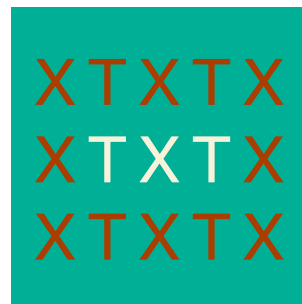


9

Timed Text in SMIL



In spite of the high profile of media types such as video, audio and images, many SMIL presentations rely extensively on text content. Text can be used for incidental labels, or as subtitles and captions that accompany other media objects. Unlike most XML text formats, text content in SMIL is not only constrained by the need to support presentation styles and layout, it is also constrained by the temporal context of the presentation. Text must appear simultaneously with related objects, and it must disappear when they are finished. Perhaps more importantly, the text content may need to be synchronized with a particular part of an accompanying media object, such as when the text is used for captions.

This chapter describes a timed-text content format that is intended to be embedded in-line in a SMIL document. This format, which is called *smilText*, balances the need for text styling with the requirement for an efficient representation that can be easily parsed and scheduled at runtime. *SmilText*, which can also be streamed from an external file, also defines a set of motion attributes to support popular types of text content movement.

We begin this section with a discussion of the requirements that guided the development of *smilText*. We then describe the general timing, styling and motion elements and attributes defined for *smilText*. We conclude with a series of examples of how *smilText* can be used within a document and outside the document as an external text container.

9.1 Text Content in a SMIL Presentation

SMIL integrates a collection of audio, graphics, image, text, and video media items into a single presentation. Since most media items are by nature very large — they consist of sampled data that can run into the gigabytes — it has been common wisdom during SMIL's development to include media objects by referencing external container formats.

While the clean separation of content from document structure is a valid concept, it often brings with it an increased authoring burden for applications in which extensive use is made of relatively lightweight text objects. Instead of maintaining (or generating) a single document, many source files need to be defined: one for the integrating structure and one each for the text fragments. It



also brings media access problems — especially in mobile environments — where multiple transfer sessions need to be initiated to fetch a few bytes of content. While this is defensible for large, typically binary media objects, it is less appropriate for text data.

One approach for supporting in-line text was to embed an existing format into SMIL, such as HTML. This had the practical disadvantage of requiring implementation of an extremely feature-rich text engine to manage styling and layout, where only relatively simple text support was required. More importantly, it did not provide any support for a key requirement of text in a SMIL documents: intra-text timing and extra-text synchronization with other objects.

To meet the needs of SMIL, it was decided to adapt an existing draft W3C timed-text specification called DFXP. The SMIL version was based on experiences with public and proprietary formats (most notably, Real's *RealText*), with the extra constraint that the format needed to be extremely easy to parse, schedule and render on a wide range of multimedia implementation platforms (including low-powered mobile devices and low-end set-top boxes).

9.1.1 Applications for smilText

The functionality proposed for *smilText* is based on a set of general use cases that are common when authoring multimedia presentations. These use cases correspond to the following classes of functionality:

- *Headline Text*: this class requires support for relatively short strings of text that can be stylistically differentiated from other text. This differentiation may be based on size, color or placement (centered vs. side-aligned).
- *Labels*: this class requires support for relatively short strings of text. The text may need to be of varying sizes for various uses, such as photo captions or button labels. Significant text formatting is typically not required.
- *Captions/subtitles*: this class requires support for multi-line strings of text that provide a visual representation of both spoken text and audio cues in content. It may also require style clues for speaker identification. This class will require light formatting (alignment, line breaks) and tight temporal coupling with other objects in the presentation. Historically, captions and subtitles have been geared for use by the hearing impaired.
- *Foreign-language subtitles*: this class is similar to those described in the *captions/subtitles* class, except that audio cues and speaker identification are typically not presented; there is an assumption that the foreign-language reader can see and hear the speakers and cues.
- *Time-constrained moving text*: this class requires support for marquee-style crawling or credits-style rolling text. The content may or may not be directly related to other content in the presentation (such as an independent news crawl under an unrelated video). Such moving text needs to have a rate associated with it that is not related to its internal content. It may also need to loop based on external (to the text) factors.



TXTXT
TXTXT
TXTXT
TXTXT

- *Inter-object triggered text*: this class requires the specification of text fragments that are either triggered by external media (such as a particular frame in a video) or that trigger external objects (such as the sound of a gong when a certain word appears in the text). The level of granularity may be at the word or phrase level, rather than at the text object level.
- *Conditional timed text*: this class requires support for the conditional specification of text into a rendering area. The conditions may depend on the state of variables in the integrating document, and may be dynamically evaluated during the presentation of the text.
- *Static block text*: this class requires support for relatively large amounts of formatted text. The text block may consist of informative information that is conditionally presented during a presentation, such as help text or content descriptions.

