed as a wallclock-sync-value [p.85] .
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 [p.83] 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 [p.82] .
    3. Else if the token contains the string "`.marker(`", then the value should be parsed as a media-marker-value [p.84] .
    4. Else, the value should be parsed as an event-value [p.83] (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. Should we allow a leading backslash before any of these names to escape them, or simply reserve these names?
E.g. begin="\wallclock.click; foo.\marker+2s"

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

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). 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.

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 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 [p.79] ("+"|"-") S [p.79] 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 a clock value. The clock value specifies a presentation time offset from the time (i.e. the begin or active end) specified by the syncbase and time symbol. 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 [p.79] ("+"|"-") S [p.79] 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 [p.82] .

The *previous element* is the 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 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.

@@Should we also discard the offset? Consider trying to apply a uniform rule like "prev.begin+2s" to a series of elements. For the first one, we really want to make it show up immediately (usually). Would we ever want to preserve the offset in this case?

@@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 [p.79] ("+"|"-") S [p.79] 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 an "id" in the host language, for the event-base element. This element must be another timed element contained in the host document.

@@ Patrick: I think that the event base need not be timed - consider button clicks in HTML, clicks on regions in SMIL layout docs, etc.

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 clock-value that is a presentation time offset from the event. 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 [p.115] .

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 [p.79] marker-symbol S
[p.79] ")" )
```

@@Should we allow offsets from markers? Is this useful? Note that the offset would be in parent time, not media time.

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 [p.80] . 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 [p.109] 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 that linear continuous media. Not all continuous media types will support time manipulations, e.g. streaming MPEG 1 video playing backwards.

Support for timing manipulations is not required. Future versions of this module will specify fall-back behavior for user agents that do not support timing manipulations.

These are placed here, as they manipulate the simple duration - they are "inside" the definition of repeating, end, etc.

In the model for time manipulations, the element's local time can be filtered or modified. The filtered time affects all descendents. Any filter that changes the effective play speed of element time may conflict with the basic capabilities of some media players. The use of these filters is not recommended with linear media players, or with time containers that contain linear media elements.

The proposed extensions support use-cases commonly associated with graphic animation.

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.

speed Attribute

The speed attribute controls the local playspeed 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 modifies the playspeed 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 playspeed. Also note that a speed of -100% is equivalent to playing the media backwards.

Proposed syntax:
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. Zero values are not allowed.
> The default is "1.0" (no modification of speed).

accelerate and decelerate Attributes

@@ 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.

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 playspeed 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 acceleration and deceleration must not exceed 1. If it does, the deceleration value will be reduced to make the sum legal.

Proposed syntax:

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 speed or rate of progress through the simple duration must be increased to account for the acceleration 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>
```

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.

Proposed syntax:

autoReverse
: Controls autoReverse playback mode.

  Argument values are Booleans.
  The default value is false (i.e. play normally).

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. See also the sections Computing the Active Duration [p.95] and Propagating Changes to Times [p.119] .

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 [p.80]
  : 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, and both `repeatCount` or `repeatDur` are specified, the `repeatCount` will be ignored, and the `repeatDur`

will be used (refer to the examples below describing `repeatDur` and an indefinite simple duration). These rules are included in the section Computing the Active Duration [p.95] .

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 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="background.au" dur="3s" 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 defines the active duration). In the following example the simple duration is indefinite, and so the `repeatCount` is effectively ignored. Nevertheless, this is not considered an error. The active duration is also indefinite.

```
<img src="foo.png" dur="indefinite" repeatCount="2" />
```

In the following example, the simple duration is 0, and so repeat behavior is not meaningful. However, the `repeatDur` still determines the active duration. The effect is 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="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 [p.81] or a smil-1.0-syncbase-value [p.81] , 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".

Controlling Active Duration

SMIL Boston provides an additional control over the active duration. The `end` attribute allows the author to specify the active end of the element using a simple offset, a syncbase, an event-base or DOM methods calls.

end Attribute

end : smil-1.0-syncbase-value [p.81] * | end-value-list [p.79] | "indefinite"

Defines the active end of the element (i.e. the end of 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 [p.81] * : "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 should not support this syntax.
end-value-list [p.79] : end-value (";" end-value-list)?
A semi-colon separated list of end values.
"indefinite"
The begin of the element will be determined by a "endElement()" method call.
The SMIL Timing and Synchronization DOM methods are described in the Supported Methods [p.126] section.

end-value : (clock-value | syncbase-value | syncToPrev-value | event-value | media-marker-value | wallclock-sync-value)
Describes the element begin.

clock-value [p.80]
Describes the active end as an offset from the default timebase. The offset is measured in local time on the parent time container.
syncbase-value [p.82] : (id-ref "." ("begin" | "end")) (("+" | "-") clock-value)?
Describes a syncbase and an offset from that syncbase. The active end is defined relative to the begin or active end of another element.
syncToPrev-value [p.83] : ("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 active end is defined relative to the begin or active end of the previous sibling element.
event-value [p.83] : (id-ref ".")? (event-ref) (("+" | "-") clock-value)?
Describes an event and an optional offset that determine the active end. 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 [p.115] .
media-marker-value [p.84] : id-ref ".marker(" marker-name ")"
Describes the active end as a marker time in a media element.
wallclock-sync-value [p.85] : "wallclock(" wallclock-value ")"
Describes the active end 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 active duration is indefinite (until resolved).

