10. The SMIL 2.1 Timing and Synchronization Module

Editor for SMIL 2.1
Dick Bulterman, CWI/Amsterdam

Editors for SMIL 2.0
Patrick Schmitz, Microsoft
Jeff Ayars, RealNetworks
Bridie Saccocio, RealNetworks
Muriel Jourdan, INRIA.

Table of contents

10.1 Overview and Summary of Changes for SMIL 2.1

This section is informative

The SMIL 2.1 specification leaves the basic syntax and semantics of the SMIL 2.0 timing model unchanged [SMIL20-timing]. The only changes for SMIL 2.1 are that SMIL 2.0's ExclTimeContainers module is deprecated and replaced with two new modules: BasicExclTimeContainers and BasicPriorityClassContainers, an errata note has been integrated into the specification that clarifies the default event base for animation elements, and a fix for a text error in the fourth bullet of the **begin** and **end** values was integrated into the text. The repartitioning of the ExclTimeContainers module is done to reduce the implementation burden of the **excl** element on low-powered devices or in implementations in which the full functionality of the priority class mechanism of SMIL 2.0 is not required. The clarification of the event base for animation elements makes integration with SMIL timing and SMIL animation clearer for implementers. The change related to the **begin** and **end** values allows an unresolved name to be ignored.

As a result of this change, SMIL 2.1 Timing and Synchronization support is broken down into 16 modules instead of the 15 modules used in SMIL 2.0. These modules are described in Appendix A: SMIL Timing and Synchronization modules.

10.2 Introduction

This section is informative

SMIL 1.0 solved fundamental media synchronization problems and defined a powerful way of choreographing multimedia content. SMIL 2.1 extends the timing and synchronization support, adding capabilities to the timing model and associated syntax. Some SMIL 1.0 syntax has been changed or deprecated. This section of the document specifies the Timing and Synchronization module.

There are two intended audiences for this module: implementers of SMIL 2.1 document viewers or authoring tools, and authors of other XML languages who wish to integrate timing and synchronization support. A language with which this module is integrated is referred to as a *host language*. A document containing SMIL Timing and Synchronization elements and attributes is referred to as a *host document*.

As this module is used in different profiles (i.e. host languages), the associated syntax requirements may vary. Differences in syntax should be minimized as much as is practical.

SMIL 2.1 Timing and Synchronization support is broken down into 16 modules, allowing broad flexibility for language designers integrating this functionality. These modules are described in Appendix A: SMIL Timing and Synchronization modules.

10.3 Overview of SMIL timing

This section is informative

SMIL Timing defines elements and attributes to coordinate and synchronize the presentation of *media* over time. The term *media* covers a broad range, including *discrete* media types such as still images, text, and vector graphics, as well as *continuous* media types that are intrinsically time-based, such as video, audio and animation.

Three synchronization elements support common timing use-cases:

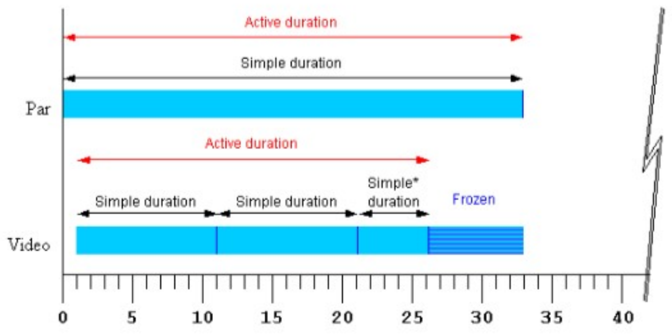
- The **<seq>** element plays the child elements one after another in a *sequence*.
- The **<excl>** element plays one child at a time, but does not impose any order.
- The **<par>** element plays child elements as a group (allowing "parallel" playback).

These elements are referred to as *time containers*. They group their contained children together into coordinated timelines.

SMIL Timing also provides attributes that can be used to specify an element's timing behavior. Elements have a begin, and a *simple duration*. The begin can be specified in various ways - for example, an element can begin at a given time, or based upon when another element begins, or when some event (such as a mouse click) happens. The *simple duration* defines the basic presentation duration of an element. Elements can be defined to repeat the simple duration, a number of times or for an amount of time. The simple duration and any effects of repeat are combined to define the *active duration*. When an element's active duration has ended, the element can either be removed from the presentation or *frozen (*held in its final state), e.g. to fill any gaps in the presentation.

An element *becomes active* when it begins its active duration, and *becomes inactive* when it ends its active duration. Within the active duration, the element is *active*, and outside the active duration, the element is *inactive*.

Figure 1 illustrates the basic support of a repeating element within a simple **<par>** time container. The corresponding syntax is included with the diagram.

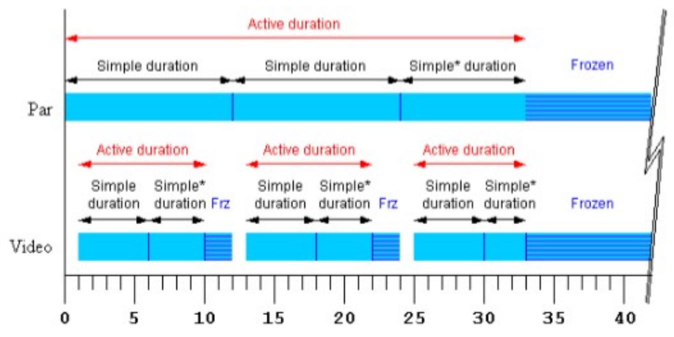


```
<par begin="0s" dur="33s">
   <video begin="1s" dur="10s" repeatCount="2.5" fill="freeze" .../>
</par>
```

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

The attributes that control these aspects of timing can be applied not only to media elements, but to the time containers as well. This allows, for example, an entire sequence to be repeated, and to be coordinated as a unit with other media and time containers. While authors can specify a particular simple duration for a time container, it is often easier to leave the duration unspecified, in which case the simple duration is defined by the contained child elements. When an element does not specify a simple duration, the time model defines an *implicit* simple duration for the element. For example, the implicit simple duration of a sequence is based upon the sum of the active durations of all the children.

Each time container also imposes certain *defaults* and *constraints* upon the contained children. For example in a **<seq>**, elements begin *by default* right after the previous element ends, and in all time containers, the active duration of child elements is *constrained* not to extend past the end of the time container's simple duration. Figure 2 illustrates the effects of a repeating **<par>** time container as it constrains a **<video>** child element.



```
<par begin="0s" dur="12s" repeatDur="33s" fill="freeze" >
   <video begin="1s" dur="5s" repeatCount="1.8" fill="freeze" .../>
</par>
```

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

The SMIL *Timing Model* defines how the time container elements and timing attributes are interpreted to construct a *time graph*. The *time graph* is a model of the presentation schedule and synchronization relationships. The time graph is a dynamic structure, changing to reflect the effect of user events, media delivery, and DOM control of the presentation. At any given instant, the time graph models the document at that instant, and the semantics described in this module. However, as user events or other factors cause changes to elements, the semantic rules are re-evaluated to yield an updated time graph.

When a **begin** or end value refers to an event, or to the begin or active end of another element, it may not be possible to calculate the time value. For example, if an element is defined to begin on some event, the begin time will not be known until the event happens. Begin and end values like this are described as *unresolved*. When such a time becomes known (i.e. when it can be calculated as a presentation time), the time is said to be *resolved.* A resolved time is said to be *definite* if it is not the value "indefinite". See also the discussion of Unifying scheduled and interactive timing.

In an ideal environment, the presentation would perform precisely as specified. However, various real-world limitations (such as network delays) can influence the actual playback of media. How the presentation application adapts and manages the presentation in response to media playback problems is termed *runtime synchronization behavior*. SMIL includes attributes that allow the author to control the runtime synchronization behavior for a presentation.

10.4 Language definition

This section is informative

The timing model is defined by building up from the simplest to the most complex concepts: first the basic timing and simple duration controls, followed by the attributes that control repeating and constraining the active duration. Finally, the elements that define time containers are presented.

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

10.4.1 Attributes

This section defines the set of timing attributes that are common to all of the SMIL synchronization elements.

Unless otherwise specified below, if there is any error in the argument value syntax for an attribute, the attribute will be ignored (as though it were not specified).

The begin and dur attributes: basic timing support

This section is informative

The basic timing for an element is described using the **begin** and **dur** attributes. Authors can specify the begin time of an element in a variety of ways, ranging from simple clock times to the time that an event (e.g. a mouse click) happens. The simple duration of an element is specified as a simple time value. The **begin** attribute syntax is described below. The normative syntax rules for each attribute value variant are described in Timing attribute value grammars; an attribute value syntax summary is provided here as an aid to the reader.

This section is normative

begin : smil-1.0-syncbase-value | begin-value-list
Defines when the element becomes active.
The attribute value is either a SMIL 1.0 syncbase declaration, or a semi-colon separated list of values.

smil-1.0-syncbase-value : "id(" Id-value ")" ("(" ("begin" | "end" | Clock-value) ")")?
Deprecated. Describes a syncbase and an offset from that syncbase. The element begin is defined relative to the begin or active end of another element.

begin-value-list : begin-value (";" begin-value-list)?
A semi-colon separated list of begin values. The interpretation of a list of begin times is detailed in the section Evaluation of begin and end time lists.

begin-value : (offset-value | syncbase-value | event-value | repeat-value | accesskey-value | media-marker-value | wallclock-sync-value | "indefinite")
Describes the element begin.

offset-value : ("+" | "-")? Clock-value
Describes the element begin as an offset from an implicit syncbase. The definition of the implicit syncbase depends upon the element's parent time container. The offset is measured in parent simple time.

syncbase-value : (Id-value "." ("begin" | "end")) (("+" | "-") Clock-value)?
Describes a syncbase and an offset from that syncbase. The element begin is defined relative to the begin or active end of another element.

event-value : (Id-value ".")? (event-ref) (("+" | "-") Clock-value)?
Describes an event and an optional offset that determine the element begin. The element begin is defined relative to the time that the event is raised. Events may be any event defined for the host language in accordance with [DOM2Events]. These may include user-interface events, event-triggers

transmitted via a network, etc. Details of event-based timing are described in the section below on Unifying Event-based and Scheduled Timing.

repeat-value : (Id-value ".")? "repeat(" integer ")" (("+" | "-") Clock-value)?
: Describes a qualified repeat event. The element begin is defined relative to the time that the repeat event is raised with the specified iteration value.

accesskey-value : "accesskey(" character ")"(("+" | "-") Clock-value)?
: Describes an accesskey that determines the element begin. The element begin is defined relative to the time that the accesskey character is input by the user.

media-marker-value : Id-value ".marker(" marker-name ")"
: Describes the element begin as a named marker time defined by a media element.

wallclock-sync-value : "wallclock(" wallclock-value ")"
: Describes the element begin as a real-world clock time. The wallclock time syntax is based upon syntax defined in [ISO8601].

"indefinite"
: The begin of the element will be determined by a "beginElement()" method call or a hyperlink targeted to the element.
The SMIL Timing and Synchronization DOM methods are described in the Reserveed DOM methods section.
Hyperlink-based timing is described in the Hyperlinks and timing section.

Begin value semantics

This section is normative

- Children of a **seq** can only specify a non-negative offset value for begin (see The seq element).
- If no **begin** is specified, the default timing is dependent upon the time container.
- If there is a syntax error in any individual value in the list of begin or end values (i.e. the value does not conform to the defined syntax for any of the time values), the host language must specify how the user agent deals with this.
- A time value may conform to the defined syntax but still be invalid (e.g. if an unknown element is referenced by ID in a syncbase value). If there is such an evaluation error in an individual value in the list of begin or end values, the individual value will be will be treated as though "indefinite" were specified, and the rest of the list will be processed normally. If no legal value is specified for a begin or end attribute, the element assumes an "indefinite" begin or end time (respectively).
- The deprecated SMIL 1.0-syncbase-values are semantically equivalent to the following SMIL 2.1 begin-value types:
  - `id(`*`Id-value`*`)(begin)` is equivalent to *`Id-value`*`.begin`
  - `id(`*`Id-value`*`)(end)` is equivalent to *`Id-value`*`.end`
  - `id(`*`Id-value`*`)(`*`Clock-value`*`)` is equivalent to *`Id-value`*`.begin+ `*`Clock-value`*

This section is informative

Children of a **par** begin by default when the **par** begins (equivalent to `begin="0s"`). Children of a **seq** begin by default when the previous child ends its active duration (equivalent to `begin="0s"`); the first child begins by default when the parent **seq** begins. Children of an **excl** default to a begin value of "**`indefinite`**".

The **begin** value can specify a list of times. This can be used to specify multiple "ways" or "rules" to begin an element, e.g. if any one of several events is raised. A list of times can also define multiple begin times, allowing the element to play more than once (this behavior can be controlled, e.g. to only allow the earliest begin to actually be used - see also the **restart** attribute).

In general, the earliest time in the list determines the begin time of the element. There are additional constraints upon the evaluation of the begin time list, detailed in Evaluation of begin and end time lists.

Note that while it is legal to include "indefinite" in a list of values for **begin**, "indefinite" is only really useful as a single value. Combining it with other values does not impact begin timing, as DOM begin methods can be called with or without specifying "indefinite" for **begin**.

When a begin time is specified as a syncbase variant, a marker value or a wallclock value, the defined time must be converted by the implementation to a time that is relative to the parent time container (i.e. to the equivalent of an offset value). This is known as *timespace conversion*, and is detailed in the section Converting between local and global times.

Handling negative offsets for begin

This section is informative

The use of negative offsets to define begin times merely defines the synchronization relationship of the element. It does not in any way override the time container constraints upon the element, and it cannot override the constraints of presentation time.

This section is normative

- For children of **<par>** and **<excl>** time containers, the computed offset relative to the parent begin time may be negative.
- A begin time may be specified with a negative offset relative to an event or to a syncbase that is not initially resolved. When the syncbase or eventbase time is resolved, the computed time may be in the past.

The computed begin time defines the *scheduled synchronization relationship* of the element, even if it is not possible to begin the element at the computed time. The time model uses the computed begin time, and not the observed time of the element begin.

This section is informative

If an element has a begin time that resolves to a time before the parent time container begins, the parent time container constraint still applies. For example:

```
<par>
   <video id="vid" begin="-5s" dur="10s" src="movie.mpg" />
   <audio begin="vid.begin+2s" dur="8s" src="sound.au" />
</par>
```

The **video** element cannot begin before the **par** begins. The begin is simply defined to occur *"in the past"* when the **par** begins. The viewer will observe that the video begins 5 seconds into the media, and ends after 5 seconds. Note that the audio element begins relative to the video begin, and that the computed begin time is used, and not the observed begin time as constrained by the parent. Thus the audio begins 3 seconds into the media, and also lasts 5 seconds.

The behavior can be thought of as a clipBegin value applied to the element, that only applies to the first iteration of repeating elements. In the example above, if either element were defined to repeat, the second and later iterations of the media would play from the beginning of the media (see also the repeatCount, repeatDur, and repeat attributes: repeating elements).

This section is normative

- When a begin time is resolved to be in the past (i.e., before the current presentation time), the element begins immediately, but acts as though it had begun at the specified time (playing from an offset into the media).

The behavior can be thought of as a clipBegin value applied to the element, that only applies to the first iteration of repeating elements.

The element will actually begin at the time computed according to the following algorithm:

```
Let o be the offset value of a given begin value,
d be the associated simple duration,
AD be the associated active duration.
Let rAt be the time when the begin time becomes resolved.
Let rTo be the resolved sync-base or event-base time without the offset
Let rD be rTo - rAt.  If rD < 0 then rD is set to 0.

If AD is indefinite, it compares greater than any value of o or ABS(o).
REM( x, y ) is defined as x - (y * floor( x/y )).
If y is indefinite or unresolved, REM( x, y ) is just x.

Let mb = REM( ABS(o), d ) - rD

If ABS(o) >= AD then the element does not begin.
Else if mb >= 0 then the media begins at mb.
Else the media begins at mb + d.
```

If the element repeats, the iteration value of the `repeat` event has the calculated value based upon the above computed begin time, and not the observed number of repeats.

This section is informative

Thus for example:

```
<smil ...>
...
<ref begin="foo.activateEvent-8s" dur="3s" repeatCount="10" .../>
...
</smil>
```

The element begins when the user activates (for example, clicks on) the element "foo". Its calculated begin time is actually 8 seconds earlier, and so it begins to play at 2 seconds into the 3 second simple duration, on the third repeat iteration. One second later, the fourth iteration of the element will begin, and the associated `repeat` event will have the iteration value set to 3 (since it is zero based). The element will end 22 seconds after the activation. The `beginEvent` event is raised when the element begins, but has a time stamp value that corresponds to the defined begin time, 8 seconds earlier. Any time dependents are activated relative to the computed begin time, and not the observed begin time.

Note: If script authors wish to distinguish between the computed repeat iterations and observed repeat iterations, they can count actual `repeat` events in the associated event handler.

Negative begin delays

A begin time specifies a synchronization relationship between the element and the parent time container. Syncbase variants, eventbase, marker and wallclock timing are implicitly converted to an offset on the parent time container, just as an offset value specifies this directly. For children of a **seq**, the result is always a positive offset from the begin of the **seq** time container. However, for children of **par** and **excl** time containers the computed offset relative to the parent begin time may be negative.

Note that an element cannot actually begin until the parent time container begins. An element with a negative time delay behaves as if it had begun earlier. The presentation effect for the element (e.g. the display of visual media) is equivalent to that for a **clipBegin** value (with the same magnitude) for the first -- and only the first -- iteration of a repeated element. If no repeat behavior is specified, the element presentation effect of a negative begin offset is equivalent to a **clipBegin** specification with the same magnitude as the offset value. Nevertheless, the *timing* side effects are *not* equivalent to a **clipBegin** value as described. Time dependents of the begin value will behave as though the element had begun earlier.

Dur value semantics

The length of the simple duration is specified using the **dur** attribute. The **dur** attribute syntax is described below.

This section is normative

dur
: Specifies the simple duration.
: The attribute value can be any of the following:

  Clock-value
  : Specifies the length of the simple duration, measured in element active time.
  : Value must be greater than 0.

  "media"
  : Specifies the simple duration as the intrinsic media duration. This is only valid for elements that define media.

  "indefinite"
  : Specifies the simple duration as indefinite.

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

If the "`media`" attribute value is used on an element that does not define media (e.g. on the SMIL 2.1 time container elements **par**, **seq** and **excl**), the attribute will be ignored (as though it were not specified). Contained media such as the children of a **par** are not considered media directly associated with the element.

If the element does not have a (valid) **dur** attribute, the simple duration for the element is defined to be the implicit duration of the element. The implicit duration depends upon the type of an element. The primary distinction is between different types of media elements and time containers. If the media element has no timed children, it is described as a *simple media element*.

- For simple media elements that specify *continuous* media (i.e. media with an inherent notion of time), the implicit duration is the intrinsic duration of the media itself - e.g. video and audio files have a defined duration. Note that **clipBegin** and **clipEnd** attributes on a media element can override the intrinsic media duration, and will define the implicit duration. See also the Media Object module.
- For simple media elements that specify *discrete* media (some times referred to as "static" media), the implicit duration is defined to be 0.
- For **par**, **seq** and **excl** time containers, and media elements that are also time containers, the implicit simple duration is a function of the type of the time container and of its **endsync** attribute. For details see the section Time container durations.

If the author specifies a value for **dur** that is *shorter* than the implicit duration for an element, the implicit duration will be cut short by the specified simple duration.

If the author specifies a simple duration that is *longer* than the implicit duration for an element, the implicit duration of the element is extended to the specified simple duration:

- For a discrete media element, the media will be shown for the specified simple duration.
- For a continuous media element, the ending state of the media (e.g. the last frame of video) will be shown from the end of the intrinsic media duration to the end of the specified simple duration. This only applies to visual media - aural media will simply stop playing (i.e. be silent).
- For a seq time container, the last child is frozen until the end of the simple duration of the seq if and only if its fill behavior is "freeze" or "hold" (otherwise the child just ends without freezing).
- Children of a par or excl are frozen until the end of the simple duration of the par or excl if and only if the children's fill behavior is "freeze" or "hold" (otherwise the children just ends without freezing).

Note that when the simple duration is "indefinite", some simple use cases can yield surprising results. See the related example #4 in Appendix B.

Examples

The following example shows simple offset begin timing. The **<audio>** element begins 5 seconds after the **<par>** time container begins, and ends 4 seconds later.

```
<par>
   <audio src="song1.au" begin="5s" dur="4s" />
</par>
```

The following example shows syncbase begin timing. The **<img>** element begins 2 seconds after the **<audio>** element begins.

```
<par>
   <audio id="song1" src="song1.au" />
   <img src="img1.jpg" begin="song1.begin+2s" />
</par>
```

Elements can also be specified to begin in response to an event. In this example, the image element begins (appears) when the user clicks on element "show". The image will end (disappear) 3 and a half seconds later.

```
<smil ...>
...
```

```
<text id="show" ... />
<img begin="show.activateEvent" dur="3.5s" ... />
...
</smil ...>
```

The end attribute: controlling active duration

This section is informative

SMIL 2.1 provides an additional control over the active duration. The end attribute allows the author to constrain the active duration by specifying an end value using a simple offset, a time base, an event-base, a syncbase, or DOM methods calls. The rules for combining the attributes to compute the active duration are presented in the section, Computing the active duration.

The normative syntax rules for each attribute value variant are described in the section Timing attribute value grammars; a syntax summary is provided here as an aid to the reader.

This section is normative

end : smil-1.0-syncbase-value | end-value-list

Defines an end value for the element that can constrain the active duration. The attribute value is either a SMIL 1.0 syncbase declaration, a semi-colon separated list of values.

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

Deprecated. Describes a syncbase and an offset from that syncbase. The end value is defined relative to the begin or active end of another element.

end-value-list : end-value (";" end-value-list)?

A semi-colon separated list of end values. The interpretation of a list of end times is detailed in the section Evaluation of begin and end time lists.

end-value : (offset-value | syncbase-value | event-value | repeat-value | accesskey-value | media-marker-value | wallclock-sync-value | "indefinite")

Describes the end value of the element.

offset-value : ("+" | "-")? Clock-value

Describes the end value as an offset from an implicit syncbase. The definition of the implicit syncbase depends upon the element's parent time container. The offset is measured in parent simple time.

syncbase-value : (Id-value "." ("begin" | "end")) (("+" | "-") Clock-value)?

Describes a syncbase and an offset from that syncbase. The end value is defined relative to the begin or active end of another element.

event-value : (Id-value ".")? (event-ref) (("+" | "-") Clock-value)?

Describes an event and an optional offset that determine the end value. The end value is defined relative to the time that the event is raised. Events may be any event defined for the host language in accordance with [DOM2Events]. These may include user-interface events, event-triggers transmitted via a network, etc. Details of event-based timing are described in the section below on Unifying Event-based and Scheduled Timing.

repeat-value : (Id-value ".")? "repeat(" integer ")" (("+" | "-") Clock-value)?

Describes a qualified repeat event. The end value is defined relative to the time that the repeat event is raised with the specified iteration value.

accesskey-value : "accesskey(" character ")"(("+" | "-") Clock-value)?
: Describes an accesskey that determines the end value. The end value is defined as the time that the accesskey character is input by the user.

media-marker-value : Id-value ".marker(" marker-name ")"
: Describes the end value as a named marker time defined by a media element.

wallclock-sync-value : "wallclock(" wallclock-value ")"
: Describes the end value as a real-world clock time. The wallclock time is based upon syntax defined in [ISO8601].

"indefinite"
: The end value of the element will be determined by an `endElement()` method call.
The SMIL Timing and Synchronization DOM methods are described in the Reserved DOM methods section.

If an end attribute is specified but none of dur, repeatCount and repeatDur are specified, the simple duration is defined to be indefinite, and the end value constrains this to define the active duration. The behavior of the simple duration in this case is defined in Dur value semantics, as though dur had been specified as "indefinite".

If the end value becomes resolved while the element is still active, and the resolved time is in the past, the element should end the active duration immediately. Time dependents defined relative to the end of this element should be resolved using the computed active end (which may be in the past), and not the observed active end.

The deprecated SMIL 1.0-syncbase-values are semantically equivalent to the following SMIL 2.1 end-value types:

- `id(`*`Id-value`*`)(begin)` is equivalent to *`Id-value`*`.begin`
- `id(`*`Id-value`*`)(end)` is equivalent to *`Id-value`*`.end`
- `id(`*`Id-value`*`)(`*`Clock-value`*`)` is equivalent to *`Id-value`*`.begin+`*`Clock-value`*

This section is informative

The end value can specify a list of times. This can be used to specify multiple "ways" or "rules" to end an element, e.g. if any one of several events is raised. A list of times can also define multiple end times that can correspond to multiple begin times, allowing the element to play more than once (this behavior can be controlled - see also the restart attribute).

In the following example, the dur attribute is not specified, and so the simple duration is defined to be the implicit media duration. In this case (and this case only) the value of end will extend the active duration if it specifies a duration greater than the implicit duration. The video will be shown for 8 seconds, and then the last frame will be shown for 2 seconds.

```
<video end="10s" src="8-SecondVideo.mpg" .../>
```

If an author wishes to specify the implicit duration as well as an end constraint, the dur attribute can be specified as "`media`". In the following example, the element will end at the earlier of the intrinsic media duration, or a mouse click:

```
<html ...>
...
<video dur="media" end="click" src="movie.mpg" .../>
...
</html>
```

These cases arise from the use of negative offsets in the sync-base and event-base forms, and authors should be aware of the complexities this can introduce. See also Handling negative offsets for end.

In the following example, the active duration will end at the earlier of 10 seconds, or the end of the "foo" element. This is particularly useful if "foo" is defined to begin or end relative to an event.

```
<audio src="foo.au" dur="2s" repeatDur="10s"
       end="foo.end" .../>
```

In the following example, the active duration will end at 10 seconds, and will cut short the simple duration defined to be 20 seconds. The effect is that only the first half of the element is actually played. For a simple media element, the author could just specify this using the dur attribute. However in other cases, it is sometimes important to specify the simple duration independent of the active duration.

```
<par>
   <audio src="music.au" dur="20s" end="10s" ... />
</par>
```

In the following example, the element begins when the user activates (e.g., clicks on) the "gobtn" element. The active duration will end 30 seconds after the parent time container begins.

```
<smil ...>
...
<par>
<audio src="music.au" begin="gobtn.activateEvent" repeatDur="indefinite"
          end="30s" ... />
     <img src="foo.jpg" dur="40s" ... />
</par>
...
</smil>
```

Note that if the user has not clicked on the target element before 30 seconds elapse, the element will never begin. In this case, the element has no active duration and no active end.

The defaults for the event syntax make it easy to define simple interactive behavior. The following example stops the image when the user clicks on the element.

```
<html ...>
...
<img src="image.jpg" end="click" />
...
</html>
```

Using **end** with an event value enables authors to end an element based on either an interactive event or a maximum active duration. This is sometimes known as *lazy interaction*.

In this example, a presentation describes factory processes. Each step is a video, and set to repeat 3 times to make the point clear. Each element can also be ended by clicking on the video, or on some element "next" that indicates to the user that the next step should be shown.

```
<smil ...>
...
<seq>
  <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
```

```
    <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
    <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
    <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
    <video dur="5s" repeatCount="3" end="activateEvent; next.activateEvent" .../>
  </seq>
  ...
  </smil>
```

In this case, the active end of each element is defined to be the earlier of 15 (5s dur * 3 repeats) seconds after it begins, or a click on "next". This lets the viewer sit back and watch, or advance the presentation at a faster pace.