SmilText was designed to provide substantial support for the first seven classes. For the eighth class of text, an external file in a static container, such as (X)HTML, was deemed more appropriate.

In the seven SMIL-relevant use cases, text content plays a subordinate role to other media in the presentation. The subordinate role of text does not mean that it is unimportant. The appropriate application of text introduces clarity that might otherwise be missing. It is often said that a “picture is worth a thousand words”; what is often forgotten is that without some form of descriptive (text) context, a picture alone may be worthless!

9.1.2 Existing Support for Text in SMIL

All versions of SMIL have had support for the `<text>` element, which is defined as an alias of the generic SMIL `<ref>` media reference element. A typical use of the `<text>` element is:

```
<text region="Title" src="Headline.html" dur="3s" />
```

This construct causes an external HTML renderer to be activated. One problem with this approach is that activating an HTML renderer is typically a heavyweight operation that, on low-resource platforms, may take longer than the 3 seconds allotted for the actual text rendering.

Users new to SMIL are often surprised that the text element does not have a content model—that is, an ability to specify the content text along with the element, such as in:

```
<text region="Title" dur="10s" >
  A Story About a Dog
</text>
```

The content model was rejected early in the development of SMIL 1.0. The idea then was that a direct separation of structure and timing from content was an important concept to hammer home. As a result of many user requests, this stance has been changed in SMIL 3.0.

T X T X T
T X T X T
T X T X T
T X T X T

More advanced users of SMIL found that they were able to insert in-line text content into the SMIL file using a *data URL*, such as:

```
<text region="Title" src="data: , A%20Story%20About%20a%20Dog" dur="3s" />
```

The restrictive syntax of this approach, plus the limited styling options available, make it a less-than-optimal way of including incidental text content into a SMIL presentation. The main benefit of this approach is processing speed.

Adding Time to Local Text

In SMIL 3.0, *smilText* functionality is partitioned across three modules: *BasicText*, *TextStyling* and *TextMotion*. Of these, all *smilText* engines need to support the functionality in *BasicText*. Support for text styling and text motion is on a best-effort basis.

The most important basic temporal and spatial concepts in *smilText* are illustrated in Example 9-1. The **version** and **baseProfile** declarations indicate that this is a SMIL 3.0 example, and therefore can only be rendered on a SMIL 3.0 user agent. Line 5 defines a region that will be used to contain the actual text. Normally, the region will define an appropriate background color and perhaps default styling information, but these have been left out of this example for simplicity. The actual *smilText* object is contained on lines 11-20.

Line 11 defines the basic temporal scope of the *smilText* object (in this case, 10 seconds) and it defines where the contents will be rendered (in the region captions). By default, any text content will be wrapped within this region — that is, text strings that take up more space than is available will be broken at a word

```
1 <smil xmlns="http://www.w3.org/ns/SMIL"
   version="3.0" baseProfile="Language">
2   ...
3   <layout>
4     ...
5     <region xml:id="captions" width="120px" height="30px"/>
6     ...
7   </layout>
8 </head>
9 <body>
10  ...
11  <smilText region="captions" dur="10s">
12    This is the first
13    <tev begin="3s"/>
14    line of captions.
15    <clear begin="5s"/>
16    <tev begin="6s"/>
17    Here is the second (and particularly long and verbose) line
18    <tev next="2s"/>
19    of captions.
20    <clear next="1s"/>
21  </smilText>
22  ...
23 </body>
24 </smil>
```

Example 9-1. The basics of *smilText*.

boundary and placed on the following line. This behavior can be changed using the `textWrapOption` attribute, discussed later in this chapter.

The content in line 12 is rendered immediately when the `<smilText>` element is activated. The *timing event value* `<tev>` element on line 13 says: activate me three seconds after the start of the `<smilText>` element. The `<tev>` element does not contain any content — it is simply a marker in the text that defines a temporal moment. Once this moment occurs, the next bit of content is processed, which in this case is the second part of the caption on line 14. By default, this is placed at the end of the previous caption, using spacing conventions defined by the `xml:space` attribute. The net effect of lines 12-13 is that a caption is written in two parts, with a three-second interval between parts.

The `<clear>` element on line 15 explicitly clears the region five seconds into the presentation. It is also a temporal marker, just like the `<tev>`, but with the side effect that the region is erased.

Line 16 defines another temporal moment, this one effectively one second after the `<clear>` (or, six seconds after the start of the `<smilText>` object). This effectively inserts a delay interval, after which the next text string is rendered: the long and verbose caption line shown.