The active duration is defined by the simple duration, the repeat attributes and end. The rules for combining these are presented in the section Computing the Active Duration [p.95] .

@@@ Need to Formalize rules for handling lists of end times.
Basic rule is to take earliest resolved, valid time, that is not in the past.

@@Need example with end cutting short a longer simpler dur might also be good.

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 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 video when the user clicks on the element.

```
<video src="video.mpg" dur="indefinite" 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 some element "next" that indicates to the user that the next step should be shown.

```
<par ... >

  <video id="step1" begin="0" dur="5s"

     repeatCount="3" end="nextBtn.click" src.../>

  <video id="step2" begin="step1.end" dur="5s"

     repeatCount="3" end="nextBtn.click" src.../>

  <video id="step3" begin="step2.end" dur="5s"

     repeatCount="3" end="nextBtn.click" src.../>

  <video id="step4" begin="step3.end" dur="5s"

     repeatCount="3" end="nextBtn.click" src.../>
```

```
  <video id="step5" begin="step4.end" dur="5s"
     repeatCount="3" end="nextBtn.click" src.../>
</par>
```

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

The table in Figure @@@ shows the semantics of all possible combinations of simple duration, `repeatCount` and `repeatDur`, and `end`.

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 [p.108] .

In the table below, if a cell is empty, it indicates that the associated attribute is omitted in the syntax. Where the table entry is "defined", this should be interpreted as a specified or implicit value other than "indefinite". Where the entry is a star ("*"), the value does not matter and can be any of the possibilities. Note that where the active duration is specified as the minimum of several values (MIN), it may not always be possible to calculate this when the document begins. If the `end` is event-based or DOM-based, then an event or method call that activates `end` *before* the duration specified by `repeatCount` or `repeatDur` will cut short the active duration at the `end` activation time.

Note that if either the simple duration or the value of `end` cannot be resolved, the respective value is considered to be "indefinite" for the purposes of evaluating the active duration. If and when the respective value becomes resolved, the active duration is reevaluated.

Note that for any active duration and simple duration that are both not indefinite, the number of repeat iterations is defined by the active duration divided by the simple duration (this may yield partial repeat iterations, just as `repeatCount` specifies).

If both `end` and either (or both) of `repeatCount` or `repeatDur` are specified, the active duration is defined by the minimum duration defined by the respective attributes.

If `end` is specified but neither of `repeatCount` or `repeatDur` are specified, then the active duration is defined as the minimum of the simple duration and the duration defined by `end`.

Note also that it is possible to have an indefinite simple duration and a defined, finite active duration. The active duration can cut short the simple duration, but the active duration does not define the simple duration, or change its value.

The following symbols are used in the table as a shorthand:

B
: The begin of an element.

d
: The simple duration of an element.

Simple duration **d**	`repeatCount`	`repeatDur`	`end`	Active Duration
defined				**d**
defined	defined			`repeatCount`*__d__
defined		defined		`repeatDur`
defined			defined	MIN(__d__, `end`-__B__)
defined	defined	defined		MIN(`repeatCount`*__d__, `repeatDur`)
defined	defined		defined	MIN(`repeatCount`*__d__, (`end`-__B__))
defined		defined	defined	MIN(`repeatDur`, (`end`-__B__))
defined	defined	defined	defined	MIN(`repeatCount`*__d__, `repeatDur`, (`end`-__B__))
indefinite	*			indefinite
indefinite	*	defined		`repeatDur`
indefinite	*		defined	`end`-__B__
indefinite	*	defined	defined	MIN(`repeatDur`, (`end`-__B__))
*	indefinite			indefinite
*		indefinite		indefinite
*	indefinite	indefinite		indefinite
*	indefinite		defined	`end`-__B__
*		indefinite	defined	`end`-__B__
*	indefinite	indefinite	defined	`end`-__B__

Fig. @@@: 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 one extension:

fill Attribute

fill : ("remove" | "freeze" | "hold")
This attribute can have the following values:

remove
Specifies that the element will not extend past the end of the active duration.
This is the default value.

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).

This attribute only has an effect on visual media elements. Non-visual media elements (audio) should ignore this.

Note that `<area>` hyperlinks defined as children of a media element are still active during fill="freeze". See also the SMIL 1.0 specification [SMIL10].

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 [p.102] .

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 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.

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 Attribute

restart
: controls under which circumstances, if any, an element is restarted

  always
  : The element can be restarted at any time.
    This is the default value.

  never
  : The element cannot be restarted for the remainder of the current simple duration of the parent time container.

  whenNotActive
  : The element can only be restarted when it is not active (i.e. after the active end). Attempts to restart the element during its active duration are ignored.

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 would likely 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 investigating defining a method to override the default and set a new default for the document.

Possibilities under consideration are a <meta> element in the head and an attribute on the <body> element. Language designers could specify other mechanisms for languages that do not have a <body> element.

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 will also restart (subject to the value of `restart` on these elements).
- An element with `begin` specified as "indefinite" can be restarted when the DOM "beginElement()" method is called repeatedly.
- An element with `begin` specified as "indefinite" can be restarted by repeatedly actuating a hyperlink that has the element as its target.

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.

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.

The restart setting for an element is evaluated not when the syncbase element restarts or when the eventbase event happens, but rather when the element would actually restart (i.e. taking into account any offset from the syncbase or eventbase). For example:

```
<img id="go_btn" fill="freeze" .../>

<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. The effective restart of "bar" is scheduled for 2 seconds later, at 8 seconds. However, since "bar" will begin playing at 7 seconds, the restart at 8 seconds will be ignored (the `restart` attribute is set to only allow restart when "bar" is not active). Element "bar" plays from 7 seconds until 17 seconds. If the user again clicks the image at 16 seconds, "foo" will restart. Although "bar" is active when the user clicks the image, the restart for "bar" is not scheduled until 2 seconds later at 18 seconds. At this point, "bar" will have ended the first time, and so can legally restart.

The same rules apply if an element is defined to begin when another ends, as in this example:

```
<img id="go_btn" fill="freeze" .../>

<video id="foo" begin="go_btn.click" dur="2s" .../>

<audio id="bar" begin="foo.end" dur="10s"

    restart="whenNotActive" ..." />
```

If the user clicks on the "go_btn" image once at 5 seconds, "foo" will play from 5 to 7 seconds, and "bar" will play from 7 to 17 seconds. If the user clicks again at 16 seconds, "foo" will restart and play again from 16 to 18 seconds. At 18 seconds, "bar" will have ended, and so will restart and play from 18 to 28 seconds.

@@ this seems useful to point out but a SMIL element cannot be visible before it begins so having a begin="click" means it won’t ever begin.

@@Patrick: except that the begin may not mean make it visible - e.g. with timeAction, it could mean apply a class or style, and so begin=click is reasonable.

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 [p.115] .

10.3.3 Time Containers

SMIL Boston specifies three time containers: <par>, <seq>, and <excl>.

The par time container

<par> [p.130]

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> [p.131]
: 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 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 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). Child elements *may* define an `end`.

Thus 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.

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. Paused elements are resumed when the interrupting element ends its active duration.

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 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>`. 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 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

@@ Note that the specifics of the pause functionality and how it is represented in syntax are still under discussion, and may well change.

As in the audio descriptions use case above, it is possible to specify whether a child of the <excl> should pause or stop the clip that it is interrupting. This is done be specifying the whenDone attribute on the children of the <excl>. The default behavior of whenDone is to stop the currently playing clip. The whenDone attribute only applies to children of an <excl>. Children of an <excl> that have been paused are placed on a *stack* and playback is resumed for elements in the stack in last-in, first-out order.

whenDone

Defines the pausing behavior for children of <excl> time containers.
This is only defined for children of <excl>.
Legal values for the attribute are:

stop
: The previously playing element (if any) will be stopped when this element begins. The active duration of the stopped element is defined to end at this point, and the element will not be resumed.
This is the default.

resume
: The previously playing element (if any) will be paused when this element begins. The paused element is placed upon a last-in, first out stack of paused elements. When the newly started element ends its active duration, the pause element will be removed from the stack, and resumed where it was paused.

@@Need to define how recursion stops: if an element is begun that is on the stack, the stack is popped all the way down to (and including) the newly begun element. This may seem harsh, but it cleanly prevents odd recursion on the stack.

@@Need to resolve whether we want to use whenDone on the interruptor, or interrupt attribute on the interruptee, or some other variant.

In the following example, "default_audio.ra" begins playing when the <excl> begins. If the user clicks on the intro button, "intro_audio.ra" begins and "default_audio.ra" is paused because the intro audio's whenDone attribute is set to "resume". If the user then selects the moreinfo button, "intro_audio.ra" is paused and "moreinfo_audio.ra" begins. When the active duration of "moreinfo_audio.ra" ends, "intro_audio.ra" resumes from where it was paused. When the active duration of "intro_audio.ra" ends,

"default_audio.ra" resumes from where it was paused. When the active duration of "default_audio.ra" ends, the simple duration of the <excl> ends.

```
<excl>

   <audio src="default_audio.ra" begin="0" .../>

   <audio src="intro_audio.ra" begin="intro.click"

          whenDone="resume" .../>

   <audio src="moreinfo_audio.ra" begin="moreinfo.click"

          whenDone="resume" .../>

</excl>
```

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 value of the whenDone attribute.

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, and so the last of these elements in document order will end up being played. The same semantics is applied if multiple children of the <excl> specify the same begin event-base value. Nevertheless, if the whenDone attribute of children sharing begin times or events is set to "resume", the elements that occur earlier in the document will be paused. In this case, the children will play back in the reverse of the document order, as the pause stack "unwinds".

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.

Seeking

Children of the `<excl>` can be activated by hyperlinks or by events. When using events, the `<excl>` time container must be active for child elements of the `<excl>` to be activated by an event. When using hyperlinks, if the `<excl>` is not active, a seek will occur to the begin time of the `<excl>` and the children of the element will be activated. If the `<excl>` is currently active when the hyperlink is selected, a seek does not occur and playback of the child element begins. Playback of other active elements outside the scope of the `<excl>` is unaffected.