Handling negative offsets for end

This section is normative

- An end time may be specified with a negative offset relative to an event or to a syncbase that is not initially resolved.
- When the syncbase or eventbase time is resolved, the computed time may be in the past.
- The computed time defines the *scheduled synchronization relationship* of the element, even if it is not possible to end the element at the computed time.
- When an end time is defined to be in the past, the element ends immediately. The defined end time is the computed time, and not the observed or performed time of the element end.

The min and max attributes: more control over the active duration

This section is informative

The min/max attributes provide the author with a way to control the lower and upper bound of the element active duration.

This section is normative

min
: Specifies the minimum value of the active duration.
The attribute value can be either of the following:

Clock-value
: Specifies the length of the minimum value of the active duration, measured in element active time.
Value must be greater than or equal to 0.

"media"
: Specifies the minimum value of the active duration as the intrinsic media duration. This is only valid for elements that define media.

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

The default value for **min** is "0". This does not constrain the active duration at all.

max
: Specifies the maximum value of the active duration.
The attribute value can be either of the following:

Clock-value
: Specifies the length of the maximum value of the active duration, measured in element active time.
Value must be greater than 0.

"media"
: Specifies the maximum value of the active duration as the intrinsic media duration. This is only valid for elements that define media.

"indefinite"
: The maximum value of the duration is indefinite, and so is not constrained.

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

The default value for **max** is "indefinite". This does not constrain the active duration at all.

If the "`media`" argument value is specified for either **min** or **max** on an element that does not define media (e.g. on the SMIL 2.1 time container elements **par**, **seq** and **excl**), the respective attribute will be ignored (as though it were not specified). Contained media such as the children of a **par** are not considered media directly associated with the element.

If both **min** and **max** attributes are specified then the **max** value must be greater than or equal to the **min** value. If this requirement is not fulfilled then both attributes are ignored.

The rule to apply to compute the active duration of an element with **min** or **max** specified is the following: Each time the active duration of an element is computed (i.e. for each interval of the element if it begins more than once), this computation is made without taking into account the **min** and **max** attributes (by applying the algorithm described in Computing the active duration). The result of this step is checked against the **min** and **max** bounds. If the result is within the bounds, this first computed value is correct. Otherwise two situations may occur:

- if the first computed duration is greater than the max value, the active duration of the element is defined to be equal to the **max** value (see the first example below).
- if the first computed duration is less than the **min** value, the active duration of the element becomes equal to the **min** value and the behavior of the element is as follows :
    - if the repeating duration (or the simple duration if the element doesn't repeat) of the element is greater than **min** then the element is played normally for the (**min** constrained) active duration. (see the second and third examples below).
    - otherwise the element is played normally for its repeating duration (or simple duration if the element does not repeat) and then is frozen or not shown depending on the value of the **fill** attribute (see the fourth and fifth examples below).

This section is informative

The following examples illustrate some simple use cases for **min** and **max** attributes:

Example 1. In the following example, the video will only play for 10 seconds.

```
<smil ...>
...
<par >
```

```
      <video id="video_of_15s" max="10s".../>
   </par>
   ...
   </smil>
```

Example 2. In the following example, if an activate event happens before 10 seconds, this activation (e.g. click) does not interrupt the video immediately, but the video plays until 10 seconds and then stops. If a click event happens after 10 seconds, the video plays (repeating) until the click happens. Note, the endEvent is only raised if a click occurs after 10 seconds, not at the simple end of each repeat.

```
   <smil ...>
   ...
   <par >
      <video id="video_of_15s" repeatDur="indefinite" end="activateEvent" min="10s".../>
   </par>
   ...
   </smil>
```

Example 3. In the following example, if an activate event happens on element "foo" at 5 seconds, this event does not end the time container immediately, but rather at 12 seconds. The simple duration is defined to be "indefinite" (because an **end** attribute is specified with no **dur** attribute), and so the time container plays normally until it ends at 12 seconds.

```
   <smil ...>
   ...
   <par end="foo.activateEvent" min="12s" >
      <video id="video_of_15s" .../>
      <video id="video_of_10s" .../>
   </par>
   ...
   </smil>
```

Example 4. In the following example, if a click event happens on the first video at 5 seconds, then the simple duration of the time container is computed as 5 seconds. Respecting the **fill** attribute in the time between the end of the simple duration and the end of the active duration, the two videos are frozen between 5 seconds and 12 seconds.

```
   <html ...>
   ...
   <par endsync="first" min="12s" fill="freeze" >
      <video id="video_of_15s" end="click" ...>
      <video id="video_of_10s" .../>
   </par>
   ...
   </html>
```

Example 5. In the following example, the time container simple duration is defined to be 5 seconds, and the min constraint defines the active duration to be 12 seconds. Since the default value of **fill** in this case is `"remove"`, nothing is shown for the time container between 5 seconds and 12 seconds.

```
   <par dur="5s" min="12s" >
      <video id="video_of_15s"/>
      <video id="video_of_10s" />
   </par>
```

The min attribute and negative begin times

If an element is defined to begin before its parent (e.g. with a simple negative offset value), the min duration is measured from the calculated begin time not the observed begin (see example 1 below). This means that the min value may have no observed effect (as in example 2 below).

Example 1. In the following example, the image will be displayed from the beginning of the time container for 2 seconds.

```
<par>
   <img id="img" begin="-5s" min="7s" dur="5s" .../>
</par>
```

Example 2. In the following example, the image will not be displayed at all.

```
<par>
   <img id="img" begin="-5s" min="4s" dur="2s" .../>
</par>
```

See also the sections The min attribute and restart and Time container constraints on child durations.

Timing attribute value grammars

This section is normative

The syntax specifications are defined using EBNF notation as defined in XML 1.1 [XML11]

In the syntax specifications that follow, allowed white space is indicated as "S", defined as follows (taken from the [XML11] definition for 'S'):

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

Begin values

This section is normative

A begin-value-list is a semi-colon separated list of timing specifiers:

```
begin-value-list ::= begin-value (S? ";" S? begin-value-list )?
begin-value      ::= (offset-value | syncbase-value
                      | event-value | repeat-value | accesskey-value
                      | media-marker-value | wallclock-sync-value
                      | "indefinite" )
```

End values

This section is normative

An end-value-list is a semi-colon separated list of timing specifiers:

```
end-value-list ::= end-value (S? ";" S? end-value-list )?
end-value      ::= (offset-value | syncbase-value
                      | event-value | repeat-value | accesskey-value
```

```
                                              | media-marker-value | wallclock-sync-value
                                              | "indefinite" )
```

Parsing timing specifiers

Several of the timing specification values have a similar syntax. To parse an individual item in a value-list, the following approach defines the correct interpretation. In addition, Id-values and Event-symbols are XML NMTOKEN values and as such are allowed to contain the full stop '`.`' and hyphen-minus '-' characters. The reverse solidus character '\' must be used to escape these characters within Id-values and Event-symbols, otherwise these characters will be interpreted as the full stop separator and hyphen-minus sign, respectively. Once these rules are interpreted, but before Id-values in syncbase values, event values, or media-marker values are further handled, all leading and embedded escape characters should be removed.

1. Strip any leading, trailing, or intervening white space characters.
2. If the value begins with a number or numeric sign indicator (i.e. `'+'` or `'-'`), the value should be parsed as an offset value.
3. Else if the value begins with the unescaped token "wallclock", it should be parsed as a wallclock-sync-value.
4. Else if the value is the unescaped token "indefinite", it should be parsed as the value "indefinite".
5. Else: Build a token substring up to but not including any sign indicator (i.e. strip off any offset, parse that separately, and add it to the result of this step). In the following, any '`.`' characters preceded by a reverse solidus '`\`' escape character should not be treated as a separator, but as a normal token character.
    1. If the token contains no '`.`' separator character, then the value should be parsed as an event-value with an unspecified (i.e. default) eventbase-element.
    2. Else if the token ends with the unescaped string "`.begin`" or "`.end`", then the value should be parsed as a syncbase-value.
    3. Else if the token contains the unescaped string "`.marker(`", then the value should be parsed as a media-marker-value.
    4. Else, the value should be parsed as an event-value (with a specified eventbase-element).

This approach allows implementations to treat the tokens **`wallclock`** and **`indefinite`** as reserved element IDs, and **`begin`**, **`end`** and **`marker`** as reserved event names, while retaining an escape mechanism so that elements and events with those names may be referenced.

Clock values

Clock values have the following syntax:

```
Clock-value         ::= ( Full-clock-value | Partial-clock-value | Timecount-value )
Full-clock-value    ::= Hours ":" Minutes ":" Seconds ("." Fraction)?
Partial-clock-value ::= Minutes ":" Seconds ("." Fraction)?
Timecount-value     ::= Timecount ("." Fraction)? (Metric)?
Metric              ::= "h" | "min" | "s" | "ms"
Hours               ::= DIGIT+; any positive number
Minutes             ::= 2DIGIT; range from 00 to 59
Seconds             ::= 2DIGIT; range from 00 to 59
Fraction                ::= DIGIT+
Timecount           ::= DIGIT+
2DIGIT                    ::= DIGIT DIGIT
DIGIT                        ::= [0-9]
```

For Timecount values, the default metric suffix is "s" (for seconds). No embedded white space is allowed in clock values, although leading and trailing white space characters will be ignored.

The following are examples of legal clock values:

- Full clock values:
  `02:30:03` = 2 hours, 30 minutes and 3 seconds
  `50:00:10.25` = 50 hours, 10 seconds and 250 milliseconds
- Partial clock value:
  `02:33` = 2 minutes and 33 seconds
  `00:10.5` = 10.5 seconds = 10 seconds and 500 milliseconds
- Timecount values:

`3.2h`	= 3.2 hours = 3 hours and 12 minutes
`45min`	= 45 minutes
`30s`	= 30 seconds
`5ms`	= 5 milliseconds
`12.467`	= 12 seconds and 467 milliseconds

Fractional values are just (base 10) floating point definitions of seconds. The number of digits allowed is unlimited (although actual precision may vary among implementations). For example:

```
00.5s = 500 milliseconds
00:00.005 = 5 milliseconds
```

Offset values

Offset values are used to specify when an element should begin or end relative to its syncbase.

This section is normative

An offset value has the following syntax:

```
offset-value   ::= ( S? ("+" | "-") S? )? ( Clock-value )
```

- An offset value allows an optional sign on a clock value, and is used to indicate a positive or negative offset.
- The offset is measured in parent simple time.

The implicit syncbase for an offset value is dependent upon the time container:

- For children of a **<par>** or an **<excl>**, the offset is relative to the begin of the parent **<par>** or **<excl>**.
- For children of a **<seq>**, the offset is relative to the active end of the previous child. If there is no previous child, the offset is relative to the begin of the parent **<seq>**. See also The seq time container.

SMIL 1.0 begin and end values

Deprecated.

```
smil-1-syncbase-value  ::= "id(" Id-value ")"
                           ( "(" ( "begin" | "end" | Clock-value) ")" )?
```

ID-Reference values

This section is normative

ID reference values are references to the value of an "id" attribute of another element in the document.

```
Id-value                    ::= Id-ref-value

Id-ref-value                 ::= IDREF | Escaped-Id-ref-value

Escaped-Id-ref-value         ::= Escape-Char NMTOKEN

Escape-Char                 ::= "\"
```

- The IDREF is a legal XML identifier.
- If the Id-ref-value is not an IDREF, then it is treated as an Escaped-ID-ref-value.
- The Escaped-Id-ref-value allows the use of a SMIL 2.1 reserved symbol as an IDREF value for an attribute by prefixing the reserved symbol with a backslash character. In this case the value should be treated as an IDREF and not as the reserved symbol, and the leading backslash is significant in the parsing as detailed in the section on parsing XML identifiers in begin and end values.
- Single characters can also be escaped by using the backslash character, and the character, minus the backslash, should be treated as a literal part of the NMTOKEN.

If the element referenced by the IDREF is ignored as described in the Content Control modules (e.g. if it specifies test attributes that evaluate false), the associated time value (i.e.. the syncbase value or the eventbase value that specifies the Id-value) will be considered invalid.

This section is informative

The semantics of ignored elements may change in a future version of SMIL. One possible semantic is that the associated sync arc arguments will not be invalid, but will instead always be "unresolved". When this behavior needs to be simulated in this version of SMIL Timing and Synchronization, an author can include the value "indefinite" in the list of values for the begin or end attribute.

Syncbase values

A syncbase value starts with a Syncbase-element term defining the value of an "id" attribute of another element referred to as the *syncbase element*.

This section is normative

A syncbase value has the following syntax:

```
Syncbase-value   ::= ( Syncbase-element "." Time-symbol )
                     ( S? ("+"|"-") S? Clock-value )?
Syncbase-element ::= Id-value
Time-symbol      ::= "begin" | "end"
```

- The syncbase element must be another timed element contained in the host document.
- The syncbase element may not be a descendent of the current element.

- If the syncbase element specification refers to an illegal element, the time-value description will be treated as though "indefinite" were specified.

The syncbase element is qualified with one of the following *time symbols:*

begin
: Specifies the begin time of the syncbase element.

end
: Specifies the Active End of the syncbase element.

- The time symbol can be followed by an offset value. The offset value specifies an offset from the time (i.e. the begin or active end) specified by the syncbase and time symbol.
- The offset is measured in parent simple time.
- If the clock value is omitted, it defaults to "0".
- No embedded white space is allowed between a syncbase element and a time-symbol.
- White space will be ignored before and after a "+" or "-" for a clock value.
- Leading and trailing white space characters (i.e. before and after the entire syncbase value) will be ignored.

Examples

`begin="x.end-5s"`	: Begin 5 seconds before "x" ends
`begin=" x.begin "`	: Begin when "x" begins
`end="x.begin + 1min"`	: End 1 minute after "x" begins

Event values

This section is informative

An Event value starts with an Eventbase-element term that specifies the *event-base element*. The event-base element is the element on which the event is observed. Given DOM event bubbling, the event-base element may be either the element that raised the event, or it may be an ancestor element on which the bubbled event can be observed. Refer to DOM-Level2-Events [DOM2Events] for details.

This section is normative

An event value has the following syntax:

```
Event-value       ::= ( Eventbase-element "." )? Event-symbol
                      ( S? ("+"|"-") S? Clock-value )?
Eventbase-element ::= ID
```

The eventbase-element must be another element contained in the host document.

If the Eventbase-element term is missing, the event-base element defaults to the element on which the eventbase timing is specified (the current element). A host language designer may override the definition of the default eventbase element. As an example of this, the SMIL 2.1 Animation modules describe Timing integration requirements for the animation elements (animate, animateMotion, etc.). These requirements specify that the default eventbase element is the target element of the animation. See SMIL 2 Animation section 3.9.1 (Integration requirements).

The event value must specify an Event-symbol. This term is an XML NMTOKEN that specifies the name of the event that is raised on the Event-base element. The host language designer must specify which events can be specified.

- Host language specifications must include a description of legal event names (with "none" as a valid description), and/or allow any name to be used.
- If an integrating language specifies no supported events, the event-base time value is effectively unsupported for that language.
- A host language may choose not to include support for offsets with event values. The language must specify if this support is omitted.
- If the host language allows dynamically created events (as supported by DOM-Level2-Events [DOM2Events]), all possible Event-symbol names cannot be specified and so unrecognized names may not be considered errors.
- Unless explicitly specified by a host language, it is not considered an error to specify an event that cannot be raised on the Event-base element (such as activateEvent or click for audio or other non-visual elements). Since the event will never be raised on the specified element, the event-base value will never be resolved.

The last term specifies an optional offset-value that is an offset from the time of the event.

- The offset is measured in parent simple time. If this term is omitted, the offset is 0.
- No embedded white space is allowed between an eventbase element and an event-symbol.
- White space will be ignored before and after a "+" or "-" for a clock value.
- Leading and trailing white space characters (i.e. before and after the entire eventbase value) will be ignored.

This section is informative

This module defines several events that may be included in the supported set for a host language, including `beginEvent` and `endEvent`. These should not be confused with the syncbase time values. See the section on Events and event model.

The semantics of event-based timing are detailed in Unifying Scheduling and Interactive Timing. Constraints on event sensitivity are detailed in Event sensitivity.

Examples:

`begin=" x.load "`	: Begin when "load" is observed on "x"
`begin="x.focus+3s"`	: Begin 3 seconds after a "focus" event on "x"
`begin="x.endEvent+1.5s"`	: Begin 1 and a half seconds after an "endEvent" event on "x"
`begin="x.repeat"`	: Begin each time a `repeat` event is observed on "x"

The following example describes a qualified repeat eventbase value:

```
<html ...>
...
<video id="foo" repeatCount="10" end="endVideo.click" ... />
<img id="endVideo" begin="foo.repeat(2)" .../>
...
</html>
```

The "endVideo" image will appear when the video "foo" repeats the second time. This example allows the user to stop the video after it has played though at least twice.

Repeat values

Repeat values are a variant on event values that support a qualified repeat event. The `repeat` event defined in Events and event model allows an additional suffix to qualify the event based upon an iteration value.

A repeat value has the following syntax:

```
Repeat-value       ::= ( Eventbase-element "." )? "repeat(" iteration ")"
                          ( S? ("+"|"-") S? Clock-value )?
iteration        ::= DIGIT+
```

If this qualified form is used, the eventbase value will only be resolved when a repeat is observed that has a iteration value that matches the specified iteration.

The qualified repeat event syntax allows an author to respond only to an individual repeat of an element.

Accesskey values

Accesskey values allow an author to tie a begin or end time to a particular key press, independent of focus issues. It is modeled on the HTML accesskey support. Unlike with HTML, user agents should not require that a modifier key (such as "ALT") be required to activate an access key.

An access key value has the following syntax:

```
Accesskey-value  ::= "accesskey(" character ")"
                        ( S? ("+"|"-") S? Clock-value )?
```

The character is a single character from [ISO10646].

The time value is defined as the time that the access key character is input by the user.

Media marker values

Certain types of media can have associated *marker* values that associate a name with a particular point (i.e. a time) in the media. The media marker value provides a means of defining a begin or end time in terms of these marker values. Note that if the referenced id is not associated with a media element that supports markers, or if the specified marker name is not defined by the media element, the associated time may never be resolved.

This section is normative

```
Media-Marker-value ::= Id-value ".marker(" S? marker-name S? ")"
```

- The marker symbol is a string that must conform to the definition of marker names for the media associated with the Id-value.

Wallclock-sync values

This section is informative

Wallclock-sync values have the following syntax. The values allowed are based upon several of the "profiles" described in [DATETIME], which is based upon [ISO8601].

This section is normative

```
wallclock-sync-value  ::= "wallclock(" S? (DateTime | WallTime | Date)  S? ")"
DateTime       ::= Date "T" WallTime
Date           ::= Years "-" Months "-" Days
WallTime       ::= (HHMM-Time | HHMMSS-Time)(TZD)?
HHMM-Time      ::= Hours24 ":" Minutes
HHMMSS-Time    ::= Hours24 ":" Minutes ":" Seconds ("." Fraction)?
Years          ::= 4DIGIT;
Months         ::= 2DIGIT; range from 01 to 12
Days           ::= 2DIGIT; range from 01 to 31
Hours24        ::= 2DIGIT; range from 00 to 23
4DIGIT         ::= DIGIT DIGIT DIGIT DIGIT
TZD            ::= "Z" | (("+" | "-") Hours24 ":" Minutes )
```

- Exactly the components shown here must be present, with exactly this punctuation.
- Note that the "T" appears literally in the string, to indicate the beginning of the time element, as specified in [ISO8601].

This section is informative

```
Complete date plus hours and minutes:

   YYYY-MM-DDThh:mmTZD (e.g. 1997-07-16T19:20+01:00)

Complete date plus hours, minutes and seconds:

   YYYY-MM-DDThh:mm:ssTZD (e.g. 1997-07-16T19:20:30+01:00)

Complete date plus hours, minutes, seconds and a decimal fraction of a second

   YYYY-MM-DDThh:mm:ss.sTZD (e.g. 1997-07-16T19:20:30.45+01:00)
```

Note that the Minutes, Seconds, Fraction, 2DIGIT and DIGIT syntax is as defined for Clock-values. Note that white space is not allowed within the date and time specification.

This section is normative

There are three ways of handling time zone offsets:

1. Times are expressed in UTC (Coordinated Universal Time), with a special UTC designator ("Z").
2. Times are expressed in local time, together with a time zone offset in hours and minutes. A time zone offset of "+hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes ahead of UTC. A time zone offset of "-hh:mm" indicates that the date/time uses a local time zone which is "hh" hours and "mm" minutes behind UTC.
3. Times are expressed in local time, as defined for the presentation location. The local time zone of the end-user platform is used.

The presentation engine must be able to convert wallclock-values to a time within the document.

- When the document begins, the current wallclock time must be noted - this is the *document wallclock begin*.
- Wallclock values are then converted to a document time by subtracting the document wallclock begin, and then converting the time to the element's parent time space as for any syncbase value, as though the syncbase were the document body.
- Date wallclock values are treated as a DateTime value of the given date at time 00:00:00.00 in the local time zone.
- WallTime values are treated as a DateTime value on the date of the *document wallclock begin* at the given time. Specified time zones must be respected, and the time converted into the local time zone before applying the *document wallclock begin.*

This section is informative

Note that the resulting begin or end time may be before the begin, or after end of the parent time container. This is not an error, but the time container constraints still apply. In any case, the semantics of the **begin** and **end** attribute govern the interpretation of the wallclock value.

EXAMPLES

The following examples all specify a begin at midnight on January 1st 2000, UTC:

```
begin="wallclock( 2000-01-01T00:00Z )"
begin="wallclock( 2000-01-01T00:00:00Z )"
begin="wallclock( 2000-01-01T00:00:00.0Z )"
begin="wallclock( 2000-01-01T00:00:00.0Z )"
begin="wallclock( 2000-01-01T00:00:00.0-00:00 )"
```

The following example specifies a begin at 3:30 in the afternoon on July 28th 1990, in the Pacific US time zone:

```
begin="wallclock( 1990-07-28T15:30-08:00 )"
```

The following example specifies a begin at 8 in the morning wherever the document is presented:

```
begin="wallclock( 08:00 )"
```

The endsync attribute

This section is normative

The **endsync** attribute controls the implicit duration of time containers, as a function of the children. The **endsync** attribute is only valid for **par** and **excl** time container elements, and media elements with timed children (e.g. **animate** or **area** elements). Integrating languages may allow the **endsync** attribute on any element with time container semantics. The **endsync** attribute is particularly useful with children that have "unknown" duration, e.g. an MPEGmovie, that must be played through to determine the duration, or elements with event-based end timing.

endsync = (`"first"` | `"last"` | `"all"` | `"media"` | *`Id-value`* | *`smil1.0-Id-value`*)

Legal values for the attribute are:

`first`

The **par**, **excl**, or media element's implicit duration ends with the earliest active end of all the child elements. This does not refer to the lexical first child, or to the first child to start, but rather refers to the first child to end its (first) active duration.

`last`
: The **par**, **excl**, or media element's implicit duration ends with the last active end of the child elements. This does not refer to the lexical last child, or to the last child to start, but rather refers to the last active end of all children that have a resolved, definite begin time. If the time container has no children with a resolved begin time, the time container ends immediately. If child elements have multiple begin times, or otherwise restart, the child elements must complete *all* instances of active durations for resolved begin times.
This is the default value for **par** and **excl** elements.

`all`
: The **par**, **excl**, or media element's implicit duration ends when all of the child elements have ended their respective active durations. Elements with indefinite or unresolved begin times *will* keep the simple duration of the time container from ending.
When all elements have completed the active duration one or more times, the parent time container can end.

`media`
: The time container element's implicit duration ends when the intrinsic media duration of the element ends. This must be defined by a host language. If the time container element does not define an intrinsic media duration, the host language must define the simple duration for the element.
This is the default value for media time container elements.

`Id-value`
: The **par**, **excl**, or media element time container's implicit duration ends when the specified child ends its (first) active duration. The id must correspond to one of the immediate timed children of the time container.

`smil1.0-Id-value`
: This is a SMIL 1.0 identifier value of the form "id(" IDREF ")". The semantics are identical to those of the Id-value immediately above. This syntax is deprecated.

- Elements may have an unresolved or indefinite begin time when the parent begins. If an element's unresolved begin time becomes resolved (and definite) before the parent time container ends the simple duration, the element must be considered by the **`endsync="last"`** semantics. This can chain, so that only one element is running at one point, but before it ends its active duration another interactive element is resolved. It may even yield "dead time" (where nothing is playing), if the resolved begin is *after* the other elements active end.
- If the **endsync** semantics consider any child that has an unresolved active duration, then the implicit duration of the time container is also unresolved.
- For the ***`Id-value`*** arg-value variant, the referenced child may have an unresolved begin time. If this causes the active end time to be unresolved as well, the implicit duration of the time container is also unresolved.
- If the **endsync** semantics consider any child that has a (resolved) indefinite active duration, then the implicit duration of the time container is also indefinite.
- Media element time containers define an intrinsic duration equal to the duration of the referenced media.If the referenced media is not continuous, the duration is 0 (**`endsync="media"`** will not generally be useful on discrete media).
- If the **`media`** argument value is used for an element that does not declare media, the attribute is ignored (as though **endsync** had not been specified).

- If the ***Id-value*** arg-value variant is not an immediate child of the time container, it is as if endsync is not specified.
- For the purpose of parsing the endsync argument value, **first**, **last**, **all**, and **media** are reserved words and must be escaped with a backslash in order to be used as **Id-value's**.

Semantics of endsync and dur and end:

- If an element specifies both endsync and dur, the endsync attribute is ignored. The element's simple duration is defined by the value of dur.
- If an element specifies both endsync and end, but none of dur, repeatDur or repeatCount, the endsync attribute is ignored. In this case the element behaves as if only end were specified, therefore the element's implicit duration is indefinite and will be constrained by the end value.

Semantics of endsync and restart:

- In the case of an element that restarts (e.g. because of multiple begin times), the element is considered to have ended its active duration when one active duration instance has completed. It is not a requirement that all instances associated with multiple begin and end times complete, to satisfy the semantics of endsync. This means that if the element is playing a second or later instance of an active duration, it may be cut short by a parent, once the other children satisfy the endsync semantics.

Semantics of endsync and paused elements:

- Note that child elements of an excl that are currently paused (by the excl semantics) have not ended their active duration. Similarly, any element paused via the DOM `pause()` method has not completed its active duration. Paused elements (that have not already completed the active duration at least once) must be considered in the evaluation ofendsync. For example, if a time container with endsync=**last** has paused child elements, the simple duration of the time container will not end until the paused children resume or otherwise end.

This section is informative

Semantics of endsync and unresolved child times:

- endsync=**"first"** means that the element must wait for any child element to actually end its active duration. It does not matter whether the first element to end was scheduled or interactive.
- endsync=**"last"** means that the element must wait for all child elements that have a resolved begin, to end the respective active durations.
  Elements with indefinite or unresolved begin times will *not* keep the simple duration of the time container from ending. If there are no children with a resolved begin time, the time container will end immediately.
  Elements with a resolved begin time but indefinite or unresolved end times *will* keep the simple duration of the time container from ending.
- endsync=**"all"** means that the element must wait for the end of every child element's active duration.
- endsync=[***Id-value***] means that the element must wait for the referenced element to actually end its active duration.