Up to this point, all timing on event markers has occurred as *absolute* references to the start of the `<smilText>` element. This gives absolute control, but can also be an absolute pain in the neck if you need to insert a new caption or adjust timing within an existing block. For this reason, *smilText* also allows relative start times to be defined using the `next` attribute. The `<tev>` on line 18 tells the *smilText* engine to wait for two seconds after the start of the previous `<tev>` or `<clear>` marker before continuing. When this occurs, the final text portion is rendered. One second later, the region is once again cleared.

In *smilText*, relative and absolute timing can be mixed within one element, but if there are inconsistencies or errors, a strict linear timing order will be followed. Consider the timing inconsistency in following fragment:

```
<smilText region="Title" dur="10s" >
  Here is some text.
  <tev next="3s" />
  Here is some more text.
  <clear begin="2s" />
</smilText>
```

This fragment defines one relative temporal moment 3 seconds after the start of the `<smilText>` object, followed by an absolute offset 2 seconds after the start of the `<smilText>` object. In this case, the clear operation will occur at the first valid moment after the absolute time specified. The content “Here is some more text.” will be displayed and then immediately cleared. Any relative timing after the `<clear>` will occur based on its actual rendering time.

In addition to absolute and relative begin times, *smilText* also supports interactive begins, either based on a standard SMIL UI event, or the beginning or ending of another SMIL element. We return to this later in the chapter.



T X T X T
T X T X T
T X T X T
T X T X T

Summary of *smilText* Timing Features

The linear characteristics of text — plus the potential needs of an external streaming text container — meant that adding time to text within a SMIL file needed a simplified approach to text-string timing. Reusing existing SMIL timing constructs, such as `<seq>` and `<par>` would have required imposing a wealth of restrictions on `begin/end` timing, and it would have added to the complexity of the text rendering engine. Instead, a streamlined timing model was developed that was designed to compliment SMIL timing.

Five features describe the basic temporal functionality of *smilText*:

- 1) *smilText* supports the atomic display of a block of text based on the timing constraints defined on the `<smilText>` element. This means that a `<smilText>` element will be given a begin time and a duration or end time that is defined by the SMIL environment. Within this time interval, the content within the `<smilText>` element will be rendered.
- 2) Within the interval defined in (1), text will appear either as a whole or as a sequence of text fragments that are appended to the rendering area based on internal timing directives.
- 3) The timing attributes for individual text fragments allow for the specification of absolute, relative and (external) event-based timing constraints.
- 4) During the display of text fragments, it is possible to explicitly clear the rendering area.
- 5) During the entire interval defined by (1), it is possible to have the text move (content crawl or roll) across the display area, if this is within the power of the rendering device. If not, text motion is ignored.

There are also several things that *smilText* explicitly *does not* support. These include:

- 1) the removal of selected fragments of text while retaining others,
- 2) the automatic reflow of text within a block based on removal of embedded fragments, and
- 3) support for out-of-order temporal specification of text within a block.

The result of these goal and non-goals is the definition of an efficient and easy-to-implement text format.

The generic `<smilText>` element is defined in the following section. Subsequent sections define the internal structuring components within the *smilText* format.

9.2 *smilText* Elements and Attributes

The `<smilText>` element is similar to other SMIL media objects defined in Chapter 5: *Referencing External Media Objects*, with the exception that it defines renderable content explicitly within the SMIL document. The presence of content means that, unlike other media objects, a `<smilText>` element also accepts a set of text styling and motion control attributes.



TXTXT
TXTXT
TXTXT
TXTXT

9.2.1 smilText Elements

The *smilText* infrastructure defines three top-level elements.

Element: <smilText>

The **<smilText>** is a standard SMIL media object, with the exception that it has a richer content model. It takes the same attributes as all SMIL media objects, and adds attributes for *text rendering*, *text styling* and *text motion*. The **<smilText>** parents are the same as for all other media objects. The children are **<metadata>** elements, elements to control rendering (**<tev>**, **<clear>** and **
), elements that support styling (<div>**, **<p>** and ****), and the actual text content that is to be rendered via the **<smilText>** element.

<smilText>	
attributes	All Media Attributes (see page 92)
	TextRendering Attributes
	TextStyling Attributes
	TextMotion Attributes
parents	All Media Parents (see page 92)
children	<textStyle>
	Text Rendering Elements
	Text Segmentation Elements
	smilText content
	<metadata>

Table 9-1. The <smilText> element.

Element: <textStyle>

The **<textStyle>** element is a logical container for grouping sets of text attributes. The attributes may include the style attributes and the motion attributes described below. If a particular element appears more than once, then only the lexically last definition is used. Styles defined on a **<textStyle>** element will be overridden by any definitions of the same attribute on the media.