When seeking to a child of the `<excl>`, normal activation rules apply. If activation of a child of the `<excl>` requires advancing the timeline forward, the document should be in a state identical to that as if the document presentation time advanced undisturbed to reach the seek time.

endSync

endSync is only valid for the `<par>` and `<excl>` time containers. It 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.
Legal values for the attribute are:

first
: The `<par>` or `<excl>` 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.

last
: The simple duration of the time container 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 with a scheduled or resolved begin.
This is the default value.

all
: The `<par>` or `<excl>` 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>` or `<excl>` 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.
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. Elements do not have to play to completion, but must have played at least once. When no elements are active, and when all elements have run one or more times, then the parent can end.
@@ we have an issue with all: is it more like "first" in that once everything has played, we can stop, and possibly cut off other elements, or is it more like "last", in that once everything has played, we wait for the last thing to end so nothing gets cut off?
- `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.

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 the last 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.

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.

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 semantics of the `whenDone` attribute, 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

Probably need to define "active" and "inactive" more formally.

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*. If the defined sync would place the element effective 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.

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. This may require a reset of some sort in the implementation to ensure that the sync relationship to the parent time container is recalculated.

Specifics for event-based timing

The parent time container must be active for the child element to receive events.

Sequence children are only sensitive to events between the active end of the previous element and the active begin of the following element. Example: all children listen to same begin event and it works:

```
<seq>
   <img src="img1.jpg" begin="foo.click" dur="5s" fill="freeze"/>
   <img src="img2.jpg" begin="foo.click" dur="5s" fill="freeze"/>
   <img src="img3.jpg" begin="foo.click" dur="5s" fill="freeze"/>
</seq>
```

Negative Begin delays

A negative begin (delay) value defines a clipBegin for the first -- and only the first -- iteration of a repeated element. Without specifying a repeat attribute, a negative begin value and clipBegin are synonymous.

10.3.4 State Transition Model

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 @@ shows the legal transitions between the states of an element:

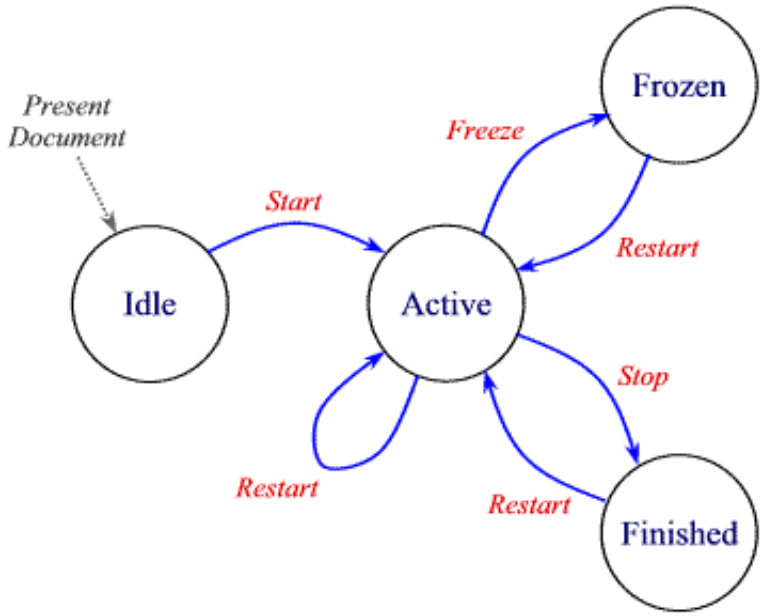


Fig @@@: State diagram of an element

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.

@@ We need to note that when the parent (or any ascendant) time container repeats or restarts, the element implicitly is reset to the "idle" state.

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.

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()` method call, or if the user activates (traverses) a hyperlink that is targeted at the element.

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 [p.75] .

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 [p.95] .

Freeze Transition: Active to Frozen

If a timed element has the `fill` attribute set to "freeze" or "hold", 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 parent time container as described in Time Container constraints on child durations [p.108] .

If the `fill` attribute has the value "hold", the *frozen* state lasts until the end of the *current simple duration* of the parent time container. The *current simple duration* refers to one iteration with a repeating time container, or the (single) instance of the simple duration for a time container that does not repeat. If the active duration of the parent time container cuts short the simple duration (e.g. for `repeatCount="2.5"`), the *frozen* state ends when the active duration of the parent time container ends.

If the `fill` attribute has the value "freeze", the *frozen* state depends upon the parent time container of the current element:

- Within a `<par>` time container, the *frozen* state is the same for both "freeze" and "hold" values of the `fill` attribute.
- Within a `<seq>` or `<excl>` time container, `fill="freeze"` specifies that the frozen state lasts until the next element in the time container begins, or until the end of the *current simple duration* of the parent time container (which ever comes first). See also the section Freezing elements [p.97] .

It should be noted that for a child of `<excl>`, the frozen state does not apply to an element that has been paused, but only to elements that have ended.

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.

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

This state transition can be controlled with the `restart` attribute. If the `restart` attribute value is "always" or "whenNotActive" the element will transition to the active state in response to an event specified in `begin`, a DOM `beginElement()` method call or a hyperlink activation. The restart transition effectively resets the state of the element; the element's simple and active duration must be recomputed as if it were being started for the first time.