This section is normative

The following pseudo-code describes the endsync algorithm:

```
//
// boolean timeContainerHasEnded()
//
// method on time containers called to evaluate whether
// time container has ended, according to the rules of endsync.
// Note: Only supported on par and excl
//
// A variant on this could be called when a child end is updated to
// create a scheduled (predicted) end time for the container.
//
// Note that we never check the end time of children - it doesn't matter.
//
// Assumes:
//     child list is stable during evaluation
//     isActive state of children is up to date for current time.
//      [In practice, this means that the children must all be
//        pre-visited at the current time to see if they are done.
//        If the time container is done, and repeats, the children
//        may be resampled at the modified time.]
//
//   Uses interfaces:
//   on TimedNode:
//     isActive()             tests if node is currently active
//     hasStarted()           tests if node has (ever) begun
//     begin and end          begin and end TimeValues of node
//
//   on TimeValue         (a list of times for begin or end)
//   is Resolved(t)          true if there is a resolved time
//                                     at or after time t
//

boolean timeContainerHasEnded()
{

TimeInstant now = getCurrentTime(); // normalized for time container

boolean assumedResult;

// For first or ID, we assume a false result unless we find a child that has ended
// For last and all, we assume a true result unless we find a disqualifying child

if( ( endsyncRule == first ) || ( endsyncRule == ID ) )
   assumedResult = false;
else
   assumedResult = true;

// Our interpretation of endsync == all:
//          we're done when all children have begun, and none is active
//

// loop on each child in collection of timed children,
//  and consider it in terms of the endsyncRule

 foreach ( child c in timed-children-collection )
{
   switch( endsyncRule ) {
      case first:
         // as soon as we find an ended child, return true.
         if( c.hasStarted() & !c.isActive() )
            return true;
         // else, keep looking (assumedResult is false)
         break;

      case ID:
         // if we find the matching child, just return result
         if( endsyncID == c.ID )
                 return( c.hasStarted() & !c.isActive() );
         // else, keep looking (we'll assume the ID is valid)
         break;

      case last:
         // we just test for disqualifying children
         // If the child is active, we're definitely not done.
         // If the child has not yet begun but has a resolved begin,
         // then we're not done.
         if( c.isActive()
             || c.begin.isResolved(now) )
             return false;
         // else, keep checking (the assumed result is true)
         break;

      case all:
         // we just test for disqualifying children
        // all_means_last_done_after_all_begin

         // If the child is active, we're definitely not done.
         // If the child has not yet begun then we're not done.
         // Note that if it has already begun,
         // then we still have to wait for any more resolved begins
         if( c.isActive() || !c.hasStarted()
             || c.begin.isResolved(now) )
             return false;
         // else, keep checking (the assumed result is true)
         break;

   } // close switch

} // close foreach loop

return assumedResult;

} // close timeContainerHasEnded()
```

The repeatCount, repeatDur, and repeat attributes: repeating elements

This section is informative

SMIL 1.0 introduced the repeat attribute, which is used to repeat a media element or an entire time container. SMIL 2.1 introduces two new controls for repeat functionality that supercede the SMIL 1.0 repeat attribute. The new attributes, repeatCount and repeatDur, provide a semantic that more closely matches typical use-cases, and the new attributes provide more control over the duration of the repeating behavior.

Repeating an element causes the simple duration to be "played" several times in sequence. This will effectively copy or *loop* the contents of the element media (or an entire timeline in the case of a time container). The author can specify either *how many times* to repeat, using repeatCount, or *how long* to repeat, using repeatDur. Each repeat *iteration* is one instance of "playing" the simple duration.

This section is normative

repeatCount
: Specifies the number of iterations of the simple duration. It can have the following attribute values:

`numeric value`

This is a (base 10) "floating point" numeric value that specifies the number of iterations. It can include partial iterations expressed as fraction values. A fractional value describes a portion of the simple duration. Values must be greater than 0.

"indefinite"

The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

repeatDur

Specifies the total duration for repeat. It can have the following attribute values:

Clock-value

Specifies the duration in element active time to repeat the simple duration.

"indefinite"

The element is defined to repeat indefinitely (subject to the constraints of the parent time container).

- The SMIL 1.0 repeat attribute is deprecated in SMIL 2.1 (it must be supported in SMIL document user agents for backwards compatibility).
- See the Computing the Active Duration section for how repeat attributes interact with other timing attributes.

Examples

In the following example, the implicit duration of the audio is constrained by repeatCount. Only the first half of the clip will play; the active duration will be 1.5 seconds.

```
<audio src="3second_sound.au" repeatCount="0.5" />
```

In this example, the 3 second (implicit) simple duration will be played three times through and then is constrained by the dur attribute on the parent par; the active duration will be 9 seconds.

```
<par dur="9s">
   <audio src="3second_sound.au" repeatCount="100" />
</par>
```

In the following example, the 2.5 second simple duration will be repeated twice; the active duration will be 5 seconds.

```
<audio src="background.au" dur="2.5s" repeatCount="2" />
```

In the following example, the 3 second (implicit) simple duration will be repeated two full times and then the first half is repeated once more; the active duration will be 7.5 seconds.

```
<audio src="3second_sound.au" repeatCount="2.5" />
```

In the following example, the audio will repeat for a total of 7 seconds. It will play fully two times, followed by a fractional part of 2 seconds. This is equivalent to a repeatCount of 2.8.

```
<audio src="music.mp3" dur="2.5s" repeatDur="7s" />
```

Note that if the simple duration is indefinite, repeat behavior is not defined (but repeatDur still contributes to the active duration). In the following example the simple duration is 0 and indefinite respectively, and so the repeatCount is ignored. Nevertheless, this is not considered an error. The active duration is equal to the simple duration: for the first element, the active duration is 0, and for the second element, the active duration is indefinite.

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

In the following example, the simple duration is 0 for the image and indefinite for the text element, and so repeat behavior is not meaningful. The active duration is 0 for the first element, however for the second element, the active duration is determined by the repeatDur value, and so is 10 seconds. The effect is that the text is shown for 10 seconds.

```
<img src="foo.jpg" repeatDur="10s" />
<text src="intro.html" dur="indefinite" repeatDur="10s" />
```

In the following example, if the audio media is longer than the 5 second repeatDur, then the active duration will effectively cut short the simple duration.

```
<audio src="8second_sound.au" repeatDur="5s" />
```

The repeatCount and repeatDur attributes can also be used to repeat an entire timeline (i.e. a time container simple duration), as in the following 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>
```

The min attribute and restart:

The min attribute does not prevent an element from restarting before the minimum active duration is reached. If in the following example, the "user.activateEvent" occurs once at 2 seconds, then again at 5 seconds, the "image" element will begin at 2 seconds, play for 3 seconds, and then be restarted at 5 seconds. The restarted interval (beginning at 5 seconds) will display the image until 12 seconds.

```
<smil ...>
...
<par>
   <img id="image" begin="user.activateEvent" min="7s" dur="5s"
        restart="always" fill="freeze".../>
</par>
...
</smil>
```

SMIL 1.0 repeat (deprecated)

The SMIL 1.0 repeat attribute behaves in a manner similar to repeatCount, but it defines the functionality in terms of a sequence that contains the specified number of *copies* of the element without the repeat attribute. This definition has caused some confusion among authors and implementers. See also the SMIL 1.0 specification [SMIL10].

In particular, there has been confusion concerning the behavior of the SMIL 1.0 end attribute when used in conjunction with the repeat attribute. SMIL 2.1 complies with the common practice of having the end attribute define the element's simple duration when the deprecated repeat attribute is used. Only SMIL document user agents must support this semantic for the end attribute. Only a single SMIL 1.0 "end" value (i.e. an offset-value or a smil-1.0-syncbase-value, but none of the new SMIL 2.1 timing) is permitted when used with the deprecated repeat attribute. If repeat is used with repeatCount or repeatDur on an element, or if repeat is used with an illegal end value, the repeat value is ignored.

This section is normative

repeat
: *This attribute has been deprecated in SMIL 2.1 in favor of the new repeatCount and repeatDur attributes.*
  This causes the element to play repeatedly for the specified number of times. It is equivalent to a seq element with the stated number of copies of the element without the "repeat" attribute as children. All other attributes of the element, including any begin delay, are included in the copies.
  Legal values are `integer iterations, greater than 0,` and `"indefinite"`.

The fill attribute: extending an element

This section is informative

When an element's active duration ends, it may be *frozen* at the final state, or it may no longer be presented (i.e., its effect is removed from the presentation). *Freezing* an element extends it, using the final state defined in the last instance of the simple duration. This can be used to fill gaps in a presentation, or to extend an element as context in the presentation (e.g. with additive animation - see [SMIL-ANIMATION]).

This section is normative

The fill attribute allows an author to specify that an element should be extended beyond the active duration by *freezing* the final state of the element. The fill attribute is also used to determine the behavior when the active duration is less than the duration specified in the min attribute. For this reason, rather than referring to the end of the active duration, this description refers to the "last instance of the simple duration".

- For discrete media, the media is simply displayed as it would be during the simple duration.
- For visual continuous media, the "frame" that corresponds to the end of the last instance of the simple duration is shown.
- For algorithmic media like animation, the value defined for the end of the last instance of the simple duration should be used.
- For time containers, freezing extends the state of all children that are active or frozen at the end of the last instance of the time container simple duration. The children are frozen as they appear at the end of the last instance of the time container simple duration. If a child element ends its active duration coincident to the end of the last instance of its parent time container simple duration, the child element fill value determines whether the child will be frozen after the end of the parent time container's last simple duration.
- A host language integrating Timing must specify the semantics of freezing elements.

The syntax of the fill attribute is the same as in SMIL 1.0, with two extensions. In addition, the fill attribute may now be applied to any timed element, including time containers.

fill = `( "remove" | "freeze" | "hold" | "transition" | "auto" | "default" )`
: This attribute can have the following values:

`remove`
: Specifies that the element will not extend past the end of the last instance of the simple duration.

`freeze`
: Specifies that the element will extend past the end of the last instance of the simple duration by "freezing" the element state at that point. The parent time container of the element determines how long the element is frozen (as described immediately below).

`hold`
: Setting this to **`"hold"`** has the same effect as setting to **`"freeze"`**, except that the element is always frozen to extend to the *end of the simple duration of the parent time container* of the element (independent of the type of time container). For profiles that support a layered layout model (e.g., SMIL 2.1 Language Profile), held elements (elements with fill=**`"hold"`**) will refresh their display area when a layer is added on top then later removed.

`transition`
: Setting this to **`"transition"`** has the same effect as setting to **`"freeze"`**, except that the element is removed at the end of the transition. This value is only allowed on elements with media directly associated with them. If specified on any other element (e.g. a time container element in the SMIL language profile), the attribute is ignored. See the SMIL Transitions module.

`auto`
: The fill behavior for this element depends on whether the element specifies any of the attributes that define the simple or active duration:

  - If none of the attributes **dur**, **end**, **repeatCount** or **repeatDur** are specified on the element, then the element will have a fill behavior identical to that if it were specified as **`"freeze"`**.
  - Otherwise, the element will have a fill behavior identical to that if it were specified as **`"remove"`**.

`default`
: The fill behavior for the element is determined by the value of the **fillDefault** attribute.
This is the default value.
If the application of **fillDefault** to an element would result in the element having a value of fill that is not allowed on that element, the element will instead have a fill value of "auto".

Note that given the default values for **fill** and **fillDefault** attributes, if the **fill** attribute is not specified for an element, and if the **fillDefault** attribute is not specified for any ascendant of the element, the behavior uses **`"auto"`** semantics.

An element with **`"freeze" behavior`** is extended according to the parent time container:

- In a **par**, the element is frozen to extend to the end of the simple duration of the **par**. In this case, **fill**=**`"freeze"`** is equivalent to **fill**=**`"hold"`**.
- In a **seq**, the element is frozen to extend to the begin of the next element in the **seq** or until the end of the simple duration of theseq. This will fill any gap in the presentation (although it may have no effect if the next element begins immediately).
- In an **excl**, the element is frozen to extend to the begin of the next element to be activated in the **excl**, or until an element in the **excl** is resumed, or until the end of the simple duration of the **excl**. This will fill any gap in the presentation (although it may have no effect if the next element interrupts the current element). Note that if an element is paused, the active duration has not ended, and so the **fill** attribute does not (yet) apply. See also the section The **excl** element.

When applied to media, fill only has a presentation effect on visual media. Non-visual media (audio) will simply be silent (although they are still frozen from a timing perspective).

The fillDefault attribute

This section is normative

fillDefault = ("remove" | "freeze" | "hold" | "transition" | "auto" | "inherit")

Defines the default value for the fill behavior for an element and all descendents. The values "remove", "freeze", "hold", "transition" and "auto" specify that the element fill behavior is the respective value.

inherit

Specifies that the value of this attribute (and of the fill behavior) are inherited from the fillDefault value of the parent element. If there is no parent element, the value is "auto".
This is the default value.

The Event sensitivity and fill

The effects of the fill attribute apply only to the timing semantics. If an element is still visible while frozen, it behaves normally with respect to other semantics such as user event processing. In particular, elements such as **a** and **area** are still sensitive to user activation (e.g. clicks) when frozen. See also the SMIL 1.0 specification [SMIL10].

This section is informative

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

```
<par endsync="last">
   <video src="intro.mpg" begin= "5s" dur="30s" fill="freeze" />
   <audio src="intro.au"  begin= "2s" dur="40s"/>
</par>
```

The video element ends 35 seconds after the parent time container began, but the video frame at 30 seconds into the media remains displayed until the audio element ends. The attribute "freezes" the last value of the element for the remainder of the time container's simple duration.

This functionality is also useful to keep prior elements on the screen while the next item of a **seq** time container prepares to display as in this example:

```
<seq>
   <video id="v1" fill="freeze" src.../>
   <video id="v2" begin="2s" src.../>
</seq>
```

The first video is displayed and then the last frame is frozen for 2 seconds, until the next element begins. Note that if it takes additional time to download or buffer video "v2" for playback, the first video "v1" will remain frozen until video "v2" actually begins.

The restart attribute

This section is informative

Note that there are several ways that an element may be restarted. The behavior (i.e. to restart or not) in all cases is controlled by the restart attribute. The different restart cases are:

- An element with begin specified as an event-value can be restarted when the named event fires multiple times.
- An element with begin specified as a syncbase value, where the syncbase element can restart. When an element restarts, other elements defined to begin relative to the begin or active end of the restarting element may also restart (subject to the value of restart on these elements).
- An element can be restarted when the DOM "beginElement()" method is called repeatedly.

As with any begin time, if an element is scheduled to restart after the end of the parent time container simple duration, the element will not restart.

This section is normative

restart = `( "always" | "whenNotActive" | "never" | "default" )`

`always`
: The element can be restarted at any time.

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

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

`default`
: The restart behavior for the element is determined by the value of the restartDefault attribute.
This is the default value.

- When an element restarts, the primary semantic is that it behaves as though this were the first time the element had begun, independent of any earlier behavior. Any effect of an element playing earlier is no longer applied (including any fill behavior), and only the new current interval of the element is reflected in the presentation. It should be obvious that this definition applies only to the behavior of the element content, and not to the evaluation of the begin and end times lists described in Evaluation of begin and end time lists.
- When an active element restarts, the element first ends the active duration, propagates this to time dependents and raises an endEvent in the normal manner (see also Evaluation of begin and end time lists). Restart semantics are evaluated *after* the active duration for an element is computed, and so ending the active duration due to a restart is not subject to the semantics of min. See also Computing the active duration.
- The synchronization relationship between an element and its parent time container is re-established when the element restarts. A new synchronization relationship may be defined. See also Controlling runtime synchronization behavior.

- Note that if the parent time container (or any ascendant time container) repeats or restarts, any state associated with restart=**"never"** will be reset, and the element can begin again normally. See also Resetting element state.

The restartDefault attribute can be used to control the default behavior of the restart attribute. This is described below in Controlling the default behavior of restart.

For details on when and how the restart attribute is evaluated, see Evaluation of begin and end time lists.

Using restart for toggle activation

This section is informative

A common use-case requires that the same UI event is used to begin an element and to end the active duration of the element. This is sometimes described as "toggle" activation, because the UI event toggles the element "on" and "off". The restart attribute can be used to author this, as follows:

```
<smil ...>
...
<img id="foo" begin="bar.activateEvent" end="bar.activateEvent"
              restart="whenNotActive" ... />
</smil ...>
```

If "foo" were defined with the default restart behavior **"always"**, a second activateEvent on the "bar" element would simply restart the element. However, since the second activateEvent cannot restart the element when restart is set to **"whenNotActive"**, the element ignores the "begin" specification of the activateEvent event. The element can then use the activateEvent event to end the active duration and stop the element.

Note that in SMIL Language documents, a SMIL element cannot be visible before it begins so having a begin=**"activateEvent"** means it won't ever begin. In languages with `timeAction` support, this may not be the case. For example, the following is reasonable:

```
<html ...>
...
<span begin="click" end="click" timeAction="class:highlight" restart="whenNotActive">
  Click here to highlight. Click again to remove highlight.
</span>
...
</html>
```

This is based upon the event sensitivity semantics described in Event sensitivity and Unifying Scheduling and Interactive Timing.

Controlling the default behavior of restart

This section is normative

The following attribute is provided to specify the default behavior for restart:

restartDefault = ("always" | "whenNotActive" | "never" | "inherit")
Defines the default value for the restart behavior for an element.
The values "always", "whenNotActive" and "never" specify that the element restart behavior is the respective value.

`inherit`

Specifies that the value of this attribute (and of the restart behavior) are inherited from the restartDefault value of the parent element. If there is no parent element, the value is "always".
This is the default value.

Given the default values of this attribute ("inherit") and of the **restart** attribute ("default"), a document that does not specify these attributes will have `restart="always"` behavior for all timed elements.

Resetting element state

This section is normative

When a time container repeats or restarts, all descendent children are "reset" with respect to certain state:

1. Any instance times associated with past event-values, repeat-values, accesskey-values or added via DOM method calls are removed from the dependent begin and end instance times lists. In effect, all events and DOM methods calls in the past are cleared. This does not apply to an instance time that defines the begin of the current interval.
2. Any syncbase times are reevaluated (i.e. the translation between timespaces must be recalculated - see Converting between local and global times).
3. A resolved syncbase time is removed from the dependent instance time list when a common ascendant of the syncbase *and* the dependent element restarts or repeats
4. Any state associated with the interpretation of the **restart** semantics is reset.

This section is informative

Thus, for example if an element specifies **restart**=**`"never"`**, the element can begin again after a reset. The **restart**=**`"never"`** setting is only defined for the extent of the parent time container simple duration.

This section is normative

When an element restarts, rules 1 and 2 are also applied to the element itself, although rule 4 (controlling restart behavior) is not applied.

Note that when any time container ends its simple duration (including when it repeats), all timed children that are still active are ended. See also Time container constraints on child durations.

When an **excl** time container restarts or repeats, in addition to ending any active children, the pause queue for the **excl** is cleared.

The syncBehavior, syncTolerance, and syncMaster attributes: controlling runtime synchronization

This section is informative

New support in SMIL 2.1 introduces finer grained control over the runtime synchronization behavior of a document. The **syncBehavior** attribute allows an author to describe for each element whether it must remain in a hard sync relationship to the parent time container, or whether it can be allowed slip with respect to the time container. Thus, if network congestion delays or interrupts the delivery of media for an element, the syncBehavior attribute controls whether the media element can slip while the rest of the document continues to play, or whether the time container must also wait until the media delivery catches up.

The syncBehavior attribute can also be applied to time containers. This controls the sync relationship of the entire timeline defined by the time container. In this example, the audio and video elements are defined with hard or "locked" sync to maintain lip sync, but the "speech" par time container is allowed to slip:

```
<par>
   <animation src="..." />
   ...
   <par id="speech" syncBehavior="canSlip" >
      <video src="speech.mpg" syncBehavior="locked" />
      <audio src="speech.au"  syncBehavior="locked" />
   </par>
   ...
</par>
```

If either the video or audio must pause due to delivery problems, the entire "speech" par will pause, to keep the entire timeline in sync. However, the rest of the document, including the animation element will continue to play normally. Using the syncBehavior attribute on elements and time containers, the author can effectively describe the "scope" of runtime sync behavior, defining some portions of the document to play in hard sync without requiring that the entire document use hard synchronization.

This functionality also applies when an element first begins, and the media must begin to play. If the media is not yet ready (e.g. if an image file has not yet downloaded), the syncBehavior attribute controls whether the time container must wait until the element media is ready, or whether the element begin can slip until the media is downloaded.

An additional extension allows the author to specify that a particular element should define or control the synchronization for a time container. This is similar to the default behavior of many user agents that "slave" video and other elements to audio, to accommodate the audio hardware inaccuracies and the sensitivity of listeners to interruptions in the audio playback. The syncMaster attribute allows an author to explicitly define that an element defines the playback "clock" for the time container, and all other elements should be held in sync relative to the syncMaster element.

In practice, linear media often need to be the syncMaster, where non-linear media can more easily be adjusted to maintain hard sync. However, a user agent cannot always determine which media behaves in a linear fashion and which media behaves in a non-linear fashion. In addition, when there are multiple linear elements active at a given point in time, the user agent cannot always make the "right" decision to resolve sync conflicts. The syncMaster attribute allows the author to specify the element that has linear media, or that is "most important" and should not be compromised by the syncBehavior of other elements.

This section is normative

syncBehavior = (`"canSlip"` | `"locked"` | `"independent"` | `"default"`)

Defines the runtime synchronization behavior for an element.
Legal values are:

`canSlip`

Allows the associated element to slip with respect to the parent time container.
When this value is used, any syncTolerance attribute is ignored.

`locked`
: Forces the associated element to maintain sync with respect to the parent time container. This can be eased with the use of the syncTolerance attribute.

`independent`
: Declares an independent timeline that is scheduled with the timegraph, but will ignore any seek operations on the parent.

`default`
: The runtime synchronization behavior for the element is determined by the value of the syncBehaviorDefault attribute.
This is the default value.

The argument value `independent` is equivalent to setting syncBehavior="`canSlip`" and syncMaster="`true`" so that the element is scheduled within the timegraph, but is unaffected by any other runtime synchronization issues. Setting syncBehavior="`canSlip`" and syncMaster=`"true"` declares the element as being the synchronization master clock and that the element may slip against its parent time line

syncTolerance **=** **(** `Clock-value | "default"` **)**
: This attribute on timed elements and time containers defines the synchronization tolerance for the associated element . The attribute has an effect only if the element's runtime synchronization behavior is `"locked"`. This allows a locked sync relationship to ignore a given amount of slew without forcing resynchronization.

`Clock-value`
: Specifies the synchronization tolerance as a value. Clock values are measured in element simple time.

`default`
: The synchronization tolerance for the element is determined by the value of the syncToleranceDefault attribute.
This is the default value.

syncMaster
: Boolean attribute on media elements and time containers that forces other elements in the time container to synchronize their playback to this element.
The default value is `false`.
The associated property is read-only, and cannot be set by script.

- The syncBehavior can affect the effective begin and effective end of an element, but the use of the syncBehavior attribute does not introduce any other semantics with respect to duration.
- When the syncBehavior attribute is combined with interactive begin timing or restarting an element, the syncBehavior only applies once the sync relationship of the element is resolved (e.g. when the specified event is raised). If at that point the media is not ready and syncBehavior is specified as `"locked"`, then the parent time container must wait until the media is ready. Once an element with an interactive begin time has begun playing, the syncBehavior semantics described above apply as though the element were defined with scheduled timing.

- The syncBehavior attribute is subordinate to any sync relationships defined by time containers, sync arcs, event arcs, restart behavior, etc. The syncBehavior attribute has no bearing on the formation of the time graph, only the enforcement of it.
- The syncMaster attribute interacts with the syncBehaviorattribute. An element with syncMaster set to true will define sync for the "scope" of the time container's synchronizationbehavior. That is, if the syncMaster element's parent time container has syncBehavior=`"locked"`, the syncMaster will also define sync for the ancestor timeContainer. The syncMaster will define sync for everything within the closest ancestor time container that is defined with syncBehavior=`"canSlip"`.
- The syncMaster attribute only applies when an element is active and not paused. If more than one element within the syncBehavior scope has the syncMaster attribute set to `"true"`, and the elements are both active and not paused at any moment in time, the sync master element is the first of these elements encountered in a post order traversal of the document tree as defined by DOM [DOM2]. In this conflict case, the other elements effectively ignore the syncMaster attribute.
- When an element is paused, the semantics of the syncMaster attribute are effectively ignored. Nevertheless, any accumulated synchronization offset associated with syncMaster semantics (that is an offset accumulated while the sync master element was not paused) is not changed when the sync master element pauses. The offset in this case will be the offset for the closest ancestor time container that is defined with syncBehavior=`"canSlip". See also` The accumulated synchronization offset.

This section is informative

Note that the semantics of syncBehavior do not describe or require a particular approach to maintaining sync; the approach will be implementation dependent. Possible means of resolving a sync conflict may include:

- Pausing the parent time container (i.e. first ancestor time container with `canSlip` behavior) until the element that slipped can "catch up".
- Pausing the element that is playing too fast until the parent (document) time container catches up.
- Seeking (i.e. resetting the current position of) the element that slipped, jumping it ahead so that it "catches up" with the parent time container. This would only apply to non-linear media types.

Additional control is provided over the hard sync model using the syncTolerance attribute. This specifies the amount of slip that can be ignored for an element. Small variance in media playback (e.g. due to hardware inaccuracies) can often be ignored, and allow the overall performance to appear smoother.

When any element is paused (including the cases described above for runtime sync behavior), the computed end time for the element may change or even become resolved, and the time model must reflect this. This is detailed in Paused elements and the active duration.

Controlling the default behavior

Two attributes are defined to specify the default behavior for runtime synchronization:

syncBehaviorDefault = (`"canSlip"` | `"locked"` | `"independent"` | `"inherit"`)

Defines the default value for the runtime synchronization behavior for an element.
The values "canSlip", "locked" and "independent" specify that the element's runtime synchronization behavior is the respective value.

`inherit`

Specifies that the value of this attribute (and the value of the element's runtime synchronization behavior) are inherited from the **syncBehaviorDefault** value of the parent element. If there is no parent element, the value is implementation dependent.
This is the default value.

syncToleranceDefault = (`Clock-value` | `"inherit"`)

Defines the default value for the runtime synchronization tolerance value for an element.
Clock values specify that the element's runtime synchronization tolerance value is the respective value.

`inherit`

Specifies that the value of this attribute (and the value of the element's runtime synchronization tolerance value) are inherited from the **syncToleranceDefault** value of the parent element. If there is no parent element, the value is implementation dependent but should be no greater than two seconds.
This is the default value.

The accumulated synchronization offset

If an element slips synchronization relative to its parent, the amount of this slip at any point is described as the *accumulated synchronization offset*. This offset is used to account for pause semantics as well as performance or delivery related slip. This value is used to adjust the conversion between element and parent times, as described in Converting between local and global times. The offset is computed as follows:

Let $\mathbf{t_c(t_{ps})}$ be the computed element active time for an element at the parent simple time $\mathbf{t_{ps}}$, according to the defined synchronization relationship for the element.

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

The accumulated synchronization offset **O** is:

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

This offset is measured in parent simple time.

Thus an accumulated synchronization offset of 1 second corresponds to the element playing 1 second "later" than it was scheduled. An offset of -0.5 seconds corresponds to the element playing a half second "ahead" of where it should be.

Attributes for timing integration: timeContainer and timeAction

The modularization of SMIL 2.1 functionality allows language designers to integrate SMIL Timing and Synchronization support into any XML language. In addition to just scheduling media elements as in SMIL language documents, timing can be applied to the elements of the host language. For example, the addition of timing to HTML (i.e. XHTML) elements will control the presentation of the HTML document over time, and to synchronize text and presentation with continuous media such as audio and video.

Two attributes are introduced to support these integration cases. The **timeContainer** attribute allows the author to specify that any XML language element has time container behavior. E.g., an HTML `<ol>` ordered list element can be defined to behave as a sequence time container. The **timeAction** attribute allows the author to specify what it means to apply timing to a given element.

The timeContainer attribute

XML language elements can be declared to have time container semantics by adding the **timeContainer** attribute. The syntax is:

```
timeContainer = ( "par" | "seq" | "excl" | "none" )
```

par
: Defines a parallel time container.

seq
: Defines a sequence time container.

excl
: Defines an exclusive time container.

none
: Defines the current element to *not* have time container behavior (i.e. to behave as a simple time leaf).
This is the default.

Constraints upon the use of the **timeContainer** attribute are:

- Language designers may restrict the set of elements that can be time containers, but any element can in principle be a time container.
- An element with the **timeContainer** attribute behaves the same as a media time container. If the element has no intrinsic media duration, then the element can behave the same as the respective time container element. Language designers must specify which elements have intrinsic media duration, and how this is defined.
- The **timeContainer** attribute *may not* be applied to any of the SMIL time container elements **par**, **seq** or **excl**.
- The **timeContainer** attribute *may* be applied to SMIL media elements (with timed children) to control the behavior of a media time container.

The timeAction attribute

The timeAction attribute provides control over the effect of timing upon an attribute. A host language must specify which values are allowed for each element in the language. A host language must specify the *intrinsic* timing behavior of each element to which timeAction may be applied. In addition, a host language may specify additional timeAction values. The syntax is:

> timeAction **= ("intrinsic" | "display" | "visibility" | "style" | "class" | "none")**

`intrinsic`
: Specifies that timing controls the intrinsic behavior of the element. **This is the default.**

`display`
: Specifies that timing controls the display of the element, as defined by CSS. The timing of the element can affect the presentation layout. For languages that incorporate CSS, the CSS "`display`" property should be controlled over time.

`visibility`
: Specifies that timing controls the visibility of the element, as defined by CSS. The timing of the element should not affect the presentation layout. For languages that incorporate CSS, the CSS "`visibility`" property should be controlled over time.

`style`
: Specifies that timing controls the application of style defined by an inline "`style`" attribute.

`class:`*`classname`*
: Specifies that timing controls the inclusion of the specified *class-name* in the set of classes associated with the element (i.e. the XML `class` attribute value list).

`none`
: Specifies that timing has no effect upon the presentation of the element.

The **`intrinsic`** behavior is defined by a host language. For example in the SMIL language, the **`intrinsic`** behavior of media elements is to schedule and control the visibility of the media. For some elements or some languages, the **`intrinsic`** behavior may default to one of the other behaviors.

Additional timeAction semantics and constraints:

- SMIL media elements define the **`intrinsic`** behavior as the scheduling and playback of the media. When timeAction is set to any other value (besides **`intrinsic`**), the **`intrinsic`** scheduling behavior will be controlled *in addition to* the specified timeAction.
- SMIL time container elements (**par**, **seq**, and **excl**) define the **`intrinsic`** behavior as simply performing the scheduling semantics. When the **`intrinsic`** behavior is specified, only the scheduling semantics are controlled. Timing does not otherwise control the presentation styling of the element. When timeAction is set to any other value (besides **`intrinsic`**), the **`intrinsic`** scheduling behavior will be controlled *in addition to* the specified timeAction.
- It is recommended that phrasal, presentation, and style-like elements (e.g., XHTML's **b**, **em**, **strong**, ...etc.) have an **`intrinsic`** behavior that applies the associated semantic (e.g. to *embolden* or *emphasize* the content of the

element). When timeAction is set to any other value (besides **intrinsic**), the **intrinsic** behavior will be controlled *in addition to* the specified timeAction.
- It is recommended that "content" elements (e.g., XHTML's **p**, **div**, **span**) have an intrinsic behavior equivalent to **visibility**. When timeAction is set to any other value (besides **intrinsic**), the **intrinsic** behavior should *not* be controlled, but rather only the specified timeAction should be applied.
- If a language supports CSS styling, the presentation effect of **display** and **visibility** should use the CSS override style, rather than setting the original value for the associated properties. This model for presentation values is described in the SMIL Animation module.
- If a language supports CSS styling, the `visibility` property should be set to "**hidden**" when the element is not active or frozen. If the original value of the `visibility` property was not "**hidden**", the original value should be used when the element is active or frozen. If the original value of the `visibility` property was "**hidden**", the property should be set to "**visible**" when the element is active or frozen.
- If a language does not support CSS styling, the presentation semantics of **display** and **visibility** must be specified, or the attribute values must be disallowed.
- If a language does not support a "`style`" attribute, the **style** value for timeAction should not be allowed.
- When the **class** argument value is specified, the specified class name should be *added to* the class list of the element when the element is active or frozen. Other values in the class list must be preserved. If the specified class name is specified in the class list (i.e. if it is specified in the `class` attribute), the class name should be removed from the class list of the element when the element is not active or frozen.
- When the **none** argument value is specified, no action is controlled by the timing, except any intrinsic behavior. This value is generally only useful on an element that also specifies a timeContainer value. In this case, the time containment semantic applies, but no presentation effect is applied to the element.

Certain special elements may have specific **intrinsic** semantics. For example, linking elements like **a** and **area** can have an **intrinsic** behavior that controls the sensitivity of the elements to actuation by the user. This may have presentation side-effects as well. In XHTML for example, making these elements insensitive also has the effect that the default styling (e.g. a color and underline) that is applied to sensitive links is removed when the element is not active or frozen.

Host language designers should carefully consider and define the behavior associated with applying timing to an element. For example, `script` elements could be defined to execute when the element begins, or the language could disallow the timeAction attribute on the element. Similarly, `link` elements could apply a linked stylesheet when the element begins or the language could disallow the timeAction attribute on `link`.

For details of the CSS properties `visibility` and `display`, see [CSS2].

Examples:

The following example shows a simple case of controlling visibility over time. The text is hidden from 0 to 3 seconds, shown normally for 5 seconds, and then hidden again.

```
<span timeAction="visibility" begin="3s" dur="5s">
   Show this text for a short period.
```

```
</span>
```

The following example shows a simple case of controlling display over time. Each list element is shown for 5 seconds, and is removed from the layout when not active or frozen. The ordered list element is set to be a sequence time container as well (note that each list element retains its ordinal number even though the others are not displayed):

```
<ol timeContainer="seq" repeatDur="indefinite">
   <li timeAction="display" dur="5s">
      This is the first thing you will see. </li>
   <li timeAction="display" dur="5s">
      You will see this second. </li>
   <li timeAction="display" dur="5s">
      Last but not least, you will see this. </li>
</ol>
```

The following example shows how an element specific style can be applied over time. The respective style is applied to each HTML `label` for 5 seconds after a focus event is raised on the element:

```
<form ...>
...
   <label for="select_red" begin="focus" dur="5s" timeAction="style"
          style="color:red; font-weight:bold" >
      Make things RED.
   </label>
   <input id="select_red" .../>
   <label for="select_green" begin="focus" dur="5s" timeAction="style"
          style="color:green; font-weight:bold" >
      Make things GREEN.
   </label>
   <input id="select_green" .../>
...
</form>
```

10.4.2 Elements

This section is informative

SMIL 2.1 specifies three types of time containers. These can be declared with the elements **par**, **seq**, and **excl**, or in some integration profiles with a **timeContainer** attribute. Media elements with timed children are defined to be "media time containers", and have semantics based upon the **par** semantics (see also Attributes for timing integration: timeContainer and timeAction and Implicit duration of media element time containers).

This document refers in general to time containers by reference to the elements, but the same semantics apply when declared with an attribute, and for media time containers.

The par element

This section is normative

par

A **par** container, short for "parallel", defines a simple time grouping in which multiple elements can play back at the same time.

The implicit syncbase of the child elements of a **par** is the begin of the **par**. The default value of **begin** for children of a **par** is `"0"`. This is the same element introduced with SMIL 1.0.

The **par** element supports all element timing.

Implicit duration of ***par***

The implicit duration of a **par** is controlled by **endsync**. By default, the implicit duration of a **par** is defined by the **endsync**`="last"` semantics. The implicit duration ends with the last active end of the child elements.

The seq element

This section is normative

seq

A **seq** container defines a sequence of elements in which elements play one after the other.

This is the same element introduced with SMIL 1.0, but the semantics (and allowed syntax) for child elements of a **seq** are clarified.

- The implicit syncbase of the child elements of a **seq** is the active end of the *previous* element. *Previous* means the element that occurs before this element in the sequence time container. For the first child of a sequence (i.e. where no previous sibling exists), the implicit syncbase is the begin of the sequence time container.
- For children of a sequence, the only legal value for begin is a (single) non-negative offset value.
  The default begin value is "0". None of the following begin values may be used:

  disallowed begin-values:
  ```
  ( syncbase-value | event-value | media-marker-value |
  wallclock-sync-value | repeat-value | accesskey-value |
  "indefinite" )
  ```
- Note however that child elements *may* define an end that references other syncbases, event-bases, etc.

The **seq** element itself supports all element timing except **endsync**.

When a hyperlink traversal targets a child of a **seq**, and the target child is not currently active, part of the seek action must be to enforce the basic semantic of a **seq** that only one child may be active at a given time. For details, see Hyperlinks and timing and specifically Implications of beginElement() and hyperlinking for **seq** and **excl** time containers.

*Implicit duration of **seq** containers*

- The implicit duration of a **seq** ends with the active end of the last child of the **seq**.
- If any child of a **seq** has an indefinite active duration, the implicit duration of the **seq** is also indefinite.

The excl element

SMIL 2.1 defines a time container, **excl**, that allows the interactive (or a-temporal) activation of child elements.

This section is normative

excl
: This defines a time container with semantics based upon **par**, but with the additional constraint that only one child element may play at any given time. If any element begins playing while another is already playing, the element that was playing is stopped. If the **priorityClass** element is also supported by a profile, child elements in an **excl** container may be grouped into categories; the behavior of the element that was playing at the time a new element starts can be defined to have stop/pause/interruption behavior. The **priorityClass** can define several levels of interrupt behavior (one per class), each of which can be controlled explicitly.

The implicit syncbase of the child elements of the **excl** is the **begin** of the **excl**. The default value of **begin** for children of **excl** is `"indefinite"`. This means that the **excl** has 0 duration unless a child of the **excl** has been added to the timegraph.

The **excl** element itself supports all element timing.

This section is informative

With the **excl** time container, common use cases that were either difficult, or impossible, to author are now easier and possible to create. The **excl** time container is used to define a mutually exclusive set of clips, and to describe pausing and resuming behaviors among these clips. Examples include:

interactive playlist
: A selection of media clips is available for the user to choose from, only one of which plays at a time. A new selection replaces the current selection.

audio descriptions
: For visually impaired users, the current video is paused and audio descriptions of the current scene are played. The video resumes when the audio description completes.

interactive video sub-titles
: Multiple language sub-titles are available for a video. Only one language version can be shown at a time with the most recent selection replacing the previous language choice, if any.

The interactive playlist use case above could be accomplished using a **par** whose sources have interactive begin times and **end** events for all other sources. This would require a prohibitively long list of values for **end** to maintain. The **excl** time container provides a convenient short hand for this - the element begin times are still interactive, but the end events do not need to be specified because the excl, by definition, only allows one child element to play at a time.

The audio descriptions use case is not possible without the pause/resume behavior provided by excl and priorityClass. This use case would be authored with a video and each audio description as children of the excl. The video element would be scheduled to begin when the excl begins and the audio descriptions, peers of the video element, would start at scheduled begin times or in response to stream events raised at specific times.

The dynamic video sub-titles use case requires the "play only one at a time" behavior of excl. In addition, the child elements are declared in such a way so as to preserve the sync relationship to the video:

```
<smil ...>
...
<par endsync="vid1">
   <video id="vid1" .../>
   <excl dur="indefinite">
      <par begin="englishBtn.activateEvent" >
         <audio begin="vid1.begin" src="english.au" />
      </par>
      <par begin="frenchBtn.activateEvent" >
         <audio begin="vid1.begin" src="french.au" />
      </par>
      <par begin="swahiliBtn.activateEvent" >
         <audio begin="vid1.begin" src="swahili.au" />
      </par>
   </excl>
</par>
...
</smil>
```

The three par elements are children of the excl, and so only one can play at a time. The audio child in each par is defined to begin when the video begins. Each audio can only be active when the parent time container (par) is active, but the begin still specifies the synchronization relationship. This means that when each par begins, the audio will start playing at some point in the middle of the audio clip, and in sync with the video.

The excl time container is useful in many authoring scenarios by providing a declarative means of describing complex clip interactions.

Implicit duration of excl containers

This section is normative

- The implicit duration of an excl container is defined the same as for a par container, using the endsync=**"last"** semantics. However, since the default timing for children of excl is interactive, the implicit duration for excl time containers with only default timing on the children will be 0.

The priorityClass element

This section is informative

Using priority classes to control the pausing behavior of children of the **excl** allows the author to group content into categories of content, and then to describe rules for how each category will interrupt or be interrupted by other categories. Attributes of the new grouping element **priorityClass** describe the intended interactions.

Each **priorityClass** element describes a group of children, and the behavior of those children when interrupted by other time-children of the **excl**. The behavior is described in terms of *peers*, and *higher* and *lower* priority elements. *Peers* are those elements within the same **priorityClass** element.

When one element within the **excl** begins (or would normally begin) while another is already active, several behaviors may result. The active element may be paused or stopped, or the interrupting element may be deferred, or simply blocked from beginning.

The careful choice of defaults makes common use cases very simple. See the examples below.

This section is normative

priorityClass

Defines a group of **excl** time-children, and the pause/interrupt behavior of the children. If a **priorityClass** element appears as the child of an **excl**, then the **excl** can only contain **priorityClass** elements (i.e. the author cannot mix timed children and **priorityClass** elements within an **excl**).

If no **priorityClass** element is used, all the children of the **excl** are considered to be *peers*, with the default **peers** behavior "stop".

- **priorityClass** elements may only appear as immediate children of **excl** elements and cannot be nested.
- Only elements allowed as immediate children of excl may appear as immediate children of **priorityClass** elements.
- The **priorityClass** element is transparent to timing, and does not participate in or otherwise affect the normal timing behavior of its children (i.e. it only defines how elements interrupt one another).
- Child elements of the **priorityClass** elements are time-children of the **excl** element (i.e. the parent **excl** of the **priorityClass** elements).
- The **priorityClass** elements are assigned priority levels based upon the order in which they are declared within the **excl**. The first **priorityClass** element has highest priority, and the last has lowest priority.
- When elements are paused or deferred, they are added to a queue of pending elements. When an active element completes its active duration, the first element (if any) in the queue of pending elements is madeactive. The queue is ordered according to rules described in Pause queue semantics.

The peers, higher, and lower attributes

This section is informative

Note that the rules define the behavior of the currently active element and the interrupting element. Any elements in the pause queue are not affected (except that their position in the queue may be altered by new queue insertions).

This section is normative

peers = (`"stop" | "pause" | "defer" | "never"`)

Controls how child elements of this **priorityClass** will interrupt one another. Legal values for the attribute are:

`stop`

If a child element begins while another child element is active, the active element is simply stopped.
This is the default for peers.

`pause`

If a child element begins while another child element is active, the active element is paused and will resume when the new (interrupting) element completes its active duration (subject to the constraints of the **excl** time container). The paused element is added to the pause queue.

`defer`

If a child element attempts to (i.e. would normally) begin while another child element is active, the new (interrupting) element is deferred until the active element completes its active duration.

`never`

If a child element attempts to (i.e. would normally) begin while another child element is active, the new (interrupting) element is prevented from beginning. The begin of the new (interrupting) element is ignored.

higher = (`"stop" | "pause"`)

Controls how elements with higher priority will interrupt child elements of this **priorityClass**.
Legal values for the attribute are:

`stop`

If a higher priority element begins while a child element of this **priorityClass** is active, the active child element is simply stopped.

`pause`

If a higher priority element begins while a child element of this **priorityClass** is active, the active child element is paused and will resume when the new (interrupting) element completes its active duration (subject to the constraints of the `excl` time container). The paused element is added to the pause queue.
This is the default for the higher attribute.

lower = (`"defer" | "never"`)

Controls how elements defined with lower priority will interrupt child elements of this **priorityClass**.
Legal values for the attribute are:

`defer`

If a lower priority element attempts to (would normally) begin while a child element of this **priorityClass** is active, the new (interrupting) element is deferred until the active element completes its active duration. The rules for adding the element to the queue are described below.
This is the default for the lower attribute.

`never`

If a lower priority element attempts to begin while a child element of this **priorityClass** is active, the new (interrupting) element is prevented from beginning. The begin of the new (interrupting) element is ignored, and it is not added to the queue.

When an element begin is blocked (ignored) because of the `"never"` attribute value, the blocked element does not begin in the time model. The time model should not propagate begin or end activations to time dependents, nor should it raise begin or end events.

The pauseDisplay attribute

This section is informative

The pauseDisplay attribute controls the *behavior when paused* of the children of a priorityClass element. When a child of a priorityClass element is paused according to excl and priorityClass semantics, the pauseDisplay attribute controls whether the paused element will continue to *show* or *apply* the element (i.e. the state of the element for the time at which it is paused), or whether it is removed altogether from the presentation (i.e. *disabled*) while paused.

This section is normative

pauseDisplay = (`"disable" | "hide" | "show"`)
Controls how child elements of the priorityClass element behave when paused. This attribute only applies if peers=`"pause"` or higher=`"pause"`. Legal values for the attribute are:

`disable`
Continue to display visual media when the element is paused by the excl and priorityClass, but appear disabled. It is implementation dependent how a disabled element appears (rendered in some different way to distinguish from the active state -- e.g., grayed out); disabled elements do not respond to mouse events.

`hide`
Remove the effect of the element (including any rendering) when the element is paused by the excl and priorityClass semantics.

`show`
Continue to show the effect of the element (including any rendering) when the element is paused by the excl and priorityClass semantics. This value has no effect on a aural media.
This is the default.

Examples using excl and priorityClass

This section is informative

Note that because of the defaults, the simple cases work without any additional syntax. In the basic case, all the elements default to be peers, and stop one another:

```
<excl dur="indefinite">
   <audio id="song1" .../>
   <audio id="song2" .../>
   <audio id="song3" .../>
   ...
   <audio id="songN" .../>
</excl>
```

is equivalent to the following with explicit settings:

```
<excl dur="indefinite">
   <priorityClass peers="stop">
     <audio id="song1" .../>
     <audio id="song2" .../>
     <audio id="song3" .../>
     ...
     <audio id="songN" .../>
   </priorityClass>
</excl>
```

If the author wants elements to pause rather than stop, the syntax is:

```
<excl dur="indefinite">
   <priorityClass peers="pause">
     <audio id="song1" .../>
     <audio id="song2" .../>
     <audio id="song3" .../>
     ...
     <audio id="songN" .../>
   </priorityClass>
</excl>
```

The audio description use case for visually impaired users would look very similar to the previous example:

```
<excl dur="indefinite">
   <priorityClass peers="pause">
     <video id="main_video" .../>
     <audio id="scene1_description" begin="20s"  dur="30s".../>
     <audio id="scene2_description" begin="2min" dur="30s" .../>
     ...
     <audio id="sceneN_description" .../>
   </priorityClass>
</excl>
```

This example shows a more complex case of program material and several commercial insertions. The program videos will interrupt one another. The ads will pause the program, but will not interrupt one another.

```
<excl dur="indefinite">
   <priorityClass id="ads" peers="defer">
     <video id="advert1" .../>
     <video id="advert2" .../>
   </priorityClass>
   <priorityClass id="program" peers="stop" higher="pause">
     <video id="program1" .../>
     <video id="program2" .../>
     <video id="program3" .../>
     <video id="program4" .../>
   </priorityClass>
</excl>
```

The following example illustrates how defer semantics and priority groups can interact. When "alert1" tries to begin at 5 seconds, the "program" **priorityClass** will force "alert1" to defer, and so "alert1" will be placed upon the queue. When "alert2" tries to begin at 6 seconds, the same semantics will force "alert2" onto the queue. Note that although the "alerts" **priorityClass** defines the **peers** rule as "never", "alert1" is not active at 6 seconds, and so the interrupt semantics between "alert1" and "alert2" are not evaluated. The resulting behavior is that when "prog1" ends at 20 seconds, "alert1" will play, and then when "alert1" ends, "alert2" will play.

```
<excl dur="indefinite">
   <priorityClass id="program" lower="defer">
     <video id="prog1" begin="0" dur="20s" .../>
   </priorityClass>
   <priorityClass id="alerts" peers="never">
     <video id="alert1" begin="5s" .../>
     <video id="alert2" begin="6s" .../>
   </priorityClass>
</excl>
```

This example illustrates pauseDisplay control. When an element is interrupted by a peer, the interrupted element pauses and is shown in a disabled state. It is implementation dependent how the disabled video is rendered. Disabled elements do not respond to mouse events.

```
<excl dur="indefinite">
   <priorityClass peers="pause" pauseDisplay="disable">
      <video id="video1" .../>
      <video id="video2" .../>
      <video id="video3" .../>
      ...
      <video id="videoN" .../>
   </priorityClass>
</excl>
```

In this example, when a child of a higher **priorityClass** element interrupts a child of the "program" **priorityClass**, the child of "program" pauses and remains onscreen. If a peer of the "program" **priorityClass** interrupts a peer, the element that was playing stops and is no longer displayed.

```
<excl dur="indefinite">
   <priorityClass id="ads" peers="defer">
      <video id="advert1" .../>
      <video id="advert2" .../>
   </priorityClass>
   <priorityClass id="program" peers="stop" higher="pause" pauseDisplay="show">
      <video id="program1" .../>
      <video id="program2" .../>
      <video id="program3" .../>
      <video id="program4" .../>
   </priorityClass>
</excl>
```

Pause queue semantics

This section is normative

Elements that are paused or deferred are placed in a priority-sorted queue of waiting elements. When an active element ends its active duration and the queue is not empty, the first (i.e. highest priority) element in the queue is *pulled* from the queue and resumed or activated.

The queue semantics are described as a set of invariants and the rules for insertion and removal of elements. For the purposes of discussion, the child elements of a **priorityClass** element are considered to have the priority of that **priorityClass**, and to have the behavior described by the **peers**, **higher** and **lower** attributes on the **priorityClass** parent.

Queue invariants

1. The queue is sorted by priority, with higher priority elements before lower priority elements.
2. An element may not appear in the queue more than once.
3. An element may not simultaneously be active and in the queue.

Element insertion and removal

1. Elements are inserted into the queue sorted by priority (by invariant 1).
    a. Paused elements are inserted *before* elements with the same priority.
    b. Deferred elements are inserted *after* elements with the same priority.
2. Where the semantics define that an active element must be paused, the element is paused at the current simple time (position) when placed on the queue. When a paused element is pulled normally from the queue, it will resume from the point at which it was paused.
3. Where the semantics define that an element must be deferred, the element is inserted in the queue, *but is not begun*. When the element is pulled normally from the queue, it will begin (i.e. be activated).
4. When an element is placed in the queue any previous instance of that element is removed from the queue (by invariant 2).
5. When the active child (i.e. time-child) of an **excl** ends normally (i.e. not when it is *stopped* by another, interrupting element), the element on the front of the queue is pulled off the queue, and resumed or begun (according to rule 2 or 3).

Note that if an element is active and restarts (subject to the **restart** rule), it does not interrupt itself in the sense of a peer interrupting it. Rather, it simply restarts and the queue is unaffected.

Runtime synchronization behavior and pause/defer semantics

The runtime synchronization behavior of an element (described in the **syncBehavior**, **syncTolerance**, and **syncMaster** attributes: controlling runtime synchronization) does not affect the queue semantics. Any element that is paused or deferred according to the queue semantics will behave as described. When a paused element is resumed, the synchronization relationship will be reestablished according to the runtime synchronization semantics. The synchronization relationship for a deferred element will be established when the element actually begins.

Calculated times and pause/defer semantics

When an element is paused, the calculated end time for the element may change or even become resolved, and the time model must reflect this. This is detailed in Paused elements and the active duration. In some cases, the end time is defined by other elements unaffected by the pause queue semantics. In the following example, the "foo" element will be paused at 8 seconds, but it will still end at 10 seconds (while it is paused):

```
<img "joe" end="10s" .../>
<excl dur="indefinite">
   <priorityClass peers="pause">
      <img id="foo" end="joe.end" .../>
      <img id="bar" begin="8s" dur="5s" .../>
   </priorityClass>
</excl>
```

If an element ends while it is in the pause queue, it is simply removed from the pause queue. All time dependents will be notified normally, and the end event will be raised at the end time, as usual.

When an element is deferred, the begin time is deferred as well. Just as described in Paused elements and the active duration, the begin time of a deferred element may become unresolved, or it may simply be delayed. In the following example, the "bar" element will initially have an unresolved begin time. If the user clicks on "foo" at 8 seconds, "bar" would resolve to 8 seconds, but will be deferred until 10 seconds (when "foo" ends):

```
<html ...>
...
<excl dur="indefinite">
   <priorityClass peers="defer">
      <img id="foo" begin="0s" dur="10s" .../>
      <img id="bar" begin="foo.click" .../>
   </priorityClass>
</excl>
...
</html>
```

If there is enough information to determine the new begin time (as in the example above), an implementation must compute the correct begin time when an element is deferred. The change to the begin time that results from the element being paused must be propagated to any sync arc time dependents (i.e. other elements with a begin or end defined relative to the begin of the deferred element). See also the Propagating changes to times section.

One exception to normal processing is made for deferred elements, to simplify the model: a deferred element ignores propagated changes to its begin time. This is detailed in the Deferred elements and propagating changes to begin section.

Scheduled begin times and excl

Although the default begin value for children of an **excl** is indefinite, scheduled begin times are permitted. Scheduled begin times on children of the **excl** cause the element to begin at the specified time, pausing or stopping other siblings depending on the **priorityClass** settings (and default values).

Handling Simultaneous Begins within excl

If children of an **excl** attempt to begin at the same time, the evaluation proceeds in document order. For each element in turn, the priorityClass semantics are considered, and elements may be paused, deferred or stopped.

This section is informative

The following examples both exhibit this behavior (it can result from any combination of scheduled times, interactive timing, hyperlink or DOM activation):