The **xml:id** used on the **<textStyle>** element can be referenced as a value for the **textStyle** attribute on a **<smilText>** element, or on the **<div>**, **<p>** or **** segmentation elements. If an attribute defined within a **<textStyle>** element is redefined on the target element, the latter definition is used.

Element: <textStyling>

The **<textStyling>** element is detailed in Table 9-3. It is a logical container for grouping a set of **<textStyle>** elements. The **<textStyling>** element does not compound or cascade style definitions. It is simply a syntactical framework for collecting style templates.

<textStyle>	
attributes	Core
	118N
	TextRendering Attributes
	TextStyling Attributes
	TextMotion Attributes
parents	<textStyling>
children	<metadata>

Table 9-2. The <textStyle> element/

T X T X T
T X T X T
T X T X T
T X T X T

The `<textStyling>` attribute may only be used in the `<head>` section. Styles may not be composed dynamically in the document body.

9.2.2 smilText Attributes

The `<smilText>` element takes all of the media control attributes defined on page 92. This includes timing, layout and system test attributes. In addition, the element allows a number of rendering, styling and (if supported) motion attributes to be initialized for use within the object. All of these attributes may be predefined by a `<textStyle>` element within the `<head>` section. They may also be overridden or modified within the `<smilText>` content.

The following attributes may be used on the `<smilText>` element:

- *Text Rendering Control Attributes*: these attributes define general alignment, text wrapping and writing mode properties of the text content. They are `xml:space`, `textAlign`, `textWrapOption` and `textWritingMode`;
- *Text Styling Attributes*: these attributes define general styling control for the text content. They define text color (`textBackgroundColor`, `textColor`) and the font used (`textFontFamily`, `textFontSize`, `textFontWeight`, `textFontStyle`). The styling attributes also include the general style definition attribute `textStyle`;
- *Text Motion Attributes*: these attributes define motion-related control for the text content. This includes the general motion mode (`textMode`), the motion rate (`textRate`) and the behavior of text after motion is completed (`textConceal`).

The definition of each of the attributes is given in the following sections.

Many of the *smilText* styling and motion attributes define complex behaviors that may not be supported on low-performance platforms. For this reason, the behavior of these attributes has been defined as *best effort*: if a particular attribute can't be supported, it will be ignored.

9.3 Rendering Control Elements and Attributes

The *smilText* architecture is based on the definition of a series of *temporal moments* within a linear stream of content. Each temporal moment defines a point at which a *smilText* renderer will perform a successor operation, such as rendering text or clearing the region. The behavior associated with the definition of the temporal moments, plus the basic forms of text control are defined by the rendering control elements and attributes described in this section.

<textStyling>	
attributes	Core
	118N
parents	<head>
children	<metadata>
	<textStyle>

Table 9-3. The `<textStyling>` element.

TXTXT
TXTXT
TXTXT
TXTXT

9.3.1 Text Rendering Control Elements

The two temporal rendering control elements are `<tev>` and `<clear>`. Each of these must have a timestamp that indicates the moment at which they become active. A third rendering control element is `
`; it allows an explicit line break to be added within a block of text. It *does not* have timing markup.

Element: `<tev>`

The `<tev>` — or *timing event value* — element defines a temporal moment within a block of *smilText*. It does not, in and of itself, cause rendering to take place: it simply defines a moment at which the *smilText* scheduler will begin to evaluate the following bits of content in the element definition. For SMIL 3.0, the only child content *within* a `<tev>` is the `<metadata>` element.

The `<tev>` element accepts all **Core** and **I18N** attributes. It also takes the **begin** and **next** timing attributes. The **begin** specifies an absolute timing reference from the start of the `<smilText>` element; the **next** attribute specifies a relative offset from the first lexically previous `<tev>`, `<clear>` or `<smilText>` element. You may use one or the other on any interior timing element, but not both.

<tev>	
attribute	Core
	I18N
	begin
	next
parents	<smilText>
children	<metadata>

Table 9-4. The `<tev>` element.

Element: `<clear>`

The `<clear>` element defines a temporal moment within a block of *smilText*. It is identical in structure and purpose to the `<tev>` element, with one important exception: using the `<clear>` element clears the rendering region containing the *smilText* content.

The `<clear>` element could also have been replaced by allowing the `<tev>` element to have a *clear* attribute. This option was not used to allow future versions of `<tev>` to contain a structured content model.

<clear>	
attribute	Core
	I18N
	begin
	next
parents	<smilText>
children	<metadata>

Table 9-5. The `<clear>` element.

T X T X T
T X T X T
T X T X T
T X T X T

Element: `
`

The `
` element indicates that a forced line break should be placed within a block of *smilText*.