Restart Transition: Active to Active

An element may receive an event specified in `begin`, a `beginElement()` DOM call or be the target of a hyperlink activation 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: Finished to Active

An element restart can result from an event specified in `begin`, a DOM call or a hyperlink activation, 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`. A `restart` value of `never` will prevent this transition from occurring.

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( activeTime, d )
```

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. This 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 [p.83] 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()` method). 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. For event-based begin times, the element becomes active (begins to play) at the time that the event is raised. The element plays from the beginning of the media (subject to any explicit clipBegin). For event-based active end times, the element becomes inactive (stops playing) when the associated event is raised.

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" beginEvent="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 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.
    1. 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:
    1. If `restart="always"`, then the element restarts and any specification of the event in `end` is ignored for this event instance.
    2. If `restart="never"` or `restart="whenNotActive"`, then any `begin` specification of the event is ignored for this instance of the event. If `end` specifies the event, the element active duration ends.

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 [p.101] .

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 next section.

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 a timed 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 affect the presentation in one of three ways, depending upon the value of the element's `begin` attribute and the current *resolution status* of the element's begin time:

- For elements where the value of the `begin` attribute is equal to "indefinite", the element will simply be started when a link targeted at the element is traversed. In this case the semantics of the `restart` attribute will govern the behavior of the element if subsequent hyperlink traversals occur.
- Elements where the value of the `begin` attribute is not equal to "indefinite", and the begin time is currently resolved (i.e. for the current simple duration of the parent time container) will seek the presentation to that resolved time and continue normally from that point. In this case the semantics of the `restart` attribute will not apply to the activation of the hyperlink.
- Elements where the value of the `begin` attribute is not equal to "indefinite", and the begin time is unresolved will not seek the document timeline and will have no effect upon the referenced element. This should not be considered an error, and should not affect other language-defined link behavior.

@@ Need to describe seeking parents as needed to activate the target.

@@ Should we deal with activating targets that are not marked "indefinite"?

@@ Need to add qualifications on a resolved begin that is outside the current simple duration of the parent time container.

@@ Need to describe that seek should prefer the current simple duration, where it makes sense.

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 cause any other elements that have resolved begin times between the current time and the seeked-to time to begin. These elements may have ended, or may still be active or frozen at the seeked-to time, depending upon their begin times and active durations. Also any elements currently active at the time of hyperlinking should "fast-forward" over the seek interval. These elements may end the active duration, may be still active or may be frozen once the seek is complete, depending upon their active durations. 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. 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.

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.

Implications of beginElement() and hyperlinking for seq time container

@@This is rough

For a child of a sequence time container, if `beginElement()` is called or 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 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.

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.

@@Need to resolve which events get raised. I think we should raise an end event only for the currently playing element, and a begin for the referenced element. Why? Because if an element will never actually play, script need not be notified, and we can preclude a flood of events that get collected during a seek operation.

@@ Compare this discussion to epsilon model.

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 "bar" is notified. However, as "bar" 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.

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.
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.

@@ Need to make clear that syncBehavior does not inherit, but does default to the inherited value for defaultSyncBehavior. Need some intro the the default*** attributes

@@ Need (somewhere) to specify that once event-based timing is resolved, the element is tied to the parent time container for the purposes of runtime sync management. Similar issue for long sync arcs

@@Should we say the following, or leave it to implementations? There may be a non-zero default value for this attribute, to be determined through testing (a proposed guess is something between 0.1 seconds and 0.5 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.

defaultSyncBehavior
: Defines the default value for the runtime synchronization behavior for an element, and all descendents.
Legal values are:
canSlip
: Allows the associated node to slip with respect to the parent time container.
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 [p.80] .

defaultSyncTolerance

Defines the default value for the synchronization tolerance for an element, and all descendents.
Legal values are Clock-values [p.80] .

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 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.

10.4.1 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.

10.4.2 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.

10.4.3 Supported Methods

@@We may support other methods in SMIL Boston, in addition to these.

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" .../>
```

@@Need to do more work on rules for beginElement, and interaction with time container constraints. Should beginElement work like a hyperlink activation, or like an event, or like a long sync-arc, or should it just make the element runnable when the parent begins?.

Calling `beginElement()` causes the element to begin in the same way that an element with event-based begin timing begins (@@ This is still under discussion, and may well change). 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 [p.99] .

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 [p.97] . 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.

10.5 Glossary

10.5.1 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].

Timebases

In scheduled timing, elements are timed relative to other elements. The timebase 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 timebase 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 timebase. 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 [SMIL-MEDIA]. 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.

10.5.2 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 [p.127] , 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 [p.115] . 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.

10.6 Appendix A: Annotated Examples

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.

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.

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 beginEvent="btn1.click">
        ...
      </par>
      <par beginEvent="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.

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.

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.

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.

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 [p.139] 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 [p.136] .
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 [p.141] 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 [p.139] . 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 [p.141] . 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 [p.141] . 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 [p.141] or Timesheets [p.141] 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 [p.73] of this working draft.

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 [p.145] 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 [p.146] 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 [p.73] 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 [p.73] , 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 [p.141] and declared [p.141] time container elements may contain any legal timing attributes defined for media elements as specified in the SMIL Timing module [p.73] .