```
<smil ...>
...
<excl>
   <img src="image1.jpg" begin="0s" dur="5s"/>
   <img src="image2.jpg" begin="0s" dur="5s"/>
   <img src="image3.jpg" begin="0s" dur="5s"/>
</excl>

<excl dur ="indefinite">
   <img id="img1" src="image1.jpg" begin="foo.activateEvent" dur="5s"/>
   <img id="img2" src="image2.jpg" begin="img1.begin" dur="5s"/>
   <img id="img3" src="image3.jpg" begin="img2.begin" dur="5s"/>
</excl>
...
</smil>
```

In the first example, the images are scheduled to begin immediately, where in the second, they will all begin once the user activates the "foo" element. The end result of the two (other than the begin time) is the same. Given the default interrupt semantics for **excl**, the first image will begin and then be immediately stopped by the second image, which will in turn be immediately stopped by the third image. The net result is that only the third image is seen, and it lasts for 5 seconds. Note that the begin and end events for the first two images are raised and propagated to all time dependents. If the behavior is set to **"pause"** as in this example, the declared order is effectively reversed:

```
<excl>
   <priorityClass peers="pause">
      <img src="image1.jpg" begin="0s" dur="5s"/>
      <img src="image2.jpg" begin="0s" dur="5s"/>
      <img src="image3.jpg" begin="0s" dur="5s"/>
   </priorityClass>
</excl>
```

In this case, the first image will begin and then be immediately paused by the second image, which will in turn be immediately paused by the third image. The net result is that the third image is seen for 5 seconds, followed by the second image for 5 seconds, followed by the first image for 5 seconds. Note that the begin events for the first two images are raised and propagated to all time dependents when the **excl** begins.

In the following slideshow example, images begin at the earlier of their scheduled begin time or when activated by a user input event:

```
<html ...>
...
<excl>
   <img src="image1.jpg" begin="0s".../>
   <img src="image2.jpg" begin="10s; image1.click".../>
   <img src="image3.jpg" begin="20s; image2.click".../>
</excl>
...
</html>
```

Note, some surprising results may occur when combining scheduled and interactive timing within an **excl**. If in the above example, the user clicks on image1 and then on image2 before ten seconds have elapsed, image 2 will re-appear at the ten second mark. Image 3 will appear at twenty seconds. The likely intent of this particular use-case would be better represented with a **seq** time container.

Side effects of activation

This section is informative

Children of the **excl** can be activated by scheduled timing, hyperlinks, events or DOM methods calls. For all but hyperlink activation, the **excl** time container must be active for child elements of the **excl** to be activated. With hyperlink activation, the document may be seeked to force the parent **excl** to be active, and a seek may occur to the begin time target child if it has a resolved begin time. That is, the normal hyperlink seek semantics apply to a timed child of an **excl**.

This section is normative

With activation via a DOM method call (e.g. the `beginElement()` method), the element will be activated at the current time (subject to the **priorityClass** semantics), even if the element has a scheduled begin time. The exclusive semantics of the time container (allowing only one active element at a time) and all **priorityClass** semantics are respected nevertheless.

See also Hyperlinks and timing and specifically Implications of beginElement() and hyperlinking for **seq** and **excl** time containers.

Implicit duration of media element time containers

This section is normative

The implicit duration of a media time container combines the intrinsic duration of the media with the children to define the implicit simple duration. For the **"*ID-REF*"** value of endsync, the semantics are the same as for a normal time container. For the **"media"** value of endsync, implicit simple duration is equal to the intrinsic duration of the media directly associated with the element. For the values **"first"**, **"last"** and **"all"**, the media element acts as a **par** time container, but treats the element's associated media as an additional condition as far as determining when the criteria for **"first"**, **"last"** and **"all"** endsync values have been satisfied.

- For **endsync**=**{ "media" or "*ID-REF*"}**: This is defined as for **par** elements.
- For **endsync**=**{"last" or "all"}**: The time children and the intrinsic media duration of the associated media define the implicit duration of the media element time container. If the associated media duration is longer than the extent of all the time children, the media duration defines the implicit duration for the media element time container. If the associated media is discrete, this is defined as for **par** elements.
- For **endsync**=**"first"**: The time children and the intrinsic media duration define the implicit duration of the media element time container. The element ends when the first active duration ends, as defined above for **endsync** on a **par**. If the media is discrete, this is defined as for **par** elements.

If the implicit duration defined by **endsync** is *longer* than the intrinsic duration for a continuous media element, the ending state of the media (e.g. the last frame of video) will be shown for the remainder of the implicit duration. This only applies to visual media - aural media will simply stop playing.

This section is informative

This semantic is similar to the case in which the author specifies a simple duration that is longer than the intrinsic duration for a continuous media element. Note that for both cases, although the media element is effectively frozen for the remainder of the simple duration, the time container simple time is not frozen during this period, and any children will run normally without being affected by the media intrinsic duration.

Examples:

Assume that "vid1" is 10 seconds long in the following examples.

The default value of **endsync** for media elements is "media", and so the simple duration in the following example is 10 seconds. This will cut short the **animate** child 8 seconds into its simple duration:

```
<video src="vid1.mpg" >
   <animate begin="2s" dur="12s" .../>
</video>
```

Specifying **endsync**=**"first"** in the example below causes the simple duration of the video element to be 10 seconds, since the media finishes before the animate child.

```
<video src="vid1.mpg" endsync="first" >
   <animate begin="2s" dur="12s" .../>
</video>
```

Specifying **endsync**=**"last"** in the following example causes the simple duration of the video element to be 14 seconds. The video will show a still frame (the last frame) for the last 4 seconds of this:

```
<video src="vid1.mpg" endsync="last" >
   <set dur="8s" .../>
   <animate begin="2s" dur="12s" .../>
</video>
```

Specifying **endsync**=**"all"** in the following example causes the simple duration of the video element to last at least 10 seconds (the intrinsic duration of the video), and at most until 5 seconds after the user clicks on the video. The video will show a still frame (the last frame) for any duration in excess of 10 seconds:

```
<html ...>
...
<video src="vid1.mpg" endsync="all" >
   <set dur="8s" .../>
   <animate begin="click" dur="5s" .../>
</video>
...
</html>
```

Thus if the user clicks on the video after 1 second, the simple duration is 10 seconds. If the user does not click until 15 seconds, the simple duration is 20 seconds, and the last frame will be shown between 10 and 20 seconds. The video can still be clicked even though it stops normal play at 10 seconds.

Media time containers of other types

In some language integrations, it will be possible to declare a media time container to have sequence or exclusive semantics, in addition to the default parallel semantics described above. For example:

```
<html ...>
...
<video src="vid1.mpg" timeContainer="seq" endsync="first" >
   <animate dur="4s" .../>
   <animate end="click" .../>
</video>
...
</html>
```

The animate children of the **video** will act in sequence. The **endsync** semantics define a simple duration for the **video** that is no more than 10 seconds (the intrinsic duration of the video) but may be just over 4 seconds, if the user clicks on the **video** as soon as the last **animate** begins.

10.4.3 Semantics of the Timing Model

Except as noted, this entire section is normative

Resolving times

A begin or end time is said to be unresolved when either an associated begin or end event has not yet occurred (within the constraints of Event sensitivity), or the begin or end time is dependent upon another element's begin or end time that is unresolved. The begin or end time becomes resolved as soon as the syncbase element's time is resolved, or when the event occurs (within the constraints of Event sensitivity).

If a begin or end value resolves to a time in the past, this value is propagated to other synchronization dependents. Similarly, a simple or active duration can be unresolved but can become resolved when end conditions are met or the parent time container constrains the element's duration.

Definite times

A resolved time is said to be *definite* if it is not the value "indefinite".

Defining the simple duration

The *simple duration* of an element is determined by the **dur** attribute, the implicit duration of the element, and one special-case rule to ensure SMIL 1.0 backward compatibility. Apply the first rule in the table that matches the given criteria.

Computation of the simple duration is based on the information available at the time the calculation is made. Unresolved quantities may require the simple duration to be recomputed when an unresolved quantity becomes resolved.

dur			Simple Duration

	implicit element duration	repeatDur and repeatCount	
unspecified	(ignored)	unspecified, end specified	`indefinite`
Clock-value	(ignored)	(ignored)	dur or Clock-value
`indefinite`	(ignored)	(ignored)	`indefinite`
unspecified	resolved	(ignored)	implicit element duration or Clock-value
unspecified	unresolved	(ignored)	unresolved
media	resolved or unresolved	(ignored)	implicit element duration

Simple Duration Table

repeatCount and unresolved simple duration

When repeatCount is specified, it is understood to represent a count of iterations of simple duration. Each iteration of the simple duration may be different, and so a simple multiplication of the repeatCount and a given simple duration may not yield an accurate active duration. In the case of a partial repeatCount and a simple duration that is not resolved, the most recent simple duration should be multiplied by the fractional part of the repeatCount to constrain the last simple duration. If the last iteration of the simple duration otherwise ends before this time, the repeatCount should be considered to be complete. If a repeatCount is less than 1 and the simple duration is unresolved, the repeatCount cannot be correctly respected, and will behave as though a repeatCount of "1" were specified.

This section is informative

If an element specifying audio media has a simple duration of 0 (e.g., because of `clipBegin` and `clipEnd` values), nothing should be played even if the repeatDur specifies an active duration. The time model behaves according to the description, but no audio should be played.

If a repeatDur is shorter than the simple duration, or if repeatCount is less than 1, the active duration can cut short the defined simple duration.

If repeatDur is "indefinite" and neither of repeatCount or end are specified, the active duration is indefinite. If repeatCount is indefinite, the simple duration is greater than 0 and neither of repeatDur or end are specified, then the active duration is indefinite.

Note that unlike in SMIL 1, when an element defines a begin offset and repeat behavior with repeatCount or repeatDur, the begin offset is *not included* in each repeat.

Computing the active duration

The *active duration* of an element defines the entire period that an element's timeline is active. It takes into account the element *simple duration* evaluated above, the **end** attribute, and any repeat behavior defined by the **repeatDur** and **repeatCount** attributes.

Active duration arithmetic rules

Computing the active duration requires defining arithmetic operations on all of the possible values that simple duration can have.

Multiplication

- zero value * value = zero value
- zero value * indefinite = zero value
- non-zero value * non-zero value = non-zero value
- non-zero value * indefinite = indefinite
- indefinite * indefinite = indefinite
- unresolved * *anything* = unresolved

Addition and Subtraction

- value +/- value = value
- indefinite +/- value = indefinite
- value +/- indefinite = indefinite
- indefinite +/- indefinite = indefinite
- unresolved +/- *anything* = unresolved
- *anything* +/- unresolved = unresolved

Minimization Function

- MIN(zero value, anything) = zero value
- MIN(non-zero value, non-zero value) = non-zero value
- MIN(non-zero value, indefinite) = non-zero value
- MIN(non-zero value, unresolved) = non-zero value
- MIN(indefinite, unresolved) = indefinite

Where *anything* means zero value, non-zero value, indefinite, or unresolved.

Maximization Function

- MAX(numeric value A, numeric value B) = B if B > A, otherwise A
- MAX(numeric value, indefinite) = indefinite
- MAX(numeric value, unresolved) = unresolved

- MAX(indefinite, unresolved) = unresolved

Active duration algorithm

This section is informative

In this section, references to begin and end values should be understood as the current effective values in each respective value list. These values are determined by the rules described in Evaluation of begin and end time lists.

This section is normative

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

B
: The begin of an element.

d
: The simple duration of an element.

PAD
: The preliminary active duration of an element, before accounting for min and max semantics.

AD
: The active duration of an element.

Computation of the active duration is based on the information available at the time the calculation is made. Unresolved quantities may require the active duration to be recomputed when an unresolved quantity becomes resolved.

To compute the active duration, use the following algorithm:

If end is specified, and none of dur, repeatDur, and repeatCount are specified, then the simple duration is `indefinite` from the simple duration table above, and the active duration is defined by the end value, according to the following cases:

> If end is resolved to a value, then **PAD** = end - **B**,
>
> else, if end is `indefinite,` then **PAD** = `indefinite`,
>
> else, if end is unresolved, then **PAD** is unresolved, and needs to be recomputed when more information becomes available.

Else, if no end value is specified, or the end value is specified as `indefinite`, then the active duration is determined from the *Intermediate Active Duration* computation given below:

> **PAD** = *Result from Intermediate Active Duration Computation*

Otherwise, an end value not equal to `indefinite` is specified along with at least one of dur, repeatDur, and repeatCount. Then the **PAD** is the minimum of the result from the *Intermediate Active Duration Computation* given below and duration between end and the element begin:

> **PAD** = MIN(*Result from Intermediate Active Duration Computation,* end - **B**)

Finally, the computed active duration **AD** is obtained by applying min and max semantics to the preliminary active duration **PAD**. In the following expression, if there is no min value, substitute a value of 0, and if there is no max value, substitute a value of "indefinite":

AD = MIN(max, MAX(min, **PAD**))

Intermediate Active Duration Computation

We define three intermediate quantities, p0, p1, and p2, and produce an intermediate result, the *Intermediate Active Duration* (**IAD**) to be used in the computation above.

p0 is the simple duration from the Simple Duration Table, given above.

If repeatCount is not specified, **p1** has the value `indefinite`. Otherwise, **p1** is the accumulated sum of the specified number of simple durations of the iterations of this element. **p1** will have a value of unresolved until the simple duration for each iteration is resolved. Partial iterations will contribute the specified fraction of the simple duration to the sum. This product can be based on either the known fixed simple duration of the media, or if unknown, the simple duration from the previous iteration of the current set of repetitions. In general for media without a fixed simple duration, **p1** will not be resolved until the specified integral number of simple durations has passed.

p2 is the value of repeatDur. If repeatDur is unspecified, then **p2** will have a value of `indefinite`.

Then **IAD** is given by:

If **p0** equals 0, then

IAD = **0**

Else if repeatDur *and* repeatCount are unspecified then:

IAD = **p0**

else:

IAD = MIN(**p1**, **p2**, `indefinite`)

This section is informative

As an example, if an element specifies:

```
<smil ...>
...
<audio dur="5s" end="foo.activateEvent" .../>
...
</smil>
```

The active duration is initially defined as 5 seconds, based upon the specified simple duration. If the user activates "foo" before 5 seconds, the end value becomes resolved and the active duration is re-evaluated. This causes the element to end at the time of the activation.

Some of the rules and results that are implicit in the algorithm, and that should be noted in particular are:

- It is possible to have an indefinite simple duration and a defined, finite active duration, or a simple duration that is greater than the active duration. In these cases, the active duration will *constrain* (cut short) the simple duration, but the active duration does not re-define the simple duration, or change its value.
- If the begin time for an element is not resolved, it may not be possible to compute the simple or active duration.

It is possible to combine scheduled and interactive timing. For example:

```
<smil ...>
...
<par dur="30s">
   <img id="mutebutton" src="mute.jpg"/>
   <text  src="description.html" />
   <audio src="audio.au" end="mutebutton.activateEvent"/>
</par>
...
</smil>
```

The image and the text appear for the specified duration of the **par** (30 seconds). The active duration of the audio is initially defined to be indefinite because its end time is unresolved. The audio will stop early if the image is activated (e.g., clicked) before the implicit end of the audio. If the image is not activated, the **dur** attribute on the parent time container will constrain playback.

It is possible to declare both a scheduled duration, as well as an event-based active end. This facilitates what are sometimes called "lazy interaction" use-cases, such as a slideshow that will advance in response to user clicks, or on its own after a specified amount of time:

```
<html ...>
...
<seq>
   <img src="slide1.jpg" dur="10s" end="click" />
   <img src="slide2.jpg" dur="10s" end="click" />
   <img src="slide3.jpg" dur="10s" end="click" />
   <!-- etc., etc. -->
</seq>
...
</html>
```

In this case, the active end of each element is defined to be the earlier of the specified duration, or a click on the element. This lets the viewer sit back and watch, or advance the slides at a faster pace.

Paused elements and the active duration

An element can be paused while it is active. This may happen in a number of ways, including via a DOM method call or because of excl semantics. When an element is paused, a resolved end time for the element may change, or it may become unresolved. The synchronization relationship between the paused element and its parent time container is re-established when the paused element is resumed. If for example the element below is paused with a DOM method call, there is no way to know when the element will end, and so the end time must be considered unresolved:

```
<img dur="30s" .../>
```

However, in the following case, the "bar" element will still end at 10 seconds, even if it is paused at 8 seconds. In this case, the end time does not change:

```
<img id="foo" dur="10s" .../>
<img id="bar" end="foo.end" .../>
```

Finally, in the following case the "foo" element will initially be computed to end at 10 seconds. If the "bar" element begins (i.e. if the user activates or clicks on "foo"), at 8 seconds, "foo" will be paused. However, since the duration of "bar" is known, and the semantics of the **excl** pause queue are well defined, the end of "foo" can be computed to be 15 seconds:

```
<smil ...>
...
<excl dur="indefinite">
   <priorityClass peers="pause">
      <img id="foo" dur="10s" .../>
      <img id="bar" begin="foo.activateEvent" dur="5s" .../>
   </priorityClass>
</excl>
...
</smil>
```

If there is enough information to determine the new end time (as in the example above), an implementation must compute the correct end time when an element is paused. Any change to the end time that results from the element being paused must be propagated to any sync arc time dependents (i.e. other elements with a begin or end defined relative to the active end of the paused element). See also the Propagating changes to times section.

In addition, when an element is paused, the accumulated synchronization offset will increase to reflect the altered sync relationship. See also The accumulated synchronization offset.

Finally, when an element is paused it may end because the parent time container ends. In this case, any fill behavior is interpreted using the element active time when the element ends (that is, it will use the element active time at which it was paused to determine what to display).

Evaluation of begin and end time lists

This section is informative

Children of par and excl time containers can have multiple begin and end values. We need to specify the semantics associated with multiple begin and end times, and how a dynamic timegraph model works with these multiple times.

The model is based around the idea of *intervals* for each element. An interval is defined by a begin and an end time. As the timegraph is played, more than one interval may be created for an element with multiple begin and end times. At any given moment, there is one *current interval* associated with each element. Intervals are created by evaluating a list of begin times and a list of end times, each of which is based upon the *conditions* described in the begin and end attributes for the element.

The list of begin times and the list of end times used to calculate new intervals are referred to as lists of "instance times". Each instance time in one of the lists is associated with the specification of a begin or end condition defined in the attribute syntax. Some conditions - for example offset-values - only have a single instance in the list. Other conditions may have multiple instances if the condition can happen more than once. For example a syncbase-value can have multiple instance times if the *syncbase* element has played several intervals, and an event-value may have multiple instance times if the event has happened more than once.

The instance times lists for each element are initialized when the timegraph is initialized, and exist for the entire life of the timegraph. Some instance times such as those defined by offset-values remain in the lists forever, while others may come and go. For example, times associated with event-values are only added when the associated event happens, and are removed when the element *resets,* as described in Resetting element state. Similarly, Instance times for syncbase-values are added to the list each time a new interval is created for the syncbase element, but these instance times are not removed by a reset, and remain in the list.

When the timegraph is initialized, each element attempts to create a first current interval. The begin time will generally be resolved, but the end time may often be unresolved. If the element can restart while active, the current interval can end (early) at the next begin time. This interval will play, and then when it ends, the element will review the lists of begin and end instance times. If the element should play again, another interval will be created and this new interval becomes the *current interval*. The history of an element can be thought of as a set of intervals.

Because the begin and end times may depend on other times that can change, the current interval is subject to change, over time. For example, if any of the instance times for the *end* changes while the current interval is playing, the current interval end will be recomputed and may change. Nevertheless, once a time has *happened*, it is fixed. That is, once the current interval has begun, its begin time can no longer change, and once the current interval has ended, its end time can no longer change. For an element to restart, it must end the current interval and then create a new current interval to effect the restart.

When a begin or end condition defines a time dependency to another element (e.g. with a syncbase-value), the time dependency is generally thought of as a relationship between the two elements. This level of dependency is important to the model when an element creates a new current interval. However, for the purposes of propagating changes to individual times, time dependencies are more specifically a dependency from a given *interval of the syncbase element* to a particular *instance time* in one of the dependent element's instance time lists. Since only the current interval's begin and end times can change, only the current interval will generate time-change notices and propagate these to the dependent instance times.

When this section refers to the begin and end times for an element, the times are described as being in the space of the *parent simple duration*. All sync-arcs, event arcs, wallclock values, etc. must be converted to this time space for easy comparison. This is especially important when referring to begin times "before 0", which assumes that "0" is the beginning of the parent simple duration. The model does not depend upon this definition - e.g. an implementation could do everything in global document time.

Cycles in the timegraph must be detected and broken to ensure reasonable functioning of the implementation. A model for how to do this in the general case is described (it is actually an issue that applies even to SMIL 1.0). A mechanism to support certain useful cyclic dependencies falls out of the model.

The rest of this section details the semantics of the instance times lists, the element life cycle, and the mechanisms for handling dependency relationships and cycles.

The instance times lists

This section is normative

Instance lists are associated with each element, and exist for the duration of the document (i.e. there is no *life cycle* for instance lists). Instance lists may change, and some times may be added and removed, but the begin and end instance times lists are persistent.

Each element can have a begin attribute that defines one or more conditions that can begin the element. In addition, the timing model describes a set of rules for determining the end of the element, including the effects of an end attribute that can have multiple conditions. In order to calculate the times that should be used for a given interval of the element, we must convert the begin times and the end times into parent simple time, sort each list of times (independently), and then find an appropriate pair of times to define an interval.

The instance times can be resolved or unresolved. In the case of the end list, an additional special value "indefinite" is allowed. The lists are maintained in sorted order, with "indefinite" sorting after all other resolved times, and unresolved times sorting to the end.

For begin, the list interpretation is straightforward, since begin times are based only upon the conditions in the attribute or upon the default begin value if there is no attribute. However, when a begin condition is a syncbase-value, the syncbase element may have multiple intervals, and we must account for this in the list of begin times associated with the conditions.

For end, the case is somewhat more complex, since the end conditions are only one part of the calculation of the end of the active duration. The instance times list for end are used together with the other SMIL Timing semantics to calculate the actual end time for an interval.

If an instance time was defined as syncbase-values, the instance time will maintain a time dependency relationship to the associated interval for the syncbase element. This means that if the associated begin or end time of the syncbase current interval changes, then the dependent instance time for this element will change as well.

When an element creates a new interval, it notifies time dependents and provides the begin and end times that were calculated according to the semantics described in "Computing the active duration". Each dependent element will create a new instance time tied to (i.e. with a dependency relationship to) the new syncbase current interval.

Building the instance times lists

The translation of begin or end conditions to instance times depends upon the type of condition:

- **offset-values** are the simplest. Each offset-value condition yields a single instance time. This time remains in the list forever, and is unaffected by reset of the element, or by repeat or restart of the parent (or other ascendants).
- **wallclock-sync-values** are similar to offset values. Each wallclock-sync-value condition yields a single instance time. This time remains in the list forever, however each time an ascendant restarts or repeats, these times are *reconverted* from the wallclock time space to the new parent simple time space.
- **event-values, accesskey-values and repeat-**values are all treated similarly. These conditions do not yield an instance time unless and until the associated event happens. Each time the event happens, the condition yields a single instance time. The event time plus or minus any offset is *added* to the list. If the event happens multiple times during a parent simple duration, there may be multiple instance times in the list associated with the event condition. However, an important distinction is that event times are cleared from the list each time the element is reset (see also Resetting element state). Within this section, these three value types are referred to collectively as *event value conditions*.
- **syncbase-values and media-marker-values** are treated similarly. These conditions do not yield an instance time unless and until the associated syncbase element creates an interval. Each time the syncbase element creates a new interval, the condition yields a single instance time. The time plus or minus any offset is *added* to the list. Unlike event times, syncbase times are *not* cleared from the element's lists simply because the element is reset. Instead, a resolved syncbase time is removed from the list when a common ascendant of the syncbase *and* the dependent element restarts or repeats. Also each time an ascendant restarts or repeats, the remaining syncbase times are re-converted from the syncbase time space to the new parent simple time space, since syncbase times are always relative to the current parent simple time space (see also Resetting element state). Within this section, these three value types are referred to collectively as *syncbase value conditions*.
- The special value **"indefinite"** does not yield an instance time in the begin list. It will, however yield a single instance with the value "indefinite" in an end list. This value is not removed by a reset.

If no attribute is present, the default begin values must be evaluated. For children of par, this is equivalent to an offset-value of 0, and yields one persistent instance value. For children of excl, this is equivalent to "indefinite", and so does not yield an instance value.

If a DOM method call is made to begin or end the element (`beginElement()`, `beginElementAt()`, `endElement()` or `endElementAt()`), each method call creates a single instance time (in the appropriate instance times list). These time instances are cleared upon reset just as for event times. See Resetting element state.

When a new time instance is added to the begin list, the current interval will evaluate restart semantics and may ignore the new time or it may end the current interval (this is detailed in Interaction with restart semantics). In contrast, when an instance time in the begin list changes because the syncbase (current interval) time moves, this does not invoke restart semantics, but may change the current begin time: If the current interval has not yet begun, a change to an instance time in the begin list will cause a re-evaluation of the begin instance lists, which may cause the interval begin time to change. If the interval begin time changes, a *time-change* notice must be propagated to all dependents, and the current interval end must also be re-evaluated.

When a new instance time is added to the end list, or when an instance time in the end list changes, the current interval will re-evaluate its end time. If it changes, it must notify dependents.

If an element has already played all intervals, there may be no current interval. In this case, additions to either list of instance times, as well as changes to any instance time in either list cause the element to re-evaluate the lists just as it would at the end of each interval (as described in End of an interval below). This may or may not lead to the creation of a new interval for the element.

When times are added to the instance times lists, they may or may not be resolved. If they are resolved, they will be converted to parent simple time. If an instance time changes from unresolved to resolved, it will be similarly converted.

There is a difference between an unresolved instance time, and a begin or end condition that has no associated instance. If, for example, an event value condition is specified in the end attribute, but no such event has happened, there will be no associated instance time in the end list. However, if a syncbase value condition is specified for end, and if the syncbase element has a current interval, there will be an associated instance time in the end list. Since the syncbase value condition can be relative to the end of the syncbase element, and since the end of the syncbase current interval may not be resolved, the associated instance time in the end list can be unresolved. Once the syncbase current interval actually ends, the dependent instance time in the end list will get a time-change notification for the resolved syncbase interval end. The dependent instance time will convert the newly resolved syncbase time to a resolved time in parent simple time. If the instance lists did not include the unresolved instance times, some additional mechanism would have to be defined to add the end instance time when the syncbase element's current interval actually ended, and resolved its end time.

The list of resolved times includes historical times defined relative to sync base elements, and so can grow over time if the sync base has many intervals. Implementations may filter the list of times as an optimization, so long as it does not affect the semantics defined herein.

Principles for building and pruning intervals

This section is informative

The following set of principles underlie the interval model. This is not a complete model - it is just meant provide an additional view of the model.

First we define the terms *pruning* and *cutting off* an interval - these concepts should not be confused.

In some cases, after an interval has been created, it must later be *pruned* (deleted/removed from the timegraph) as more information becomes known and semantic constraints must be applied. When an interval is *pruned*, it will not be shown, it will not raise begin or end events, and any associated instance times for syncbase time dependents must be removed from the respective instance times lists. It is as though the *pruned* interval had not been specified.

In other cases, especially related to negative begin times on parent time containers, a valid interval for a child may not be shown, even though it is otherwise legal with respect to the parent time constraints. For example:

```
<par begin="-10s" dur="20s">
   <img id="slide1" src="slide1.jpg" dur="3s" />
   <img id="slide2" src="slide2.jpg" begin="slide1.end+3s" dur="10s" />
   <img id="note1" src="note1.jpg" begin="slide1.beginEvent" dur="20s" />
</par>
```

The "slide1" image will be *cut off*, but is not *pruned*. It is *cut off* because the par could not have been started 10s before its parent time container, and instead will be started at 0s into its parent time synced at 10s into its simple duration. The "slide1" image begins and ends before 10s into the par, and so cannot be shown and is *cut off*, Intervals that are *cut off* are not shown and do not raise begin or end events, but still create valid instance times for any syncbase time dependents. Thus, "slide2" *will* be shown (the interval is from minus 4 seconds to 6 seconds, document time, and so will be shown for 6 seconds, from 0 seconds to 6 seconds), but "note1" will not be shown.

The principles underlying the interval life cycle model are:

1. Try to build the current interval as early as possible.
   A. The "next" interval can be computed no earlier than the end of the current interval.
2. Do not change any interval time that is in the past. Do not prune an interval that has already begun. Note that this refers to **intervals** and not **instance times**.
3. When building an interval from a set of instance times, if the duration is resolved and negative, reject the interval; do not propagate the interval to time dependents.
   A. When the current interval has not yet begun, if the interval times change such that the duration is negative, prune the interval.
4. When building an interval from a set of instance times, if the end is resolved and is <= 0 (in parent simple time), reject the interval; do not propagate the interval to time dependents.
   A. When the current interval has not yet begun, if the interval times change such that the end is <= 0, prune the interval.
5. When building an interval from a set of instance times, if the interval begin is >= the (resolved) simple end of the parent time container, reject the interval.
   A. When the current interval has not yet begun, if the interval times change such that the begin is >= the parent time container simple end, prune the interval.
   B. When the current interval has not yet begun, if the parent simple end time changes such that the current interval begin is >= the parent time container simple end, prune the interval.

An implication of principle 5 is that we will get no intervals with **unresolved** begin times, since these will necessarily compare >= the parent simple end.

Element life-cycle

This section is normative

The life cycle of an element can be thought of as the following basic steps:

1. Startup - getting the first interval
2. Waiting to begin the current interval
3. Active time - playing an interval
4. End of an interval - compute the next one and notify dependents
5. Post active - perform any fill and wait for any next interval

Steps 2 to 5 can loop for as many intervals as are defined before the end of the parent simple duration. At any time during step 2, the begin time for the current interval can change, and at any time during steps 2 or 3, the end time for the current interval can change. When either happens, the changes are propagated to time dependents.

When the document and the associated timegraph are initialized, the instance lists are empty. The simple offset values and any "indefinite" value in an end attribute can be added to the respective lists as part of initialization, as they are independent of the begin time of parent simple time.

When an element has played all allowed instances, it can be thought of as stuck in step 5. However any changes to the instance lists during this period cause the element to jump back to step 4 and consider the creation of a new current interval.

Startup - getting the first interval

An element life cycle begins with the beginning of the simple duration for the element's parent time container. That is, each time the parent time container (or more generally *any* ascendant time container) repeats or restarts, the element resets (see also Resetting element state) and starts "life" anew.

Three things are important about the beginning of the life-cycle:

1. Any and all resolved times defined as event-values, repeat-values, accesskey-values or added via DOM method calls are cleared.
2. Any and all resolved times defined as syncbase-values, wallclock-sync-values or media-marker-values must be reconverted from the syncbase time space to the parent simple time space.
3. The first current interval is computed.

Action 1) is also described in Resetting element state. This action also happens each time the element restarts, although in that case the element must not clear an event time that defined the current begin of the interval.

Action 2) Simply updates values to reflect the current sync relationship of the parent simple duration to the rest of the document.

The third action requires some special consideration of the lists of times, but is still relatively straightforward. It is similar to, but not the same as the action that applies when the element ends (this is described in End of an interval). The basic idea is to find the first interval for the element, and make that the current interval. However, the model should handle three edge cases:

> The element can begin before the parent simple begin time (i.e. before 0 in parent simple time), and so appears to begin part way into the local timeline

(somewhat like a clipBegin effect on a media element). The model must handle begin times before the parent begin.
The element has one or more intervals defined that begin *and end* before the parent simple begin (before 0). These are filtered out of the model.
The element has one or more intervals defined that begin after the parent simple end. These are filtered out of the model. Note that if the parent simple end is unresolved, any resolved begin time happens before the parent simple end.

Thus the strict definition of the first acceptable interval for the element is the first interval that ends after the parent simple begin, and begins before the parent simple end. Here is some pseudo-code to get the first interval for an element. It assumes an abstract type "Time" that supports a compare function. It can be a resolved numeric value, the special value INDEFINITE (only used with end), and it can be the special value UNRESOLVED. Indefinite compares "greater than" all resolved values, and UNRESOLVED is "greater than" both resolved values and INDEFINITE. The code uses the instance times lists associated with the begin and end attributes, as described in the previous section.

```
// Utility function that returns true if the end attribute specification
// includes conditions that describe event-values, repeat-values or accesskey-values.
boolean endHasEventConditions();

// Calculates the first acceptable interval for an element
// Returns:
//    Interval if there is such an interval
//    FAILURE if there is no such interval
Interval getFirstInterval()
{
Time beginAfter=-INFINITY;

while( TRUE ) // loop till return
{
   If (currentInterval.end > currentInterval.begin)

       Set tempBegin = the first value in the begin list that is >= beginAfter.

   Else

       Set tempBegin = the first value in the begin list that is > beginAfter.


   If there is no such value  // No interval
      return FAILURE;

   If tempBegin >= parentSimpleEnd // Can't begin after parent ends
      return FAILURE;

   If there was no end attribute specified
      // this calculates the active end with no end constraint
      tempEnd = calcActiveEnd( tempBegin );
   else
   {
      // We have a begin value - get an end
      Set tempEnd = the first value in the end list that is >= tempBegin.
      // Allow for non-0-duration interval that begins immediately
      // after a 0-duration interval.
      If tempEnd == tempBegin && tempEnd has already been used in
        an interval calculated in this method call
      {
         set tempEnd to the next value in the end list that is > tempEnd
      }
      If there is no such value
      {
         // Events leave the end open-ended. If there are other conditions
         // that have not yet generated instances, they must be unresolved.
         if endHasEventConditions()
            OR if the instance list is empty
            tempEnd = UNRESOLVED;
         // if all ends are before the begin, bad interval
         else
            return FAILURE;
      }
      // this calculates the active dur with an end constraint
      tempEnd = calcActiveEnd( tempBegin, tempEnd );
   }

   // We have an end - is it after the parent simple begin?
   // Handle the zero duration intervals at the parent begin time as a special case

   if( tempEnd > 0 || (tempBegin==0 && tempEnd==0))
      return( Interval( tempBegin, tempEnd ) );

   else
      // Change beginAfter to find next interval, and loop
      beginAfter = tempEnd;

} // close while loop

} // close getFirstInterval
```

Note that while we might consider the case of `restart=always` separately from `restart=whenNotActive`, it would just be busy work since we need to find an interval that begins *after* `tempEnd`.

If the model yields no first interval for the element, it will never begin, and so there is nothing more to do at this point. However if there is a valid interval, the element must notify all time dependents that there is a *new interval* of the element. This is a notice from this element to all elements that are direct time dependents. This is distinct from the propagation of a changed time.

When a dependent element gets a "new interval" notice, this includes a reference to the new interval. The new interval will generally have a resolved begin time and may have a resolved end time. An associated instance time will be added to the begin or end instance time list for the dependent element, and this new instance time will maintain a time dependency relationship to the syncbase interval.

Waiting to begin the interval

This period only occurs if the current interval does not begin immediately when (or before) it is created. While an interval is waiting to begin, any changes to syncbase element current interval times will be propagated to the instance lists and may result in a change to the current interval.

If the element receives a "new interval" notice while it is waiting to begin, it will *add* the associated time (i.e. the begin or end time of the syncbase interval) to the appropriate list of resolved times.

When an instance time changes, or when a new instance time is added to one of the lists, the element will re-evaluate the begin or end time of the current interval (using the same algorithm described in the previous section). If this re-evaluation yields a changed interval, time change notice(s) will be sent to the associated dependents.

It is possible during this stage that the begin and end times could change such that the interval would never begin (e.g. the interval end is before the interval begin). In this case, the interval must be *pruned* and all dependent instance times must be removed from the respective instance lists of dependent elements. These changes to the instance lists will cause re-evaluation of the dependent element current intervals, in the same manner as a changed instance time does.

One exception to normal processing is made for elements that are *deferred* according to **excl** interrupt semantics: a deferred element ignores propagated changes to its begin time. This is detailed in the Deferred elements and propagating changes to begin section.

Active time - playing an interval

This period occurs when the current interval is active (i.e. once it has begun, and until it has ended). During this period, the end time of the interval can change, but the begin time cannot. If any of the instance times in the begin list change after the current interval has begun, the change will not affect the current interval. This is different from the case of *adding* a new instance time to the begin list, which *can* cause a restart.

If the element receives a "new interval" notice while it is active, it will *add* the associated time (i.e. the begin or end time of the syncbase interval) to the appropriate list of resolved times. If the new interval adds a time to the begin list, restart semantics are considered, and this may end the current interval.

If restart is set to "always", then the current interval will end early if there is an instance time in the begin list that is before (i.e. earlier than) the defined end for the current interval. Ending in this manner will also send a changed time notice to all time dependents for the current interval end. See also Interaction with restart semantics.

End of an interval

If an element specifies `restart="never"` then no further action is taken at the end of the interval, and the element sits in the "post interval" state unless and until an ascendant time container repeats or restarts.

If an element specifies other values for `restart`, when it ends the current interval the element must reconsider the lists of resolved begin and end times. If there is another legal interval defined to begin at or after the just completed end time, a new interval will be created. When a new interval is created it becomes the *current interval* and a new interval notice is sent to all time dependents.

The algorithm used is very similar to that used in step 1, except that we are interested in finding an interval that begins after the most recent end.

```
// Calculates the next acceptable interval for an element
// Returns:
//    Interval if there is such an interval
//    FAILURE if there is no such interval
```

```
Interval getNextInterval()
{
// Note that at this point, the just ended interval is still the "current interval"
Time beginAfter=currentInterval.end;

   Set tempBegin = the first value in the begin list that is >= beginAfter.
   If there is no such value  // No interval
      return FAILURE;

   If tempBegin >= parentSimpleEnd // Can't begin after parent ends
      return FAILURE;

   If there was no end attribute specified
      // this calculates the active end with no end constraint
      tempEnd = calcActiveEnd( tempBegin );
   else
   {
      // We have a begin value - get an end
      Set tempEnd = the first value in the end list that is >= tempBegin.
      // Allow for non-0-duration interval that begins immediately
      // after a 0-duration interval.
      If tempEnd == currentInterval.end
      {
         set tempEnd to the next value in the end list that is > tempEnd
      }
      If there is no such value
      {
         // Events leave the end open-ended. If there are other conditions
         // that have not yet generated instances, they must be unresolved.
         if endHasEventConditions()
            OR if the instance list is empty
            tempEnd = UNRESOLVED;
         // if all ends are before the begin, bad interval
         else
            return FAILURE;
      }
      // this calculates the active dur with an end constraint
      tempEnd = calcActiveEnd( tempBegin, tempEnd );
   }

   return( Interval( tempBegin, tempEnd ) );

} // close getNextInterval
```

Post active

This period can extend from the end of an interval until the beginning of the next interval, or until the end of the parent simple duration (whichever comes first). During this period, any fill behavior is applied to the element. The times for this interval can no longer change. Implementations may as an optimization choose to break the time dependency relationships since they can no longer produce changes.

Interaction with restart semantics

There are two cases in which restart semantics must be considered:

1. When the current interval is playing, if `restart="always"` then any instance time (call it `T`) in the begin list that is after (i.e. later than) the current interval begin but earlier than the current interval end will cause the current interval to end at time

T. This is the first step in restarting the element: when the current interval ends, that in turn will create any following interval.

2. When a new instance time is added to the begin list of instance times, restart rules can apply. The new instance times may result from a begin condition that specifies one of the syncbase value conditions, for which a new instance notice is received. It may also result from a begin condition that specifies one of the event value conditions, for which the associated event happens.
   In either case, the restart setting and the state of the current interval controls the resulting behavior. The new instance time is computed (e.g. from the syncbase current interval time or from the event time, and including any offset), and added to the begin list. Then:
   - If the current interval is waiting to play, the element recalculates the begin and end times for the current interval, as described in the Element life-cycle step 1 (for the first interval) or step 4 (for all later intervals). If either the begin or end time of the current interval changes, these changes must be propagated to time dependents accordingly.
   - If the current interval is playing (i.e. it is active), then the restart setting determines the behavior:
     - If `restart="never"` then nothing more is done. It is possible (if the new instance time is associated with a syncbase value condition) that the new instance time will be used the next time the element life cycle begins.
     - If `restart="whenNotActive"` then nothing more is done. If the time falls within the current interval, the element cannot restart, and if it falls after, then the normal processing at the end of the current interval will handle it. If the time falls before the current interval, as can happen if the time includes a negative offset, the element does not restart (the new instance time is effectively ignored).
     - If `restart="always"` then case 1 above applies, and will cause the current interval to end.

Cyclic dependencies in the timegraph

There are two types of cycles that can be created with SMIL 2.1, *closed* cycles and *open* or *propagating* cycles. A *closed* cycle results when a set of elements has mutually dependent time conditions, and no other conditions on the affected elements can affect or change this dependency relationship, as in examples 1 and 2 below. An *open* or *propagating* cycle results when a set of elements has mutually dependent time conditions, but at least one of the conditions involved has more than one resolved condition. If any one of the elements in the cycle can generate more than one interval, the cycle can propagate. In some cases such as that illustrated in example 3, this can be very useful.

Times defined in a closed cycle are unresolved, unless some external mechanism resolves one of the element time values (for example a DOM method call or the traversal of a hyperlink that targets one of the elements). If this happens, the resolved time will propagate through the cycle, resolving all the associated time values.

Closed cycles are an error, and may cause the entire document to fail. In some implementations, the elements in the cycle may just not begin or end correctly. Examples 1 and 2 describe the most forgiving behavior, but implementations may simply reject a document with a closed cycle.

Detecting Cycles

Implementations can detect cycles in the timegraph using a *visited* flag on each element as part of the processing that propagates changes to time dependents. As a changed time notice is propagated, each dependent element is marked as having been *visited*. If the change to a dependent instance time results in a change to the current interval for that element, this change will propagate in turn to its dependents. This second *chained* notice happens in the context of the first time-change notice that caused it. The effect is like a stack that builds as changes propagate throughout the graph, and then unwinds when all changes have propagated. If there is a dependency cycle, The propagation path will traverse an element twice during a given propagation chain. This is a common technique used in graph traversals.

A similar approach can be used when building dependency chains during initialization of the timegraph, and when propagating new interval notices - variations on the theme will be specific to individual implementations.

When a cycle is detected, the change propagation is ignored. The element that detected the second visit ignores the second change notice, and so breaks the cycle.

Examples

Example 1: In the following example, the 2 images define begin times that are mutually dependent. There is no way to resolve these, and so the images will never begin.

```
<img id="foo" begin="bar.begin" .../>
<img id="bar" begin="foo.begin" .../>
```

Example 2: In the following example, the 3 images define a less obvious cycle of begin and end times that are mutually dependent. There is no way to resolve these. The image "joe" will begin but will never end, and the images "foo" and "bar" will never begin.

```
<img id="foo" begin="joe.end" .../>
<img id="bar" begin="foo.begin" dur="3s" .../>
<img id="joe" begin="0" end="bar.end" .../>
```

Example 3: In the following example, the 2 images define begin times that are mutually dependent, but the first has multiple begin conditions that allow the cycle to propagate forwards. The image "foo" will first be displayed from 0 to 3 seconds, with the second image "bar" displayed from 2 to 5 seconds. As each new current interval of "foo" and "bar" are created, they will add a new instance time to the other element's begin list, and so the cycle keeps going forward. As this overlapping "ping-pong" behavior is not otherwise easy to author, these types of cycles are not precluded. Moreover, the correct behavior will fall out of the model described above.

```
<img id="foo" begin="0; bar.begin+2s" dur="3s" .../>
<img id="bar" begin="foo.begin+2s" dur="3s" .../>
```

Example 4: In the following example, an open cycle is described that propagates backwards. The intended behavior does not fall out of the model, and is not supported. In this example, however, each time the parent time container repeats, the video elements will begin two seconds earlier than they did in the previous parent iteration. This is because the begin instance times associated with syncbase value conditions are not cleared when the parent repeats. By the last iteration of the parent time container, both video elements would begin so early that they will be completely cut off by the parent begin constraint.

```
<par dur="10s" repeatCount="11" >
   <video id="foo" begin="0; bar.begin-1s" dur="10s" .../>
   <video id="bar" begin="foo.begin-1s" dur="10s" .../>
</par>
```

Timing and real-world clock times

This section is informative

In this specification, elements are described as having local "time". In particular, many offsets are computed in the simple time of a parent time container. However, simple durations can be repeated, and elements can begin and restart in many ways.

This section is normative

- There is no direct relationship between the local "time" for an element, and the real world concept of time as reflected on a clock.

Interval timing

This section is informative

The SMIL timing model assumes the most common model for *interval timing*.

This section is normative

- Interval timing describes intervals of time (i.e. durations) in which the begin time of the interval is included in the interval, but the end time is excluded from the interval.

This section is informative

This is also referred to as *end-point exclusive* timing. This model makes arithmetic for intervals work correctly, and provides sensible models for sequences of intervals.

Background rationale

In the real world, this is equivalent to the way that seconds add up to minutes, and minutes add up to hours. Although a minute is described as 60 seconds, a digital clock never shows more than 59 seconds. Adding one more second to "00:59" does not yield "00:60" but rather "01:00", or 1 minute and 0 seconds. The theoretical end time of 60 seconds that describes a minute interval is excluded from the actual interval.

In the world of media and timelines, the same applies: Let "A" be a video, a clip of audio, or an animation. Assume "A" begins at 10 and runs until 15 (in any units - it does not matter). If "B" is defined to follow "A", then it begins at 15 (and not at 15 plus some minimum interval). When a runtime actually renders out frames (or samples for audio), and must render the time "15", it should not show both a frame of "A" and a frame of "B", but rather should only show the new element "B". This is the same for audio, or for any interval on a timeline. If the model does not use endpoint-exclusive timing, it will draw overlapping frames, or have overlapping samples of audio, of sequenced animations, etc.

Note that transitions from "A" to "B" also adhere to the interval timing model. They *do* require that "A" not actually end at 15, and that both elements actually overlap. Nevertheless, the "A" duration is simply extended by the transition duration (e.g. 1 second). This new duration for "A" is *also* endpoint exclusive - at the end of this new duration, the transition will be complete, and only "B" should be rendered - "A" is no longer needed.

Implications for the time model

For the time model, several results of this are important: the definition of repeat, and the state of the element applied or displayed when the element is "frozen".

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

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

At time 0, the simple duration is also at 0, and the first frame of video is presented. This is the *inclusive* begin of the interval. The simple duration proceeds normally up to 4 seconds.

This section is normative

- The appropriate way to map time on the active duration to time on the simple duration is to use the remainder of division by the simple duration:

  `simpleTime = REMAINDER( t, d )` where t is within the active duration

  Note: `REMAINDER( t, d )` is defined as `t - (d*floor(t/d))`

Using this, a time of **4** (or 8 or 12) maps to the time of **0** on the simple duration. The endpoint of the simple duration is *excluded* from (i.e. not actually sampled on) the simple duration.

For most continuous media, this aligns to the internal media model, and so no frames (or audio samples) are ever excluded. However for sampled timeline media (like animation), the distinction is important, and requires a specific semantic for elements that are frozen.

- If the active duration is an even multiple of the simple duration, the media to show when frozen is the last frame (or last value) defined for the simple duration.

The effect of this semantic upon animation functions is detailed in the [SMIL-ANIMATION] module.

Event sensitivity

This section is informative

The SMIL 2.1 timing model supports synchronization based upon unpredictable events such as DOM events or user interface generated events. The model for handling events is that the notification of the event is delivered to the timing element, and the timing element uses a set of rules to resolve any synchronization dependent upon the event.

Note:

- The SMIL 2.1 test suite contains many examples of event-based timing, and states the preferred behavior for these tests. However, it is acknowledged that the timing of event propagation is implementation dependent, and so there are occasions in which delivery of an event may not occur because an intervening state change in the timegraph precludes event delivery (as per the event sensitivity rules). That is, after the event generation but before the event dispatch and handling something else in the timegraph is evaluated which precludes event delivery. This includes the internally generated timing events beginEvent, endEvent, and repeatEvent, as well as externally generated events such as the focusInEvent and focusOutEvent.
  In these cases, it is desirable for model implementations to behave as if they responded to the internal timing events instantaneously, but an implementation is considered conformant if event propagation delay precludes this behavior.
- For example, in timing test case 1.15, the delivery of the image1.beginEvent to the image2 element may not occur until after the par has ended at 3s (due to image1 having 0 duration and no other children of the par with resolved begin times), and so it is also compliant behavior for the image1 element to show for 0 seconds and then for the par to end.

This section is normative

The semantics of element sensitivity to events are described by the following set of rules:

1. While a time container is not active (i.e. before the time container begin or after the time container active end), child elements do *not* respond to events (with respect to the Time model). Note that while a parent time container is frozen, it is not active, and so children do not handle begin or end event specifications.
    a. If an element and an ascendant time container are both specified to begin with the same event, the behavior is not predictable (based upon DOM event semantics). Authors are discouraged from authoring these cases.
2. If an element is not active (but the parent time container is), then events are only handled for begin specifications. Thus if an event is raised and **begin** specifies the event, the element begins. While the element is not active, any **end** specification of the event is ignored.
3. If an element is (already) active when an event is raised, and **begin** specifies the event, then the behavior depends upon the value of restart:
    a. If **restart**=`"always"`, then a new begin time is resolved for the element based on the event time. Any specification of the event in **end** is ignored for this event instance.
    b. If **restart**=`"never"` or **restart**=`"whenNotActive"`, then any **begin** specification of the event is ignored for this instance of the event. If **end** specifies the event, an end value is resolved based upon the event time, and the active duration is re-evaluated (according to the rules in Computing the active duration).

It is important to notice that in no case is a single event occurrence used to resolve both a begin and end time on the same element.

This section is informative

Rule 1a discourages the use of cases such as the following:

```
<smil ...>
...
<par id="bad_example" begin="link9.activateEvent">
   <img begin="link9.activateEvent" .../>
</par>
...
</smil>
```

Various alternative approaches can be used. One possible approach is to define the descendent element to begin relative to the ascendant begin, as in the following example (the begin rule for the image could be simpler, but this illustrates the general point):

```
<smil ...>
...
<par id="better_example" begin="link9.activateEvent">
   <img begin="better_example.begin" .../>
</par>
...
</smil>
```

The event sensitivity rules can be used with the restart attribute to describe "toggle" activation use cases, as described in the section: Using restart for toggle activation.

Since the same event instance cannot be used to resolve both the begin and end time on a single element, uses like the following will have behavior that may seem non-intuitive to some people:

```
<html ...>
...
<audio src="bounce.wav" begin="foo.click"
        end="foo.click+3s" restart="whenNotActive"/>
...
</html>
```

This example will begin repeating the audio clip when "foo" is clicked, and stop the audio clip 3 seconds after "foo" is clicked *a second time*. It is incorrect to interpret this example as playing the audio clip for 3 seconds after "foo" is clicked. For that behavior, the following markup should be used:

```
<html ...>
...
<audio src="bounce.wav" begin="foo.click" dur="3s"
        restart="whenNotActive"/>
...
</html>
```

User event sensitivity and timing

The timing model and the user event model are largely orthogonal. While the timing model does reference user events, it does not define how these events are generated, and in particular does not define semantics of keyboard focus, mouse containment, "clickability", and related issues. Because timing can affect the presentation of elements, it may impact the rules for user event processing, however it only has an effect to the extent that the presentation of the element is affected.

In particular, many user event models will make no distinction between an element that is "playing" and one that is "frozen". The effects of the fill attribute apply only to the timing semantics. If an element is still visible while frozen, it behaves normally with respect to other semantics such as user event processing. In particular, elements such as a and area are still sensitive to user activation (e.g. clicks) when frozen.

Link Activation compared to Event activation

Related to event-activation is *link-activation*. Hyperlinking has defined semantics in SMIL 1.0 to seek a document to a point in time. When combined with interactive timing (e.g. begin=`"indefinite"`), hyperlinking yields a variant on user-interactive content.

This section is normative

- A hyperlink can be targeted at an element that does not have a scheduled begin time.
- When the link is traversed, the element begins.
- Note that unlike event activation, the hyperlink activation is not subject to the constraints of the parent time container.

The details of when hyperlinks activate an element, and when they seek the document timeline are presented in the section Hyperlinks and timing.

Converting between local and global times

This section is normative

To convert a document time to an element local time, the original time is converted to a simple time for each time container from the root time container down to the parent time container for the element. This recursive algorithm allows for a simple model of the conversion from parent simple time to element active and element simple time. The first step calculates element active time, and the second step calculates element simple time.

The steps below assume that the associated times are resolved and not indefinite. If a required time is not resolved or is indefinite, then the conversion is not defined, and cannot be performed.

Element active time calculation

The input time is a time in parent simple time. This is normalized to the element active duration, adjusting for the accumulated synchronization offset (described in The accumulated synchronization offset).

> Let $\mathbf{t_{ps}}$ be a time in parent simple time, **B** be the begin time for an element, and **O** be the accumulated synchronization offset for an element, measured in parent simple time.
>
> The element active time $\mathbf{t_a}$ for any child element is:
>
> $$\mathbf{t_a = t_{ps} - B - O}$$

Element simple time calculation

The element simple time is the time that is used to establish runtime synchronization for a media element, or to compute an animation function's input value or sampling time. If the element is a time container, this is also the time that is seen by all children of a time container (as the time container element's simple time).

To compute the element simple time $\mathbf{t_s}$ from an element active time $\mathbf{t_a}$, accounting for any repeat behavior:

> If there is no repeating behavior:
>
> $$\mathbf{t_s = t_a}$$
>
> Else, the element simple time is just computed from the begin time of the most recent iteration - call this $\mathbf{t_{last\text{-}repeat}}$. Some other mechanism (such as endsync logic or a media player) must note when the simple duration ends, and reset the value of $\mathbf{t_{last\text{-}repeat}}$. If the element has not yet repeated, a value of 0 is used in place of $\mathbf{t_{last\text{-}repeat}}$.
>
> $$\mathbf{t_s = t_a - t_{last\text{-}repeat}}$$

Note that the above semantic covers the special (ideal) case when the simple duration **dur** is fixed and does not vary. In this case (and this case only) $\mathbf{t_{last\text{-}repeat}}$ can be obtained directly for the simple duration **dur** and so the expression can be reduced to:

> $$\mathbf{t_s = REMAINDER(\ t_a, dur\ )}$$
>
> where **REMAINDER(t, d)** is defined as **(t - d*floor(t/d))**.

Converting wall-clock values

When the document begins, the current wall-clock time is noted and saved as $\mathbf{t_{wallclock\text{-}begin}}$. To convert a wall-clock value $\mathbf{t_{wc}}$ to an element active simple time $\mathbf{t_s}$, first convert $\mathbf{t_{wc}}$ to a document global time $\mathbf{t_{ra}}$ (i.e. an element active time for the root time container):

> $$\mathbf{t_{ra} = t_{wc} - t_{wallclock\text{-}begin}}$$

This may yield a negative time if the wallclock value is a time before the document began. Nevertheless, this is a legal value.

The time $\mathbf{t_{ra}}$ is then converted normally to element active time or element local time as needed.

Converting from event time to element time

Event times are generally stamped with a time relative to system time or when the document began. The conversion is as for wallclock values, in that the event time is converted to an active time for the root time container, and then converted normally to an element time.

Converting from element time to element time

To convert from one element timespace to another, the time for the first element $\mathbf{t_{e1}}$ must first be converted to a simple time on the closest ascendant time container that contains both elements. Converting from an element time to the parent time reverses the process described above. Again, it is recursive, and so the conversions are described generically from element simple to element active time, and from element active to parent simple time.

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

$$\mathbf{t_a = t_s + t_{last\text{-}repeat}}$$

Conversion from element active time to parent simple time uses the associated begin of the element and the accumulated synchronization offset.

$$\mathbf{t_{ps} = t_a + B + O}$$

Time conversions and sampling the time graph

Note that the pure conversions do not take into account the clamping of active durations, nor the effects of fill (where time is frozen). Global to local time conversions used to translate between timespaces must ignore these issues, and so may yield a time in the destination local timespace that is well before or well after the simple duration of the element.

An alternate form of the conversion is used when actually sampling the time graph. A time container is only sampled if it is active or frozen, and so no times will be produced that are before a time container begins. If the global to local time conversion for a time container yields a time during which the time container is frozen, the time is clamped to the value of the active end.

Hyperlinks and timing

This section is informative

Hyperlinking semantics must be specifically defined within the time model in order to ensure predictable behavior. Earlier hyperlinking semantics, such as those defined by SMIL 1.0 are insufficient because they do not handle unresolved times, nor do they handle author-time restart restrictions. Here we extend SMIL 1.0 semantics for use in presentations using elements with unresolved timing (including interactive timing) and author-time restart restrictions.

This section is normative

A hyperlink may be targeted at an element by specifying the value of the id attribute of an element in the fragment part of the link locator. Traversing a hyperlink that refers to a timed element will behave according to the following rules:

1. If the target element is active, seek the document time back to the begin time of the current interval for the element.

2. Else if the target element begin time is resolved (i.e. there is at least one interval defined for the element), seek the document time (forward or back, as needed) to the begin time of the first interval for the target element. Note that the begin time may be resolved as a result of an earlier hyperlink, DOM or event activation. Once the begin time is resolved (and until the element is reset, e.g. when the parent repeats), hyperlink traversal always seeks. For a discussion of "reset", see Resetting element state. Note also that for an element begin to be resolved, the begin time of all ancestor elements must also be resolved.
3. Else (i.e. there are no defined intervals for the element), the target element begin time must be resolved. This may require seeking and/or resolving ancestor elements as well. This is done by recursing from the target element up to the closest ancestor element that has a resolved begin time (again noting that for an element to have a resolved begin time, all of its ancestors must have resolved begin times). Then, the recursion is "unwound", and for each ancestor in turn (beneath the resolved ancestor) as well as the target element, the following steps are performed:
    1. If the element begin time is resolved, seek the document time (forward or back, as needed) to the begin time of the first interval for the target element.
    2. Else (if the begin time is not resolved), just resolve the element begin time at the current time on its parent time container (given the current document position). Disregard the sync-base or event base of the element, and do not "back-propagate" any timing logic to resolve the element, but rather treat it as though it were defined with begin="indefinite" and just resolve begin time to the current parent time. This should create an interval and propagate to time dependents.

In the above rules, the following additional constraint must also be respected:

1. If a begin time to be used as the seek target occurs before the beginning of the parent time container, the seek-to time is *clamped* to the begin time of the parent time container. This constraint is applied recursively for all ascendant time containers.
2. If a begin time to be used as the seek target occurs after the end of any ascendant time container's simple duration, then the seek-to time is *clamped* to the time container simple end time.

This section is informative

Note that the first constraint means that a hyperlink to a child of a time container will never seek to a time earlier than the beginning of the time container. The second constraint implies that a hyperlink to a child that begins after the end of the parent simple duration will seek to the end of the parent, and proceed from there. While this may produce surprising results, it is the most reasonable fallback semantic for what is essentially an error in the presentation.

If a seek of the presentation time is required, it may be necessary to seek either forward or backward, depending upon the resolved begin time of the element and the presentation current time at the moment of hyperlink traversal.

This section is normative

- After seeking a document forward, the document should be in largely the same state as if the user had allowed the presentation to run normally from the current time until reaching the element begin time (but had otherwise not interacted with the document). One exception relates to event-based timing in the document, and is described below.

- Seeking the presentation time forward should also begin any other elements that have resolved begin times between the current time and the seeked-to time. The elements that are begun in this manner may still be active, may be frozen, or may already have ended at the seeked-to time.
- If an element begins and ends within the seek interval (between the current time before the hyperlink traversal and the seeked-to time) it logically begins and ends during the seek.
    - In this case, the associated `beginEvent`, `endEvent` and any `repeatEvent` events are not raised.
    - Dependent times defined relative to these events will not be resolved as a result of the seek.
    - DOM events (in profiles that support a DOM) will not be raised, and script or other listeners associated with these events will not be called.
- Any elements currently active at the time of hyperlinking should "fast-forward" over the seek interval. These elements may also be active, frozen or already ended at the seeked-to time.
    - If the element ends during the seek interval, an `endEvent` is raised. The associated time for the event is the document time before the seek.
    - If the element repeats within the seek interval, any associated `repeatEvents` are not raised.
- Automatic hyperlinks (i.e. those with an **actuate** value of **`onLoad`**) will not be actuated when they are seeked over during hyperlink traversal (the active duration of the hyperlink begins and ends during the seek interval). However, automatic hyperlinks that are only partially seeked over will be actuated (seeking into the active duration of an automatic hyperlink will actuate the hyperlink). This models the effects of seeking and automatic hyperlinks in the same manner as timing events.

The net effect is that seeking forward to a presentation time puts the document into a state largely identical to that as if the document presentation time advanced undisturbed to reach the seek time. If the presentation is authored with no `beginEvent`, `endEvent` or `repeatEvent` based timing and no automatic hyperlinks, then state of the document after a seek should be identical to that had the document presentation time advanced undisturbed to reach the seeked-to time.

If the resolved activation time for an element that is the target of a hyperlink traversal occurs in the past, the presentation time must seek backwards. Seeking backwards will rewind any elements active at the time of hyperlinking.

- Just as for forward seeks, elements that begin and end within the seek intervals will not raise `beginEvent`, `endEvent` or `repeatEvent` events.
- If an element is active at the time of hyperlinking and the element's current interval begins during the seek interval, the element is turned off and an `endEvent` is raised. The associated time for the event is the document time before the seek. This action does not resolve any times in the instance times list for end times.
- If the element repeats within the seek interval, any associated `repeatEvents` are not raised.
- Resolved begin times (e.g. a begin associated with an event) are not cleared or lost by seeking to an earlier time. Resolved end times associated with events, repeat-values, accesskey-values or added via DOM method calls are cleared

when seeking to time earlier than the resolved end time. This follows the semantics for resetting element state.

- Seeking to a time before a resolved begin time does not affect the interpretation of a "restart=never" setting for an element; once the begin time is resolved, it cannot be changed or restarted.
- When the document seeks backwards before a resolved begin for an element time, this does not reset the element.
- Once resolved, begin times are not cleared by hyperlinking. However, they can be overwritten by subsequent resolutions driven by multiple occurrences of an event (i.e. by restarting).

This section is informative

These hyperlinking semantics assume that a record is kept of the resolved begin time for all elements, and this record is available to be used for determining the correct presentation time to seek to. For example:

```
<html ...>
...
<par begin="0">
   <img id="A" begin="10s" .../>
   <img id="B" begin="A.begin+5s" .../>
   <img id="C" begin="B.click" .../>
   <img id="D" begin="C.begin+5s" .../>
   ...
   <a href="#D">Begin image D</a>
</par>
...
</html>
```

The begin time of elements **A** and **B** can be immediately resolved to be at 10 and 15 seconds respectively. The begin of elements **C** and **D** are unresolved when the document starts. Therefore activating the hyperlink will resolve the begin of **D** but have no effect upon the presentation time for element **C**.

Now, assume that **B** is clicked at 25 seconds into the presentation. The click on **B** resolves the begin of **C**; this in turn resolves **D** to begin at 30 seconds. From this point on, traversing the hyperlink will cause the presentation time to be seeked to 30 seconds.

If at 60 seconds into the presentation, the user again clicks on **B**, **D** will become re-resolved to a presentation time of 65 seconds. Subsequent activation of the hyperlink while **D** is active will result in the seeking the presentation to 65 seconds. If the hyperlink is activated when **D** is no longer active, the presentation will seek to the earliest resolved begin time of **D**, at 30 seconds.

*Implications of beginElement() and hyperlinking for **seq** and **excl** time containers*

This section is normative

For a child of a sequence time container, if a hyperlink targeted to the child is traversed, this seeks the sequence to the beginning of the child.

- If the seek is forward in time and the child does not have a resolved begin time, the document time must seek past any scheduled active end on preceding

elements, and then activate the referenced child. In such a seek, if the currently active element does not have a resolved active end, it should be ended at the current time. An `endEvent` event is raised, with the current time as the associated event time.
- If there are other intervening siblings (between the currently playing element and the targeted element), the document time must seek past all scheduled times, and resolve any unresolved times as seek proceeds (time will resolve to intermediate values of "now" as this process proceeds).
- As times are resolved, all associated time dependents get notified. Note however, that since no events are raised for elements that begin and end (or repeat) within the seek interval, time dependents defined relative to these events will not be notified and dependent times will not be resolved.
- When `beginElement()` or `beginElementAt()` is called for the child of a sequence time container (subject to restart semantics), any currently active or frozen child is stopped and the new child is begun at the current time (even if the element has a scheduled begin time). Unlike hyperlinking, no seek is performed. The sequence will play normally following the child that is begun with the method call (i.e. as though the child had begun at its normal time).
- Note that the presentation agent need not actually prepare any media for elements that are seeked over, but it does need to propagate the sync behavior to all time dependents so that the effect of the seek is correct.

This section is informative

Note that if a hyperlink targets (or if `beginElement()` or `beginElementAt()` is called for) an element **A** defined to begin when another element **B** ends, and the other element **B** has (e.g.) an event-base or syncbase end, the hyperlink or method call will not end element **B**. It will only activate element **A**. If the two elements are siblings within a **seq** or **excl** time container, the parent time container enforces its semantics and stops (or pauses) the running element.

If a hyperlink targets a child of an **excl** time container, activating the link will seek to the earliest computed begin. This means that pause/defer stack semantics do not need to be accounted for when linking to an element. Instead the document timeline will simply be seeked to the first resolved time for the element, or seeked to the start of the time container and the target element simply started if there is no resolved begin time.

Propagating changes to times

This section is informative

There are several cases in which times may change as the document is presented. In particular, when an element time is defined relative to an event, the time (i.e. the element begin or active end) is resolved when the event occurs. Another case arises with restart behavior - the element gets a new begin and active end time when it restarts. Since the begin and active end times of one element can be defined relative to the begin or active end of other elements, any changes to times must be propagated throughout the document.

When an element "foo" has a begin or active end time that specifies a syncbase element (e.g. "bar" as below):

```
<img id="foo" begin="bar.end" .../>
```

we say that "foo" is a *time-dependent* of "bar" - that is, the "foo" begin time depends upon the active end of "bar". Any changes to the active end time of "bar" must be propagated to the begin of "foo" so that "foo" begins properly when "bar" ends. The effect on "foo" of the propagated change depends upon the state of "foo" when the change happens.

This section is normative

- If an element begin time or end time changes, this change must be propagated to any other elements that are defined relative to the changed time (i.e. to all time dependents). More specifically, when the begin or end of the current interval for an element changes, it must propagate the change to all dependent instance times. If the dependent element current interval begin or end times change as a result, these changes must in turn also be propagated. For details, see Evaluation of begin and end time lists.
- If an element ends later than its specified endtime (e.g. if a negative offset is specified with a userevent), the propagated time must be the computed time and not the observed time.
- If an element ends earlier than its specified endtime (e.g. if a parent time container constraint cuts short the element, or a DOM method call ends the element), the propagated time must be the constrained time.

Deferred elements and propagating changes to begin

One exception to normal processing is made for elements that are *deferred* according to **excl** interrupt semantics. This exception is made to simplify the model: once an element is deferred, it will stop normal handling of time change notices that are propagated to the element begin conditions, as time dependents of syncbase elements. That is, with respect to the behavior of the element as a time dependent, the element behaves as though it had already begun. This exception is made so that the deferred element cannot change its begin time due to syncbase element changes, while it is deferred. In effect, the element *should have begun* at the time it was deferred, and so it should no longer handle changed time notices.

This section is informative

Restart and propagating changes to times

In some cases, the semantics of restart may preclude the correct propagation of changes to time, as in the following example:

```
<html ...>
...
<par>
   <img id="img1" dur="10s" end="click" .../>
   <video begin="img1.end-3s" restart="whenNotActive" .../>
</par>
...
</html>
```

If the user clicks the image at 8 seconds, the image will end at that point, and the changed end time will propagate to the video. However, the video will have begun at 7 seconds (3 seconds before the calculated end of 10 seconds), and cannot restart. The propagated change will be ignored. See also Interaction with restart semantics in the section on Evaluation of begin and end time lists.

Time container duration

This section is informative

The implicit duration of a time container is defined in terms of the children of the container. The children can be thought of as the "media" that is "played" by the time container element. The semantics are specific to each of the defined time container variants, and are described in the respective sections: The **par** element, the **seq** element, and the **excl** element.

Note that the term "computed values" should not be confused with the values of times that are dynamic within the time graph. In the following example, the video will be cut short if the user activates (e.g., clicks on) it before 10 seconds. If the user does not click, the **par** has a simple duration of 10 seconds. If the user activates the video at 5 seconds, the **par** has a simple duration of 8 seconds. Although the original end time for the video could be computed by an implementation as 10 seconds, the endsync semantics must be evaluated with the updated times that account for the user events.

```
<smil ...>
...
<par endsync="last" >
   <audio dur="8s" .../>
   <video begin="0" dur="10s" end="click" .../>
</par>
...
</smil>
```

Time container constraints on child durations

This section is informative

Time containers place certain overriding constraints upon the child elements. These constraints can cut short the active duration of any child element.

This section is normative

All time containers share the basic overriding constraint:

- A child element may not be active before the beginning, nor after the end of either the parent simple duration or parent active duration.
- Note the time container is itself subject to the same constraints, and so may be cut short by some ascendant time container. When this happens, the children of the time container are also cut off, in the same manner as for the last partial repeat in the example below.

This section is informative

While the child may define a sync relationship that places the begin before the parent begin, the child is not active until the parent begins. This is equivalent to the semantic described in Negative begin delays.

If the child defines an active duration (or by the same token a simple duration) that extends beyond the end of the parent simple duration, the active duration of the child will be cut short when the parent simple duration ends. Note that this does not imply that the child duration is automatically shortened, or that the parent simple duration is "inherited" by the child.

For example:

```
<par dur="10s" repeatDur="25s">
   <video dur="6s" repeatCount="2" .../>
   <text id="text1" begin="5s" dur="indefinite" .../>
   <audio begin="text1.end" .../>
</par>
```

The video will play once for 6 seconds, and then a second time but only for 4 seconds - the last 2 seconds will get cut short and will not be seen. The text shows up for the last 5 seconds of the **par**, and the indefinite duration is cut short at the end of the simple duration of the **par**. The audio will not show up at all, since it is defined to begin at the end of the active duration of the previous element (the **text** element). Since the text element ends when the time container ends, the audio would begin after the time container has ended, and so never is heard. When the **par** repeats the first time, everything has happens just as it did the first time. However the last repeat is only a partial repeat (5 seconds), and so only the video will be seen, but it will not be seen to repeat, and the last second of the video will be cut off.

In addition, **excl** time containers allow only one child to play at once. Subject to the **priorityClass** semantics, the active duration of an element may be cut short when another element in the time container begins.

*The **min** attribute and time container constraints on child durations*

The fill attribute is also used to extend the active duration if it is less than the duration specified in the min attribute.

```
<par dur="5s">
   <img id="img" min="7s" dur="4s" fill="freeze".../>
</par>
```

Time container constraints on sync-arcs and events

This section is informative

SMIL 1.0 defined constraints on sync-arc definition (e.g., begin="id(image1)(begin)"), allowing references only to qualified siblings. SMIL 2.1 explicitly removes this constraint. SMIL 2.1 also adds event-based timing. Both sync-arcs and event-timing are constrained by the parent time container of the associated element as described above.

Specifics for sync-arcs

This section is normative

While a sync-arc is explicitly defined relative to a particular element, if this element is not a sibling element, then the sync is resolved as a sync-relationship *to the parent* (i.e. to an offset from the parent begin).

- If the defined sync would place the resolved element begin before the parent time container begin, part of the element will simply be cut off when it first plays. This is like the behavior obtained using clipBegin.
- However unlike with clipBegin, if the sync-arc defined child element also has repeat specified, only the first iteration will be cut off, and subsequent repeat iterations will play normally. See also Negative begin delays.

This section is informative

Note that in particular, an element defined with a sync-arc begin will not automatically force the parent or any ancestor time container to begin.

For the case that an element with a sync-arc is in a parent (or ancestor) time container that repeats: for each iteration of the parent or ancestor, the element is played as though it were the first time the parent timeline was playing. With each repeat of the parent, the sync-arc will be recalculated to yield a begin time relative to the parent time container. See also the section Resetting element state.

Specifics for event-based timing

This section is informative

The specifics for event-based timing are discussed in the Event Sensitivity section.

Behavior of 0 duration elements

This section is normative

- Media elements with an active duration of zero or with the same begin and end time trigger begin and end events, and propagate to time dependents. If an element's end time is before its begin time, no events are triggered (see also Evaluation of begin and end time lists).

Whether or not media with zero duration and no fill period is retrieved and/or briefly rendered is implementation dependent.

10.4.4 Clarifications and surprising results

When an element begins, any event-based begin times are cleared. In the following example, if an activate event occurs and then one second later bar ends, then foo begins immediately and the element does not restart four seconds later regardless of the restart setting. However, if an activate event occurs and bar does not end during the next five seconds, the element will restart at the end of that time.

```
<audio id="foo" begin="bar.end; activateEvent+5s".../>
```

See Evaluation of begin and end time lists.

10.5 Integrating SMIL Timing and Synchronization into a host language

This section is informative

This section describes what a language designer must actually do to specify the integration of SMIL Timing and Synchronization support into a host language. This includes basic definitions, constraints upon specification, and allowed/supported events.

10.5.1 Required host language definitions

This section is informative

The host language designer must define some basic concepts in the context of the particular host language. These provide the basis for timing and presentation semantics.

This section is normative

- Any host language that includes SMIL 2.1 Timing and Synchronization markup (either via a hybrid DTD or schema, or via namespace qualified extensions) must preserve the semantics of the model defined in this specification.
- Only SMIL *document user agents* must support the deprecated SMIL 1.0 attribute names as well as the new SMIL 2.1 names. A SMIL *document user agent* is an application that supports playback of SMIL Language documents (i.e. documents with the associated MIME type "`application/smil+xml`" or "`application/smil`").
- The host language designer must define what "presenting a document" means. A typical example is that the document is displayed on a screen.
- The host language designer must define the *document* begin. Possible definitions are that the document begins when the complete document has been received by a client over a network, or that the document begins when certain document parts have been received.
- The host language designer must define the *document* end. This is typically when the associated application exits or switches context to another document.

10.5.2 Required definitions and constraints on element timing

This section is normative

- A host language must specify which elements support timing
- A host language must specify the semantics of the top level time container, even if the language does not otherwise include time containers.
- A host language must specify the semantics of an element being active, frozen, paused, and not active in the sense of timing. This may include support for additional syntax to indicate this semantic. See also the **timeAction** attribute.
- A host language must specify which elements define media directly. This defines which elements may take the "`media`" argument value to the **dur** attribute.

Supported events for event-base timing

This section is normative

- The host language must specify which event names are legal in event base values.
- If the host language defines no allowed event names, event-based timing is effectively precluded for the host language.

- Host languages may specify that dynamically created events (as per the [DOM2Events] specification) are legal as event names, and not explicitly list the allowed names.

10.5.3 Error handling semantics

This section is normative

- The host language designer may impose stricter constraints upon the error handling semantics.
- In the case of syntax errors, the host language may specify additional or stricter mechanisms to be used to indicate anerror. An example would be to stop all processing of the document, or to halt all animation.
- Host language designers may not relax the error handling specifications, or the error handling response (as described in "Handling syntax errors"). For example, host language designers may not define error recovery semantics for missing or erroneous values in the **begin** or **end** attribute values.

10.6 Document object model support

This section is informative

Any XML-based language that integrates SMIL Timing will inherit the basic interfaces defined in DOM [DOM2]. SMIL Timing specifies the interaction of timing functionality and DOM. SMIL Timing also defines constraints upon the basic DOM interfaces. A separate document will define specific DOM interfaces to support SMIL Timing, however this document presumes that there is a mechanism to begin and end elements, and to pause and resume them.

This section is normative

No syntax support is required to make use of the presumed interfaces, although the "indefinite" argument value on the begin and end attributes can be used to describe timing that will be initiated by DOM methods. In any case, the actions of DOM timing methods will be subject to the constraints of the time model, as described in this document.

A language integrating SMIL Timing and Synchronization need not require a DOM implementation.

10.6.1 Element and attribute manipulation, mutation and constraints

If the timing attributes of timed elements are manipulated through DOM interfaces while the timegraph is running, the behavior is not defined by this document. Similarly, if timed elements are inserted into or removed from the document while the timegraph is running, the behavior is not defined. The behavior and any constraints related to this will be specified in a future document.

10.6.2 Events and event model

This section is informative

SMIL event-timing assumes that the host language supports events, and that the events can be bound in a declarative manner. DOM Level 2 Events [DOM2Events] describes functionality to support this.

This section is normative

The specific events supported are defined by the host language. If no events are defined by a host language, event-timing is effectively omitted.

This module defines a set of events that may be included by a host language. These include:

beginEvent
: This event is raised when the element local timeline begins to play. It will be raised each time the element begins the active duration (i.e. when it restarts, but not when it repeats). It may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was begun with a DOM method.

endEvent
: This event is raised at the active end of the element. Note that this event is not raised at the simple end of each repeat. This event may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was ended with a DOM method.

repeatEvent and repeat
: Depending on the profile, one or the other of these events is raised when the element local timeline repeats. It will be raised each time the element repeats, after the first iteration.

repeat (n)
: This event is raised when the element local timeline repeats. It will be raised each time the element repeats, after the first iteration. Associated with the repeat event is an integer that indicates which repeat iteration is beginning. The value is a 0-based integer, but the repeat event is not raised for the first iteration and so the observed values will be >= 1.

If an element is restarted while it is currently playing, the element will raise an `endEvent` and then a `beginEvent`, as the element restarts.

In order to make the model operate consistently and remove the effects of synchronization slew in a chain of event times, the timestamp value associated with events such as the `beginEvent`, `endEvent`, and `repeat` events is not (necessarily) the actual time that the event is raised, nor is it the time when a time dependent is actually notified of the event. Rather the event timestamp is the *earliest* time that the event *could* be raised (given the timing model semantics, and assuming that elements would begin and end *precisely* when they are defined to). There are three basic cases corresponding to begin and end conditions with zero, positive, and negative offsets respectively:

Example 1

These examples assume video and audio media that are recorded to be in exact sync with one another.