The `
` element does not contain any timing markup: the **begin** and **next** attributes are *not* allowed. The element accepts `<metadata>` as a child.

attribute	Core
	I18N
parents	<smilText>
children	<metadata>

Table 9-6. The `
` element.

9.3.2 Text Rendering Control Attributes

The following four rendering control attributes can be expected to be supported by all *smilText* implementations. They provide basic scheduling and rendering support for incidental text.

Attribute: begin

This attribute specifies the temporal moment of the `<tev>` or `<clear>` element. It contains a SMIL time value list of at most one non-negative CSS2 clock value and one or more event-based times. If only event values are used, the `<tev>/<clear>` element will block until the event is resolved (or the parent time container ends). All times specified by `begin` are *absolute* offsets from the start of the `<smilText>` element. If time values are encountered after the specified offset, they will be activated immediately.

Another SMIL element may use the resolved begin time identified by this element as an event trigger. In this case, the `<tev>/<clear>` element containing the `begin` attribute needs to also define an ID attribute.

When used in an external file, event timing is not supported.

Attribute: next

This attribute specifies the temporal moment of the `<tev>` or `<clear>` element. It contains a single non-negative CSS2 clock offset. Event-based timing may not be used. All times specified by `next` are *relative* offsets from the start of the lexically previous `<tev>/<clear>/<smilText>` element. The actual start time of the previous element is used.

Another SMIL element may use the resolved begin time identified by this element as an event trigger. In this case, the `<tev>/<clear>` element containing the `next` attribute needs to also define an ID attribute.

Attribute: textWrapOption

This attribute defines the rendering behavior for text lines that do not fit within the rendering space of the SMIL region. The values may be `wrap` (which is the default) or `noWrap`. A value of `inherit` is allowed, which permits a previous setting to be used on an interior object.

The wrapping of text is a complex business, requiring knowledge of language hyphenation rules and breaking conventions. This complexity is beyond the scope of *smilText*. As a result, only relatively simple text wrapping can be expected — usually only on word boundaries.

Attribute: xml : space

In *smilText*, extra white space between characters is processed according to the model defined in XML 1.1. The `xml : space` attribute allows the values:

- `default` (default): whitespace is collapsed according to the semantics defined by XML 1.1 (section 2.10). Line breaks are suppressed (unless the

TXTXT
TXTXT
TXTXT
TXTXT

`
` element is used) and treated as a space character. Multiple occurrences of white space are collapsed into a single white space character.

- `preserve`: As defined by XML 1.1, whitespace in a block of text is treated as explicit character content. Any whitespace around linebreaks are maintained and linefeeds are respected.

The `preserve` value may yield strange results when used together with motion-based text.

9.4 Segmentation and Styling Elements and Attributes

9.4.1 Text Segmentation Elements

The *smilText* infrastructure provides support for optional text segmentation. A simple styling model, based on XSL, may be applied to individual segments. None of the segments contain timing markup in SMIL 3.0.

Element: `<div v>`

The `<div v>` element is a logical container for in-line formatting attributes. The `<div v>` element does not define any temporal semantics.

Use of the `<div v>` causes an implicit line break before the rendering of the element. User agents may supply an additional line of white space.

The parent may be the top-level `<smilText>` element, or it may be nested. It may contain other structuring containers, rendering control elements or text content.

The `<div v>` is similar to `<p>`, except that it may specify the `textAlign` attribute.

<code><div v></code>	
attributes	<i>Core, 118N</i>
	<code>textWrapOption</code>
	<code>xml:space</code>
	<i>Text Styling Attributes</i>
parents	<code><smilText></code> , <code><div v></code>
children	<code><metadata></code>
	Text Content
	<code><tev></code> , <code><clear></code> , <code>
</code>
	<code><div v></code> , <code><p></code> , <code></code>

Table 9-7. The `<div v>` element.

T X T X T
T X T X T
T X T X T
T X T X T

Element: `<p>`

The `<p>` element is a logical container for in-line formatting attributes. The `<p>` element does not define any temporal semantics within a `<smilText>` element. Use of the `<p>` causes an implicit line break before the rendering of the element. User agents may supply an additional line of white space or try to merge the space produced by a `<div v>`.

<code><p></code>	
attributes	<i>All <div v> attributes except <code>textAlign</code></i>
parents	<code><smilText></code> , <code><div v></code>
children	<code><metadata></code>
	Text Content
	<code><tev></code> , <code><clear></code> , <code>
</code>
	<code></code>

Table 9-8. The `<p>` element.

The `<p>` may be nested in a parent `<div>`, or the parent may be the top-level `<smilText>` element. The `<p>` may contain a `` structuring containers, it may contain rendering elements or it may contain text content. The `<p>` element may not be nested inside another `<p>` element.