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` must be specified by the host language. These values could include: "display" (a reasonable default value) to make an element appear and disappear on its parent's time line, "visibility" to make an element appear and disappear visibly without affecting its presentation space, and "class" to apply the associated timing to the element's style class.

Examples:

Note: the In-line Timing Framework [p.139] section contains several examples [p.140] using SMIL timing [p.73] .

11.4.2 Specification: Future Specifications Under Consideration

Future Specification: CSS Timing

See Appendix D: Future Specification: CSS Timing [p.150] .

Future Specification: Timesheets

See Appendix E: Future Specification: Timesheets [p.150] .

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. 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, and to halt playback of the document if it had begun before the erroneous code was received by the parser.

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 working draft 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 [p.141] 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 [p.139] section, above, will be needed. "timeContainer" [p.139] is needed to be able to declare that an element should act as a time container. The "animate" element and/or the "timeAction" [p.139] 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 [p.73] . 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 [p.148] .
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 [p.141] ; 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 [p.145] , 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 working draft.
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 [p.136] . 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 [p.141] section contains an example [p.145] using SMIL timing [p.73] .

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 [p.140] .

11.10 Appendix E. Future Specification: Timesheets

Timesheet syntax has not been specified, but several possibilities are under consideration. The Timesheets Framework [p.141] section contains several examples (1 [p.147] , 2 [p.148]) using SMIL timing [p.73] .

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 but noticeably absent from SMIL is 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 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 which 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 lanauge 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 [p.157] 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 [p.163] 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 [p.157] section, we define a taxonomy of transitions and describe the families of transitions. Next in the Parameters [p.160] 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 [p.163] section, we describe the semantics of applying a transition class to a media element. Next, in the Multiple-Element Transitions [p.165] 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.

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" [default], "fromRight", "fromBottom", "fromLeft"
slideWipe	"fromTop" [default], "fromRight", "fromBottom", "fromLeft", "angular"
fade	"crossfade" [default], "fadeToColor", "fadeFromColor"

warp	"explode" [default], "implode", "zoomOver", "zoomBoth"

For each of the types, the first subtype is labelled 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 would be 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.

Note that in the following parameter list, we do not call these parameters "attributes" or "properties". If the transition class is expressed in SMIL, then each of these parameter names become *attributes* of a "transition" element in the <head> tag. The attribute names can be exactly the same as the parameter names below.

However, if the transition class is expressed in 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 define the following transition parameters:

type
: This is the type or family of transition. The "type" parameter is required and must be one of the transition families listed in the Taxonomy [p.157] 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 [p.157] 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. See Section 4.2.4 of the SMIL 1.0 Recommendation for a definition of its semantics. This parameter is *required*.

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.how many times to repeat the wipe pattern along the vertical axis.

  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.how many times to repeat the wipe pattern along the vertical axis.

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.how many times to repeat the wipe pattern along the vertical axis.

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 ths 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".

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 CSS 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.

12.4 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, we simply add a "CLASS" atttribute and with the value being the transition class defined in the class selector of the transition definition. For SMIL, we propose adding a "transition" attribute to all media object elements. The value of the "transition" attribute would correspond to value of the "id" attribute of one of the <transition> elements defined in the <head> of the document. For instance, 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 (see section 4.2.4 of the SMIL 1.0 Recommendation for a more detailed explanation of effective end). 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>
...
```

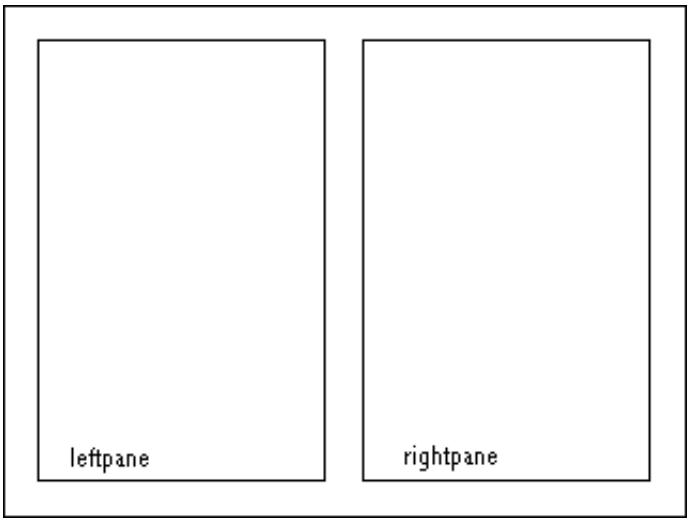
crossfades both at the beginning and end of foo2.mpg are between active frames of both foo1.mpg and foo2.mpg.

12.5 Multi-Element Transitions

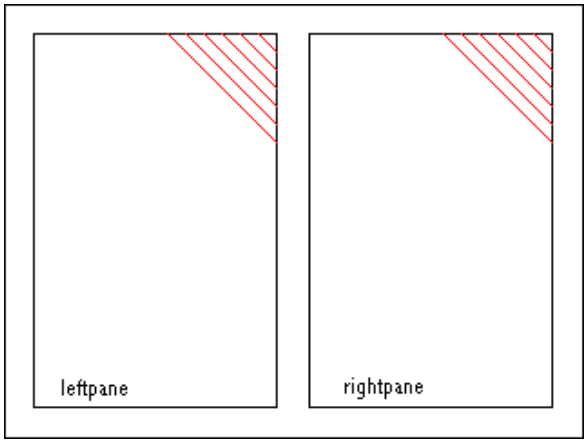
Up until this point in the discussion, we have applied transitions to single media object elements. However, it is common practice to apply transitions across several different media at once. Consider the following example:

```
<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"  region="whole"/>
                <img src="left1.jpg"  region="leftpane"/>
                <img src="right1.jpg" region="rightpane"/>
            </par>
            <par>
                <img src="back2.jpg"  region="whole"/>
                <img src="left2.jpg"  region="leftpane"/>
                <img src="right2.jpg" region="rightpane"/>
            </par>
        </seq>
    </body>
</smil>
```

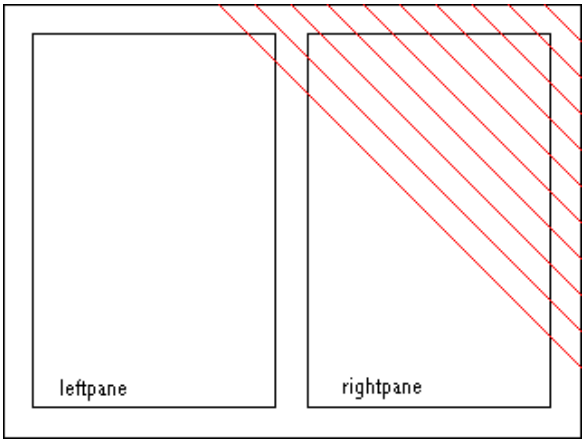
where the regions of this presentation look like:



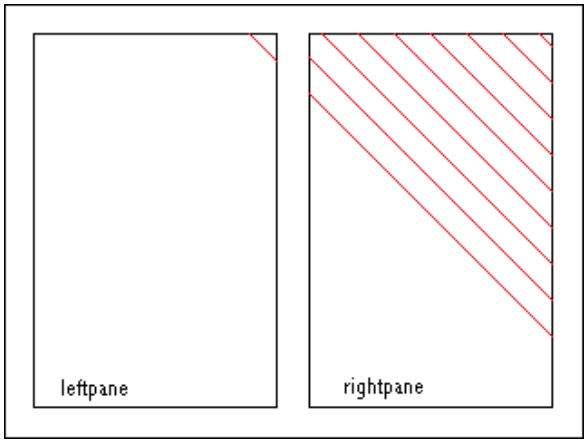
Suppose that we had defined a transition class called "diagwipe" to be a 1-second diagonal wipe from upper right to lower left. In this example, we consider 4 possible different cases of how we might want to apply this transition to this presentation:



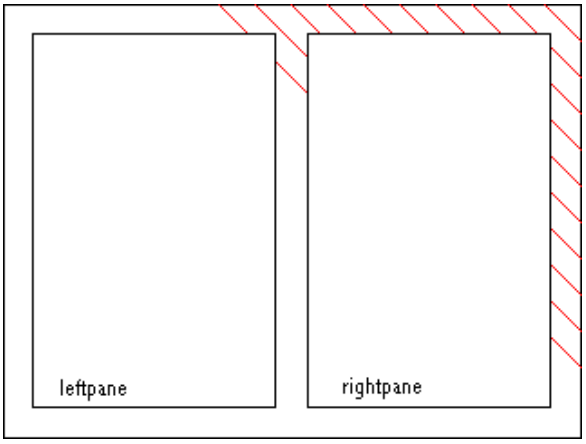
Case 1



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"  region="whole"/>
                <img src="left1.jpg"  region="leftpane"/>
```

```
            <img src="right1.jpg" region="rightpane"/>
        </par>
        <par>
            <img src="back2.jpg"  region="whole"/>
            <img src="left2.jpg"  region="leftpane"  transition="diagwipe"/>
            <img src="right2.jpg" 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"  region="whole"/>
                <img src="left1.jpg"  region="leftpane"/>
                <img src="right1.jpg" region="rightpane"/>
            </par>
            <par>
                <img src="back2.jpg"  region="whole" transition="diagwipe"/>
                <img src="left2.jpg"  region="leftpane"/>
                <img src="right2.jpg" 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? In SMIL, we will now allow the <region> element to 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 for the purpose of

placing lightweight media (such as solid fills, pattern fills, and transitions) on the timeline. Since *everybody* usually has an opinion about element names, I will not even bother trying to name this element. The name "fill" has been suggested, but this seems to imply that it is always doing fills of some sort, which is not solely the intention. Therefore, for now, I will call this element "phil". This element can be used to place solid or pattern fills in regions at points along the timeline. Also, 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 making a parent region allows more flexibility.

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"  region="whole"/>
                <img src="left1.jpg"  region="leftpane"/>
                <img src="right1.jpg" region="rightpane"/>
            </par>
            <par>
                <phil                  region="whole" transition="diagwipe"/>
                <img  src="back2.jpg"  region="whole"/>
                <img  src="left2.jpg"  region="leftpane"/>
                <img  src="right2.jpg" 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 <phil> 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"  region="whole"/>
                <img src="left1.jpg"  region="leftpane"/>
                <img src="right1.jpg" region="rightpane"/>
            </par>
            <par>
                <phil                  region="whole" transition="diagwipe"/>
                <img  src="back2.jpg"  region="whole"/>
                <img  src="left2.jpg"  region="leftpane"/>
                <img  src="right2.jpg" 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.

Appendix A. References

[CC/PP]
"CC/PP", technology of the W3C Mobile Interest Group. W3C Note,
Available at http://www.w3.org/TR/NOTE-CCPP/

[CSS1]
"Cascading Style Sheets, level 1", Håkon Wium Lie, Bert Bos, 17 Dec 1996, revised 11 Jan 1999. W3C Recommendation,