```
<par dur="indefinite">
  <img id="foo" end="click" .../>
  <video id="bar" begin="foo.endEvent" .../>
```

```
    <audio id="copy" begin="foo.end" .../>
  </par>
```

The image "foo" will end when the user clicks on it. The defined time of the end is actually the time of the click event (even if it takes a while to propagate the click event through the presentation mechanism). The "foo" element will raise an `endEvent` with a timestamp equal to the time of the click event. The behavior in this example is that "bar" and "copy" will be in precise synchronization (although "bar" may actually begin very slightly later, since it can take a while to propagate the events through a system).

Example 2

```
  <par dur="indefinite">
    <img id="foo" .../>
    <video id="bar" begin="foo.click+3s" .../>
    <audio id="copy" begin="bar.beginEvent" .../>
  </par>
```

The video "bar" will begin 3 seconds after the user clicks on "foo". The `beginEvent` for "bar" will have a timestamp equal to the "foo.click" event timestamp plus 3 seconds. The behavior is that in the example above, "bar" and "copy" will be in precise synchronization (although "copy" may actually begin slightly later, since it can take a while to propagate the events through a system).

Example 3

```
  <par dur="indefinite">
    <img id="foo" .../>
    <video id="bar" begin="foo.click-3s" .../>
    <audio id="copy" begin="bar.beginEvent" .../>
  </par>
```

The video "bar" will begin when the user clicks on "foo". The video will begin to play at a 3 second offset into the actual content, because it is defined to begin 3 seconds before the click. However, since "bar" cannot begin any sooner than "now" when the event is raised, it will raise a `beginEvent` that has the same time as the "foo.click" event. Thus in this case, the audio element "copy" will be precisely three seconds behind (out of sync with) the video.

Additional time model constraints can cause the `beginEvent` (or `endEvent`) event timestamp to differ from the calculated begin (or end) time for an element. For example the element can specify a begin time before the beginning of its parent time container (either with a negative offset value, or with a syncbase time that resolves to a time before the parent begin). In this case, a time dependent of the begin syncbase time will be defined relative to the calculated begin time. However, the element is constrained to not actually begin before the parent time container. The `beginEvent` will be raised when the element actually begins - in the example case when the parent time container begins. Similarly, the `endEvent` is raised when the element actually ends, which may differ from the calculated end time (e.g. when the end is specified to be after the end of the parent simple duration).

The distinction between syncbase and event times can be useful in certain situations. Consider the following example:

```
  <par>
    <par begin="5s">
      <par begin="-5s">
```

```
        <img id="foo" begin="1s; 8s" dur="3s" .../>
      </par>
    </par>
    <img id="bar" begin="foo.begin" dur="1s" .../>
    <audio id="beep" begin="foo.beginEvent" dur="1s" .../>
  </par>
```

The "foo" element defines two intervals. The inner par cuts off - but does not prune - the first interval, because the innermost par is constrained by the middle par and cannot actually begin until 5s into the document. However the inner par is still synchronized to the document time of 0s. As such, "bar" will play twice: once at 1 second, and again at 8 seconds, because syncbase values use calculated interval times. However the "beep" audio will only play once at 8 seconds which is when "foo" is actually displayed, because intervals that are cut off do not raise events.

While authors are unlikely to author the above example, similar cases can easily arise using syncbase timing. When it is important to distinguish the observed begin time from the scheduled begin time, event-value timing with the `beginEvent` or `endEvent` can be used. However, the author must be aware of the constraints on event-value timing. These include the event sensitivity constraints, and the fact that many implementations will not optimize scheduling and media preparation for elements with event-value timing as well as for elements with scheduled syncbase-value timing. See also the discussion Propagating changes to times.

10.6.3 Reserved DOM methods

This section is normative

SMIL Timing reserves four DOM methods for controlling the timing of elements: `beginElement()`, `beginElementAt()`, `endElement()`, and `endElementAt()`, and describes their effect on the timing model. Full definition of these methods is left to a future document describing DOM functionality.

The four DOM methods are used to begin and end the active duration of an element. Authors can (but are not required to) declare the timing to respond to the DOM using the following syntax:

```
<img begin="indefinite" end="indefinite" .../>
```

The `beginElement()`, `beginElementAt()`, `endElement()`, and `endElementAt()` methods are all subject to time container constraints in much the same way that event-based times are. If any of these methods are called when the parent time container is not active, the methods have no effect.

If a DOM method call is made to begin or end the element (`beginElement()`, `beginElementAt()`, `endElement()` or `endElementAt()`), each method call creates a single instance time (in the appropriate instance times list). These times are then interpreted as part of the semantics of lists of times, as described in Evaluation of begin and end time lists. These time instances are cleared upon reset just as for event times.

- The instance time associated with a `beginElement()`or `endElement()` call is the current presentation time at the time of the DOM method call.
- The instance time associated with a `beginElementAt()` or `endElementAt()` call is the current presentation time at the time of the DOM method call, plus or minus the specified offset. The offset is measured in parent simple time.

- Note that `beginElement()` and `beginElementAt()`are subject to restart semantics. Refer also to the section The **restart** attribute.

10.7 Glossary

10.7.1 General concepts

This section is informative

The following concepts are the basic terms used to describe the timing model.

Synchronization relationship

A synchronization relationship is defined by the author to express that two or more elements' playback is synchronized.

Time graph

A time graph is used to represent the temporal relations of elements in a document with SMIL timing. Nodes of the time graph represent elements in the document. Parent nodes can "contain" children, and children have a single parent. Siblings are elements that have a common parent. The links or "arcs" of the time graph represent synchronization relationships between the nodes of the graph.

Descriptive terms for times

The time model description uses a set of adjectives to describe particular concepts of timing:

implicit
: This describes a time that is defined intrinsically by the element media (e.g. based upon the length of a movie), or by the time model semantics (e.g., duration of **par** time container).

explicit
: This describes a time that has been specified by the author, using the SMIL syntax.

desired
: This is a time that the author intended - it is generally the explicit time if there is one, or the implicit time if there is no explicit time.

effective
: This is a time that is actually observed at document playback. It reflects both the constraints of the timing model as well as real-world issues such as media delivery.

definite
: A time is definite if it is resolved to a finite, non-indefinite value.

Local time and global time

Global time is defined relative to the common reference for all elements, the document root. This is sometimes also referred to as *document time*.

Within a document, when a given element is active or "plays", the contents of that element progress from the beginning of the active duration to the end of the active duration. There will also be a progression from the beginning to the end of each simple duration (the distinction is clearest when the element repeats). It is often convenient to talk about times in terms of a given element's simple duration or its active duration. Generically, this is referred to as *local time*, meaning that times are relative to an element-local reference.

The following terms are used to more precisely qualify local times:

active time
: Time as measured relative to the element's active duration. A time is measured as an offset from the active begin of the element.

simple time
: Time as measured relative to the element's simple duration. A time is measured as an offset from the beginning of a particular instance of the simple duration.

media time
: Time as measured relative to the element's media duration. A time is measured as an offset from the beginning of the media, as modified by any clipBegin or clipEnd attributes.

To be meaningful, these terms are described relative to some element. For example, when describing timing semantics, *element active time* refers to active time for the element under discussion, and *parent simple time* refers to simple time for that element's parent.

Conversion from global (document) time to an element time, or from one element time to another element time, is described in Converting between local and global times.

When measuring or calculating time, a reference element and the local time form (active, simple or media time) are specified. The measured time or duration is defined in terms of the element time progress. E.g. if the reference element pauses, this may impact the semantics of times or durations measured relative to the element.

Linear and Non-linear media

Linear media is continuous media that cannot be played in a random-access manner. For example, most Internet streaming video and audio are linear.

Non-linear media can be played in a random access manner. For example, algorithmic animation is non-linear. Discrete media may behave in a non-linear manner.

The linear or non-linear behavior of the media is not a function of the media type, but rather of the renderer or playback engine, and often depends upon the delivery mechanism for the media.

Scheduled timing

An element is considered to have scheduled timing if the element's start time is given relative to the begin or active end of another element. A scheduled element can be inserted directly into the time graph.

This section is normative

document begin

The start of the interval in which the document is presented is referred to as the *document begin*.

document end

The end of the interval in which the document is presented is referred to as the *document end*.

document duration

The difference between the end and the begin is referred to as the *document duration*.

This section is informative

Events and interactive timing

Begin and active end times in SMIL 2.1 can be specified to be relative to events that are raised in the document playback environment. This supports declarative, interactive timing. *Interactive* in this sense includes user events such as mouse clicks, events raised by media players like a `mediaComplete` event, and events raised by the presentation engine itself such as a `pause` event.

Syncbases

In scheduled timing, elements are timed relative to other elements. The syncbase for an element *A* is the other element *B* to which element *A* is relative. More precisely, it is the begin or active end of the other element. The syncbase is not simply a scheduled point in time, but rather a point in the time graph.

Sync arcs

"Sync-arc" is an abbreviation for "synchronization arc". Sync-arcs are used to relate nodes in the time graph, and define the timing relationship between the nodes. A sync-arc relates an element to its syncbase. The sync-arc may be defined implicitly by context, explicitly by Id-value or event name, or logically with special syntax.

Clocks

A Clock is a particular timeline reference that can be used for synchronization. A common example that uses real-world local time is referred to as **wall-clock** timing (e.g. specifying 10:30 local time). Other clocks may also be supported by a given presentation environment.

UTC: Coordinated Universal Time

"Universal Time" (abbreviated UT) is sometimes also referred to as "Greenwich Mean Time" (abbreviated GMT). The two terms are often used loosely to refer to time kept on the Greenwich meridian (longitude zero), five hours ahead of Eastern Standard Time. Times given in UT are almost always given in terms of a 24-hour clock. Thus, 14:42 is 2:42 p.m., and 21:17 is 9:17 p.m.

Hyperlinking and timing

A hyperlink into or within a timed document may cause a seek of the current presentation time or may activate an element (if it is not in violation of any timing model rules).

Activation

During playback, an element may be activated automatically by the progression of time, via a hyperlink, or in response to an event. When an element is activated, playback of the element begins.

Discrete and continuous Media

SMIL includes support for declaring media, using element syntax defined in "The SMIL Media Object Module". The media that is described by these elements is described as either *discrete* or *continuous*:

discrete

The media does not have intrinsic timing, or intrinsic duration. These media are sometimes described as "rendered" or "synthetic" media. This includes images, text and some vector media.

continuous

The media is naturally time-based, and generally supports intrinsic timing and an intrinsic notion of duration (although the duration may be indefinite). These media are sometimes described as "time-based" or "played" media. This includes most audio, movies, and time-based animations.

10.7.2 Timing concepts

Time containers

Time containers group elements together in time. They define common, simple synchronization relationships among the grouped child elements. In addition, time containers constrain the time that children may be active. Several containers are defined, each with specific semantics and constraints on its children.

Content/Media elements

SMIL timing and synchronization support ultimately controls a set of content or media elements. The content includes things like video and audio, images and vector graphics, as well as text or HTML content. SMIL documents use the SMIL media elements to reference this content. XML and HTML documents that integrate SMIL 2.1 functionality may use SMIL media elements and/or content described by the integrated language (e.g. paragraphs in HTML).

Basic markup

All elements - content/media as well as time containers - support timing markup to describe a begin time and a duration, as well as the ability to play repeatedly. There are several ways to define the begin time. The semantics vary somewhat depending upon an element's time container.

Simple and active durations

The time model defines two concepts of duration for each element - the simple duration and the active duration. These definitions are closely related to the concept of playing something repeatedly.

simple duration
: This is the duration defined by the basic begin and duration markup. It does not include any of the effects of playing repeatedly, or of fill. The simple duration is defined by the explicit begin and duration, if one is specified. If the explicit times are not specified, the simple duration is defined to be the implicit duration of the element.

active duration
: This is the duration during which the element plays normally. If no repeating behavior is specified, and end is not specified, the active duration is the same as the simple duration. If the element is set to play repeatedly, the simple duration is repeated for the active duration, as defined by the repeat markup.
The active duration does not include the effect of fill, except when the effect of the min attribute extends a shorter active duration. See The min and max attributes: more control over the active duration.

The constraints of a parent time container may override the duration of its children. In particular, a child element may not play beyond the simple end of the time container.

The terms for these durations can be modified with the Descriptive Terms for Times, to further distinguish aspects of the time graph.

Hard and soft sync

SMIL 1.0 introduced the notion of synchronization behavior, describing user agent behavior as implementing either "hard synchronization" or "soft synchronization". Using hard sync, the entire presentation would be constrained to the strict description of sync relationships in the time graph. Soft sync allowed for a looser (implementation dependent) performance of the document.

While a document is playing, network congestion and other factors will sometimes interfere with normal playback of media. In a SMIL 1.0 hard sync environment, this will affect the behavior of the entire document. In order to provide greater control to authors, SMIL 2.1 extends the hard and soft sync model to individual elements. This support allows authors to define which elements and time containers must remain in strict or "hard" sync, and which elements and time containers can have a "soft" or slip sync relationship to the parent time container.

See also the section: The syncBehavior, syncTolerance, and syncMaster attributes: controlling runtime synchronization.

Pruning and cutting off an interval

The concepts of interval *pruning* and *cutting off* are distinct and should not be confused.

In some cases, after an interval has been created, it must later be *pruned* (deleted/removed from the timegraph) as more information becomes known and semantic constraints must be applied. When an interval is *pruned*, it will not be shown, it will not raise begin or end events, and any associated instance times for syncbase time dependents must be removed from the respective instance times lists. It is as though the *pruned* interval had not been specified.

In other cases, especially related to negative begin times on parent time containers, a valid interval for a child may not be shown, even though it is otherwise legal with respect to the parent time constraints. These intervals are said to be *cut off*.

For example:

```
<par begin="-10s" dur="20s">
   <img id="slide1" src="slide1.jpg" dur="3s" />
   <img id="slide2" src="slide2.jpg" begin="slide1.end+3s" dur="10s" />
   <img id="note1" src="note1.jpg" begin="slide1.beginEvent" dur="20s" />
</par>
```

The "slide1" image will be *cut off*, but is not *pruned*. It is *cut off* because the par could not have been started 10s before its parent time container, and instead will be started at 0s into its parent time synced at 10s into its simple duration. The "slide1" image begins and ends before 10s into the par, and so cannot be shown and is *cut off*, Intervals that are *cut off* are not shown and do not raise begin or end events, but still create valid instance times for any syncbase time dependents. Thus, "slide2" *will* be shown (the interval is from minus 4 seconds to 6 seconds, document time, and so will be shown for 6 seconds, from 0 seconds to 6 seconds), but "note1" will not be shown.

10.8 Appendix A: SMIL Timing and Synchronization modules

This section is normative.

This section defines the sixteen SMIL 2.1 Timing Modules, which include the BasicInlineTiming module and fifteen other modules that combine to provide full SMIL 2.1 timing support. The separation of the SMIL 2.1 Timing modules is based on the inclusion of the syntactic expression of features using elements, attributes, and attribute values. Including a module in a profile adds both the syntax and associated semantics defined elsewhere in this specification to that profile.

AccessKeyTiming

This module defines the attribute value syntax for the **begin** and **end** attributes that allow elements to begin and end based upon the user actuating a designated access key.

Module dependencies

None.

Included features

begin and **end** with access key values.

Other module specific integration requirements

The access key requested by the author may not be made available by the player (for example it may not exist on the device used, or it may be used by the user agent itself). Therefore the user agent should make the specified key available, but may map the access key to a different interaction behavior. The user agent must provide a means of identifying the access keys that can be used in a presentation. This may be accomplished in different ways by different implementations, for example through direct interaction with the application or via the user's guide.

BasicInlineTiming

This module defines the attributes that make up basic timing support for adding timing to XML elements.

Module dependencies
None.
Included features
dur with all allowed values, and **begin** and **end** attributes with simple offset values, and "indefinite".
Other module specific integration requirements
None.

BasicTimeContainers
This module defines basic time container elements, attributes that describe an element's display behavior within a time container, and end conditions for time containers.

Module dependencies
None.
Included features
par, **seq** elements, **fill**, **endsync** attributes.
Other module specific integration requirements
`fill=transition` is only supported when BasicTransitions or InlineTransitions is included in the language profile. If FillDefault is not included in the profile, `fill=default` is interpreted the same as `fill=auto`.

EventTiming
This module defines the attribute value syntax for **begin** and **end** attributes that allow elements to begin and end in response to an event.

Module dependencies
None.
Included features
begin and **end** with event values.
Other module specific integration requirements
None. A Host language may specify that it does not support offsets on event values.

ExclTimeContainers
This module is depreciated in SMIL 2.1.
BasicExclTimeContainers
This module is new to SMIL 2.1. It includes a time container that defines a mutually exclusive set of elements and describes the 'stop' interrupt semantic among these elements.

Module dependencies
None.
Included features
excl element, **fill** and **endsync** attributes.
Other module specific integration requirements
`fill="transition"` is only supported when BasicTransitions or InlineTransitions is included in the language profile. If FillDefault is not included in the profile, `fill="default"` is interpreted the same as `fill="auto"`.

BasicPriorityClassContainers

This module is new to SMIL 2.1. It includes a child element for the **excl** that is used to describe interrupt semantics among group of children of the the exclusive element.

Module dependencies

The BasicExclTimeContiners module must be included in a profile containing the BasicPriorityClassContainers module

Included features

priorityClass element.

Other module specific integration requirements

None.

FillDefault

This module defines syntax for specifying default display behavior for elements.

Module dependencies

BasicTimeContainers or ExclTimeContainers or TimeContainerAttributes.

Included features

fillDefault attribute.

Other module specific integration requirements

`fill=transition` is only supported when BasicTransitions or InlineTransitions is included in the language profile.

MediaMarkerTiming

This module defines the attribute value syntax for the **begin** and **end** attributes that allow elements to begin and end based upon markers contained in the source content.

Module dependencies

None.

Included features

begin and **end** with media marker values.

Other module specific integration requirements

None.

MinMaxTiming

This module defines the attributes that allow setting minimum and maximum bounds on element active duration.

Module dependencies

None.

Included features

The **max** and **min** attributes.

Other module specific integration requirements

None.

MultiArcTiming

This module extends the attribute value syntax for the **begin** and **end** attributes to allow multiple semicolon-separated values. Any combination of the simple **begin** and **end** value types provided by the other timing modules included in the profile are allowed.

Module dependencies

At least one of: AccessKeyTiming, BasicInlineTiming, EventTiming, MediaMarkerTiming, RepeatValueTiming, SyncbaseTiming, WallclockTiming.

Included features

Any combination of the individual **begin** and **end** attribute values included in the profile, separated by semicolons.

Other module specific integration requirements

None.

RepeatTiming

This module defines the attributes that allow repeating an element for a given duration or number of iterations.

Module dependencies

None.

Included features

The **repeatDur**, **repeatCount** , and **repeat** attributes.

Other module specific integration requirements

repeat is deprecated and only requires inclusion in SMIL Host Language conformant profiles.

RepeatValueTiming

This module defines the attribute value syntax for **begin** and **end** attributes that allow elements to begin and end in response to repeat events with a specific iteration value.

Module dependencies

None.

Included features

begin and **end** with repeat values.

Other module specific integration requirements

None.

RestartDefault

This module defines syntax for specifying default restart semantics for elements.

Module dependencies

RestartTiming.

Included features

restartDefault attribute.

Other module specific integration requirements

None.

RestartTiming

This module defines an attribute for controlling the begin behavior of an element that has previously begun.

Module dependencies

None.

Included features

restart attribute.

Other module specific integration requirements

If this module is not included, the integrating profile must define the semantics of attempting to restart and element that has already begun.

SyncBehavior

This module defines syntax for specifying the runtime synchronization behavior among elements.

Module dependencies

BasicTimeContainers or ExclTimeContainers or TimeContainerAttributes.

Included features

syncBehavior, **syncTolerance** attributes.

Other module specific integration requirements

None.

SyncBehaviorDefault

This module defines syntax for specifying default synchronization behavior for elements and all descendents.

Module dependencies

SyncBehavior.

Included features

syncBehaviorDefault, **syncToleranceDefault** attributes.

Other module specific integration requirements

None.

SyncbaseTiming

This module defines the attribute value syntax for the **begin** and **end** attributes that allow elements to begin and end relative to each other.

Module dependencies

None.

Included features

begin and **end** with syncbase values.

Other module specific integration requirements

None.

SyncMaster

This module defines syntax for specifying the synchronization master for a timeline.

Module dependencies

SyncBehavior.

Included features

syncMaster attribute.

Other module specific integration requirements

None.

TimeContainerAttributes

This module defines attributes for adding time container support to any XML language elements.

Module dependencies

None.

Included features

timeContainer, **timeAction**, **fill** and **endsync** attributes.

Other module specific integration requirements

The profile must define on what elements these attributes are legal.
`fill=transition` is only supported when BasicTransitions or

InlineTransitions is included in the language profile. If FillDefault is not included in the profile, `fill=default` is interpreted the same as `fill=auto`.

WallclockTiming

This module the attribute value syntax for the begin and end attributes that allow elements to begin and end relative to real world clock time.

Module dependencies

None.

Included features

begin and end with wallclock times.

Other module specific integration requirements

None.

10.9 Appendix B: Annotated examples

10.9.1 Example 1: Simple timing within a Parallel time container

This section includes a set of examples that illustrate both the usage of the SMIL syntax, as well as the semantics of specific constructs. This section is informative.

Note: In the examples below, the additional syntax related to layout and other issues specific to individual document types is omitted for simplicity.

All the children of a par begin by default when the par begins. For example:

```
<par>
   <img id="i1" dur="5s"  src="img.jpg" />
   <img id="i2" dur="10s" src="img2.jpg" />
   <img id="i3" begin="2s" dur="5s" src="img3.jpg" />
</par>
```

Elements "i1" and "i2" both begin immediately when the par begins, which is the default begin time. The active duration of "i1" ends at 5 seconds into the par. The active duration of "i2" ends at 10 seconds into the par. The last element "i3" begins at 2 seconds since it has an explicit begin offset, and has a duration of 5 seconds which means its active duration ends 7 seconds after the par begins.

10.9.2 Example 2: Simple timing within a Sequence time container

Each child of a seq begins by default when the previous element ends. For example:

```
<seq>
   <img id="i1" begin="0s" dur="5s" src="img1.jpg" />
   <img id="i2" dur="10s" src="img2.jpg" />
   <img id="i3" begin="1s" dur="5s" src="img3.jpg" />
</seq>
```

The element "i1" begins immediately, with the start of the seq, and ends 5 seconds later. Note: specifying a begin time of 0 seconds is optional since the default begin offset is always 0 seconds. The second element "i2" begins, by default, 0 seconds after the previous element "i1" ends, which is 5 seconds into the seq. Element "i2" ends 10 seconds later, at 15 seconds into the seq. The last element, "i3", has a begin offset of 1 second specified, so it begins 1 second after the previous element "i2" ends, and has a duration of 5 seconds, so it ends at 21 seconds into the seq.

10.9.3 Example 3: excl time container with child timing variants

1. Exclusive element, children activated via link-based activation:

```
<par>
    <excl>
        <par id="p1">
        ...
        </par>
        <par id="p2">
        ...
        </par>
    </excl>
    <a href="p1"><img src="Button1.jpg"/></a>
    <a href="p2"><img src="Button2.jpg"/></a>
</par>
```

   This example models jukebox-like behavior. Activating the first image hyperlink activates the media items of parallel container "p1". If the link on the second image is traversed, "p2" is started (thereby deactivating "p1" if it would still be active) from time 0.

2. Exclusive element combined with event-based activation:

```
<smil ...>
...
<par>
    <excl>
        <par begin="btn1.activateEvent">
        ...
        </par>
        <par begin="btn2.activateEvent">
        ...
        </par>
    </excl>
    <img id="btn1" src=... />
    <img id="btn2" src=... />
</par>
...
<smil>
```

   The same jukebox example, using event-based activation.

3. Exclusive element using scheduled timing:

```
<excl>
    <ref id="a" begin="0s" ... />
    <ref id="b" begin="5s" ... />
</excl>
```

   In the example above, the beginning of "b" deactivates "a" (assuming that a is still active after 5 seconds). Note that this could also be modeled using a sequence with an explicit duration on the children. While the scheduled syntax is allowed, this is not expected to be a common use-case scenario.

10.9.4 Example 4: default duration of discrete media

For simple media elements (i.e., media elements that are not time containers) that reference discrete media, the implicit duration is defined to be 0. This can lead to surprising results, as in this example:

```
<seq>
   <img src="img1.jpg" />
   <video src="vid2.mpg" />
   <video src="vid3.mpg" />
</seq>
```

The implicit syncbase of a sequence is defined to be the effective active end of the previous element in the sequence. In the example, the implicit duration of the image is used to define the simple and active durations. As a result, the default begin of the second element causes it to begin at the same time as the image. Thus, the image will not show at all! Authors will generally specify an explicit duration for any discrete media elements.

10.9.5 Example 5: end specifies end of active dur, *not* end of simple dur

There is an important difference between the semantics of end and dur. The dur attribute, in conjunction with the begin time, specifies the simple duration for an element.

This is the duration that is repeated when the element also has a repeat behavior specified. The attribute end on the other hand overrides the active duration of the element. If the element does not have repeat behavior specified, the active duration is the same as the simple duration. However, if the element has a repeat behavior specified, then the end will override the repeat, but will not affect the simple duration. For example:

```
<html ...>
...
<seq repeatCount="10" end="stopBtn.click">
   <img src="img1.jpg" dur="2s" />
   <img src="img2.jpg" dur="2s" />
   <img src="img3.jpg" dur="2s" />
</seq>
...
</html>
```

The sequence will play for 6 seconds on each repeat iteration. It will play through 10 times, unless the user clicks on a "stopBtn" element before 60 seconds have elapsed.

10.9.6 Example 6: DOM-initiated timing

When an implementation supports the DOM methods described in this document, it will be possible to make an element begin or end the active duration using script or some other browser extension. When an author wishes to describe an element as interactive in this manner, the following syntax can be used:

```
<audio src="song1.au" begin="indefinite" />
```

The element will not begin until the `beginElement()` method is called.

10.10 Appendix C: Differences from SMIL 1.0

SMIL 1.0 defines the model for timing, including markup to define element timing, and elements to define parallel and sequence time containers. This version introduces some syntax variations and additional functionality, including:

- A new time container for hypermedia interactions
- Additional control over the repeat behavior
- A syntax for interactive (event-based) timing
- Change in constraints on sync-arcs
- A means of specifying a logical time-base relationship
- Support for wall-clock timing
- Support for time manipulations
- Fill is now allowed on time containers as well as "leaf" elements

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

10.11 Appendix D: Unifying event based and scheduled timing

A significant motivation for SMIL 2.1 is the desire to integrate declarative, determinate scheduling with interactive, indeterminate scheduling. The goal is to provide a common, consistent model and a simple syntax.

Note that "interactive" content does not refer simply to hypermedia with support for linking between documents, but specifically to content within a presentation (i.e. a document) that is *activated* by some interactive mechanism (often user-input events, but including local hyperlinking as well).

SMIL 2.1 describes extensions to SMIL 1.0 to support interactive timing of elements. These extensions allow the author to specify that an element should begin or end in response to an event (such as a user-input event like "activateEvent" or "click"), or to a hyperlink activation, or to a DOM method call.

The syntax to describe this uses event-value specifications and the special argument value "**`indefinite`**" for the **begin** and **end** attribute values. Event values describe user interface and other events. If an element should only begin (or end) with a DOM method call, the **begin** and **end** attributes allow the special value "**`indefinite`**" to indicate this. Setting **begin**=`"indefinite"` can also be used when a hyperlink will be used to begin the element. The element will begin when the hyperlink is actuated (usually by the user clicking on the anchor). It is not possible to control the active end of an element using hyperlinks.

10.11.1 Background

SMIL 2.1 represents an evolution from earlier multimedia runtimes. These were typically either pure, static schedulers or pure event-based systems. Scheduler models present a linear timeline that integrates both discrete and continuous media. Scheduler models tend to be good for storytelling, but have limited support for user-interaction. Event-based systems, on the other hand, model multimedia as a graph of event bindings. Event-based systems provide flexible support for user-interaction, but generally have poor scheduling facilities; they are best applied to highly interactive and experiential multimedia.

The SMIL 1.0 model is primarily a scheduling model, but with some flexibility to support continuous media with unknown duration. User interaction is supported in the form of timed hyperlinking semantics, but there was no support for activating individual elements via interaction.

10.11.2 Modeling interactive, event-based content in SMIL

To integrate interactive content into SMIL timing, the SMIL 1.0 scheduler model is extended to support several new concepts: *indeterminate timing* and *event-activation*.

With *indeterminate timing*, an element has an undefined **begin** or **end** time. The element still exists within the constraints of the document, but the **begin** or **end** time is determined by some external *activation*. Activation may be event-based (such as by a user-input event), hyperlink based (with a hyperlink targeted at the element), or DOM based (by a call to the `beginElement()` or `beginElementAt()` methods). From a scheduling perspective, the time is described as *unresolved*.

The event-activation support provides a means of associating an event with the **begin** or **end** time for an element. When the event is raised (e.g. when the user clicks on something), the associated time is *resolved* to a *determinate* time. The actual **begin** or **end** time is computed as the time the event is raised plus or minus any specified offset.

The computed time defines the synchronization for the element relative to the parent time container. It is possible for the computed **begin** or **end** time to occur in the past, e.g. when a negative offset value is specified, or if there is any appreciable delay between the time the event is raised and when it is handled by the SMIL implementation. See also the section Handling negative offsets for begin.

Note that an event based **end** will not be activated until the element has already begun. Any specified **end** event is ignored before the element begins.

The constraints imposed on an element by its time container are an important aspect of the event-activation model. In particular, when a time container is itself inactive (e.g. before it begins or after it ends), no events are handled by the children. If the time container is frozen, no events are handled by the children. No event-activation takes place unless the time container of an element is active. For example:

```
<smil ...>
...
<par begin="10s" dur="5s">
   <audio src="song1.au" begin="btn1.activateEvent" />
</par>
...
</smil>
```

If the user activates (e.g., clicks on) the "btn1" element before 10 seconds, or after 15 seconds, the audio element will not play. In addition, if the audio element begins but would extend beyond the specified active end of the **par** container, it is effectively cut off by the active end of the **par** container.

See also the discussion of Event sensitivity.

previious next contents