Element: ``

The `` element is a logical container for in-line formatting attributes. The `` element does not define any temporal semantics within a `<smilText>` element.

Use of the `` does not cause an implicit line break before the rendering of the element.

The `` may be nested. It may also contain rendering elements or it may contain text content.

The main use of the `` element is to provide local styling. The text direction may be changed within a span, but not the writing mode applied to the entire (sub-)block.

<code></code>	
attributes	All <code><p></code> attributes except <code>textWritingMode</code>
	<code>textDirection</code>
parents	<code><smilText></code> , <code><div></code> , <code><p></code> , <code></code>
children	<code><metadata></code>
	Text Content
	<code><tev></code> , <code><clear></code> , <code>
</code>
	<code></code>

Table 9-9. The `` element.

9.4.2 Text Styling Attributes

The text styling support in *smilText* is the minimal functionality that meets the needs of the use cases described above. The attributes are grouped into *general rendering support* and *content styling support*.

General Rendering Attributes

The general rendering attributes are: `textMode`, `textAlign`, `textPlace`, `textWritingMode` and `textDirection`.

Attribute: `textMode`

This attribute specifies how each new fragment of content at a `<tev>` or `<clear>` is added to the rendering area. It is used on the `<smilText>` element to define behavior within a single `<smilText>` object rather than multiple `<smilText>` objects.

The values supported are: `append`, `replace` and `inherit`. The initial value is `append`, which is inherited by subsequent elements.

Only `append` mode is required for basic implementations. The values for this attribute are further extended by the text motion attributes (page 191).

Attribute: `textAlign`

This attribute specifies how text is aligned in a layout area, either as an absolute directive or relative to the direction defined by the `textWritingMode` attribute.



Permitted values are: {start | end | left | right | center | inherit}. The initial value is start, which is inherited by default.

If the primary writing mode is left-to-right, `textAlign="start"` is equivalent to `textAlign="left"`; if the writing mode is right-to-left languages, `textAlign="start"` is equivalent to `textAlign="right"`.

Given the implementation complexity of this attribute, not all players will support relative alignment in other than a left-to-right primary writing mode.

Attribute: textPlace

This attribute determines where a new block of text should be placed within the rendering region, both at the start of the object and after a `<clear>` element. The placement is defined in terms of the secondary writing direction. (For the standard `l r-tb` mode, `textPlace` impacts where the text is placed vertically.)

The values are: {start | center | end | inherit}. The start value means that new content is placed at the starting edge of the secondary writing direction (the top, in `l r-tb`), and then flowed downward. The end value means that the content is placed at the ending edge in the secondary writing direction (bottom, in the case of `l r-tb`), and then pushed upward when new linebreaks are encountered.

The initial value is start, which is inherited by subsequent elements.

Attribute: textWritingMode

This attribute provides a hint on the desired primary and secondary writing direction of the text placed in a region. The primary direction is used for successive characters, while the secondary direction is used for successive lines.

SMIL 3.0 allows the following combinations to be defined: {`l r-tb` | `r l-tb` | `tb-r l` | `tb-l r` | `l r` | `r l` | inherit}. The initial value is `l r-tb`, which is inherited by subsequent elements. Note that reliable support for other than `l r-tb` may be difficult to find in initial implementations.

This attribute may be ignored if text motion is used. A `` element may override the specified writing mode using the `textDirection` attribute.



T X T X T
T X T X T
T X T X T
T X T X T

Attribute: textDirection

This attribute allows the direction of text defined within a `` to be specified. Only simple text direction control is available in *smilText*; for more elaborate text processing (including `tb-l r`), an external text formatting language should be used.

Permitted values are: {`l t r` | `r t l` | `l t r o` | `r t l o` | inherit}. The `l t r o` and `r t l o` values indicate that any direction information provided when using the *Unicode Bidirectional* (UnicodeBiDi) algorithm should be overridden. The initial value is `l t r`, which may be inherited.

Support for override behavior is on a best-effort basis in SMIL 3.0.

The consistent use of `textDirection` and `textWritingMode` with Unicode-BiDi should lead to predictable results for non-Western languages.



Content Styling Attributes

The content styling attributes allow basic text formatting. All of the support for styling is defined to be on a best-effort basis: unsupported attributes may be ignored by a given implementation. The attributes are restricted to the following: `textColor`, `textBackgroundColor`, `textFontFamily`, `textFontSize`, `textFontStyle`, `textFontWeight`.

All of these attributes may be specified as defaults on the `<smilText>` element or on the `<region>` definition. They may be modified on a `<div>`, `<p>` or `` within a `<smilText>` content block.