Available at http://www.w3.org/TR/REC-CSS1

[CSS2]
"Cascading Style Sheets, level 2", Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, 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,12 May, 1998. W3C Recommendation,
Available at http://www.w3.org/TR/REC-CSS2/selector.html#q1

[DATETIME]
"Date and Time Formats", M. Wolf, C. Wicksteed, 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"
Available at http://www.w3.org/TR/REC-DOM-Level-1

[DOM2Events]
"Document Object Model Level 2 Events", W3C Working Draft. T. Pixley
Available at http://www.w3.org/TR/WD-DOM-Level-2/events.html

[DOM2]
"W3C (World Wide Web Consortium) Document Object Model (DOM) Level 2 Specification.
Available at http://www.w3.org/TR/WD-DOM-Level-2

[draft-ietf-avt-rtp-mime-00]
"MIME Type Registration of RTP Payload Formats", Steve Casner and Philipp Hoschka, June 1999.
Available at ftp://ftpeng.cisco.com/casner/outgoing/draft-ietf-avt-rtp-mime-00.txt

[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

[HTML40]
"HTML 4.0 Specification" D. Raggett, A. Le Hors, I. Jacobs, 24 Apr 98.
Available at http://www.w3.org/TR/REC-html40

[NAMESPACES]
"Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999.
Available at http://www.w3.org/TR/REC-xml-names

[RDFsyntax]
"Resource Description Framework (RDF) Model and Syntax Specification", Ora Lassila and Ralph R. Swick, 03 March 1999. Available at http://www.w3.org/TR/REC-rdf-syntax/

[RDFschema]
"Resource Description Framework (RDF) Schema Specification", Dan Brickley and R.V. Guha, 22 February 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

[SMIL10]
"Synchronized Multimedia Integration Language (SMIL) 1.0" P. Hoschka, 15 June 1998.
Available at http://www.w3.org/TR/REC-smil.

[SMIL-ANIMATION]
"SMIL Animation Module" Patrick Schmitz, Aaron Cohen and Philipp Hoschka November xxx 1999.
Available at http://www.w3.org/TR/smil-animation/

[SMIL-CSS2]
"Displaying SMIL Basic Layout with a CSS2 Rendering Engine".
Available at: http://www.w3.org/TR/NOTE-CSS-smil.html

[SMIL-DOM]
"Synchronized Multimedia Integration Language Document Object Model (DOM)".
A W3C Working Draft.Available at http://www.w3.org/TR/SMIL-Boston-DOM

[SMIL-MEDIA]
"The SMIL Media Object Module", Philipp Hoschka, Rob Lanphier.
Available at http://www.w3.org/1999/08/WD-smil-boston-19990803/Media/extended-media-object.

[SMIL-MOD]
"Synchronized Multimedia Modules based upon SMIL 1.0", Patrick Schmitz, Ted Wugofski and Warner ten Kate.
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.
Available at http://www.w3.org/TR/SVG

[VOYAGER]
"Synchronization and Special Effects in Voyager", Ted Wugofski
Available at http://www.w3.org/MarkUp/Group/Contrib/Wugofski/sync.html

[XHTML10]
"The Extensible HyperText Markup Language: A Reformulation of HTML 4.0 in XML 1.0" W3C Proposed Recommendation 24 August 1999.
Available at http://www.w3.org/TR/xhtml1

[XLINK]
"XML Linking Language (XLink)", S. DeRose, D. Orchard and B. Trafford, 26 July 1999.
Available at http://www.w3.org/TR/xlink

[XML10]
"Extensible Markup Language (XML) 1.0" T. Bray, J. Paoli and C.M. Sperberg-McQueen, 10 February 1998.
Available at http://www.w3.org/TR/REC-xml

[XML-NS]
"Namespaces in XML" T. Bray, D. Hollander and A. Layman, 14 January 1999.
Available at http:/www.w3.org/TR/REC-xml-names

[XMOD]
"Modularization of XHTML", Shane McCarron, Murray Altheim, et al. W3C WD, work in progress.
Available at http://www.w3.org/TR/xhtml-modularization

[XPTR]
"XML Pointer Language (XPointer)" Steve DeRose and Ron Daniel Jr., W3C Working Draft 9 July 1999.
Available at http://www.w3.org/TR/WD-xptr

[XSL]
"Extensible Stylesheet Language (XSL) Specification" Stephen Deach, W3C Working Draft 21 Apr 1999.
Available at http://www.w3.org/TR/WD-xsl/