Attribute: `textColor`

This attribute specifies the color used to render text content within the layout area. The permitted value is a CSS2 color specification. The initial value is undefined and will be implementation dependent.

Attribute: `textBackgroundColor`

This attribute specifies the background color used to fill the area around text content within the content's glyph area. It *does not* specify the background color of the region. The permitted value is a CSS2 color specification. The initial value is transparent.

Attribute: `textFontFamily`

This attribute defines the name or family of font used to render text. Permitted values are a list of font names, potentially augmented by the standard types: {monospace | sansSerif | serif | inherit}. The first font name to be resolved is chosen. The resolution of a generic family name to a specific font instance is not defined by *smilText*, but may be defined by a specific implementation. If all font families are unrecognized, then monospace is used. The sansSerif and serif fonts are expected to be proportional. The initial value is sansSerif.

Attribute: `textFontSize`

This attribute defines the size of the font to be used. In order to reduce implementation burden and to provide scalability across device classes, only absolute and relative sizes (as defined in XSL 1.1) are supported. Permitted values are: {absolute-size | relative-size | inherit}. The absolute sizes supported are: {xx-small | x-small | small | medium | large | x-large | xx-large}. The relative sizes supported are: {larger | smaller}.

As recommended by XSL, the step-factor for absolute sizes is 1.2.

Attribute: `textFontStyle`

This attribute defines the style of the text displayed in the (portion of the) layout area. Of the styles allowed by XSL, only the following subset is: {normal |

TXTXT
TXTXT
TXTXT
TXTXT

`italic | oblique | reverseOblique | inherit`}. If a defined style is not available, `normal` will be used.

Attribute: `fontWeight`

This attribute defines the weight of text displayed. The values permitted are: `{normal | bold | inherit}`.

If `bold` is not available, then `normal` will be used

Attribute: `textStyle`

This attribute is an ID value that identifies a set of text attribute definitions set using the `<textStyle>` element defined on page 183.

9.5 Text Motion Elements and Attributes

One of the important uses of text within a multimedia presentation is as a scrolling or crawling object. Typical uses for scrolling objects are credits or teleprompter-like text. Typical uses for crawling objects are news feeds or stock tickers.

SMIL provides support for motion-based text by extending the `textMode` attribute with three new alternatives and by defining the `textConceal` and the `textRate` attributes.

9.5.1 smilText Motion Elements

SMIL 3.0 does not define any new elements to support text motion. It does allow the motion attributes defined in the next section to be used on the following elements: `<region>`, `<smilText>` and `<textStyle>`.

9.5.2 smilText Motion Attributes

SMIL defines three motion control attributes: `textMode`, `textConceal` and `textRate`. Various control attributes can be combined to achieve a wide range of text rendering effects. For authoring convenience, various combinations may be predefined by a particular profile or by using the `<textStyling>` and `<textStyle>` elements. (For example, by combining rendering attributes, the *window types* defined by RealText can be duplicated.) In all cases, the various rendering attributes will be considered as hints to the implementation platform. If a particular platform cannot support the desired type for performance or other reasons, the attributes may be ignored.

Attribute: `textMode`

This attribute describes how new fragments are added to content within the same `<smilText>` element. The text motion module extends the definition given earlier on page 188 with three new values. The complete value list is:

`{append | replace | crawl | scroll | jump | inherit}`



T X T X T
T X T X T
T X T X T
T X T X T



The `append` and `replace` values were discussed on page 188; these values retain their original meaning. The `crawl`, `scroll` and `jump` values define motion-based modes. The `crawl` attribute causes text to move as a single line against the primary writing direction defined by `textWritingMode` attribute. The starting position of the text will be determined by the `textAlign` attribute, possibly modified by the `textConceal` attribute defined next. The `scroll` attribute causes text to move against the secondary writing direction within the rendering region. The initial position is defined by the `textPlace` attribute. The `jump` mode allows text to enter the display area one logical line at a time and then be pushed up a line when each preceding line is full. The initial position will be determined by the `textPlace` attribute.

In all cases of motion-based modes, the `textRate` attribute (defined below) determines the speed of motion. If an implementation cannot support (or determine) the motion speed, an implementation-dependent value is used.

If `<tev>` or `<clear>` elements are placed within the motion text stream, an implementation will support incremental text display on a best-effort basis.

Attribute: textConceal

This attribute identifies whether a moving text string starts outside or inside the rendering region, and whether it exits the region at the end of its duration. The supported values are: { `none` | `initial` | `final` | `both` | `inherit` }.

When the `none` value is used, text will be rendered at the starting point in the region defined by the `textPlace` and `textAlign` attributes, and will finish at the terminating placement determined by the normal processing of the content. No leading or trailing space will be added.

The `initial` value will result in the text being initially rendered just outside the visible area of the region, and then entering the region along its motion path. It will finish at the terminating placement determined by the normal processing of the content. No trailing space will be added.

The `final` value will result in the text being initially rendered at the starting point in the region defined by the `textPlace` and `textAlign` attributes, and fully exiting the region at the end of its duration. Only trailing space is added.

The `both` value will cause leading and trailing space to be added, so that the text starts and ends outside the region.

The initial value is `none`, which is inherited. (See Figure 9-1.)

Attribute: textRate

This attribute describes the rate at which text motion occurs. The permitted values are: { `auto` | `CSS2 pixel value` | `non-negative integer` }. The default rate is `auto`.

Value: auto

For the `auto` value, the text movement rate will depend on the value of the `textMode` attribute. When used with the `crawl` or `scroll` text modes, `auto`



T X T X T
T X T X T
T X T X T
T X T X T

specifies that the *smilText* user agent should determine the appropriate rate of movement (in pixels per second) to display the entire content of the `<smilText>` element (including any extra spacing required for processing the `textConceal` attribute) within its simple duration. All timing markup on `<tev>/<clear>` elements is ignored and no activation events will be generated from within the `<smilText>` element. If the simple duration is not known, implementations will make an informed guess or provide a default rate. When used with the jump text mode, `auto` specifies that a single line in the secondary writing direction is jumped when an explicit or implicit line break is generated. All timing markup on `<tev>/<clear>` elements is respected. When used with all other text modes, this attribute is ignored.

Value: CSS2 pixel value

The interpretation of the pixel rate value depends on the value of `textMode` attribute. For `crawl` and `scroll`, the pixel value defines a non-negative integer representing a pixel movement rate in pixels per second. (The “px” unit qualifier in pixel values may be omitted.) When used with the jump text mode, this value will be interpreted as a non-negative integer defining the number of lines to jump. Again, the “px” qualifier may be omitted. When used with all other text modes, this attribute is ignored.

Value: non-negative integer

A non-negative integer is always interpreted as a CSS2 pixel value without the “px” qualifier.

9.6 Examples of smilText Use

This section contains a number of examples of *smilText* use. Please recall that *smilText* functionality is only available on a SMIL 3.0 or later user agent. The text functionality can be expected in players that conform to the LANGUAGE or UNIFIEDMOBILE profiles.



T X T X T
T X T X T
T X T X T
T X T X T

9.6.1 Basic smilText

The most basic use of *smilText* is as a wrapper for a simple text string. All of the timing is placed on the `<smilText>` element, and no styling is used within the object:

```
...  
<smilText dur="5s" region="label">  
  The story of my dog Gretchen.  
</smilText>  
...
```

This example is not particularly exciting, but it does illustrate text functionality that SMIL has been missing for 10 years.



One simple extension of our basic example is to add styling information on the `<smilText>` element. This could include a particular font, a text weight or a text color. An example is:

```
<smilText dur="5s" region="label" textColor="red" textFontWeight="bold"
  textFontStyle="italic" textFontFamily="palatino, garamond, serif">
  The story of my dog Gretchen.
</smilText>
```

Note that while this example looks pretty straightforward, certain implementations of *smilText* may not be able to provide styling support. In all cases, styling is performed on a best effort basis.

Most *smilText*-capable user agents that support styling will probably also support content segmentation. In this case, familiar forms of local style application can be expected. For example, consider the following fragment:

```
<smilText dur="5s" region="label" textColor="red" textFontWeight="bold"
  textFontStyle="italic" textFontFamily="palatino, garamond, serif">
  The story of my dog <span fontColor="green">Gretchen</span>.
</smilText>
```

Here, the name of the dog is colored green, while the rest of the text is red.

In most in-line uses of *smilText*, the content within the `<smilText>` element will be rendered in a SMIL region. The region definition determines the rendering space available for the text content. If the text in the object is too wide for the region (assuming left-to-right rendering), then the `textWrapOpti on` will determine whether the content is clipped or wrapped to a following line.

```
<smil ... >
<head>
...
<layout>
...
<region xml:id="label" top="5px" left="10%" width="80%" height="30px"/>
...
</layout>
</head>
<body>
...
<smilText dur="5s" region="label">
  The story of my dog Gretchen, who was not only a beautiful animal
  but also one of my best friends.
</smilText>
...
</body>
</smil >
```

The default behavior is that long text is wrapped to the next line. If there is no vertical space available, the text is placed outside of the visible area of the region.

The `<region>` definition may also be used to hold a set of default styling values. These apply to all text content rendered in that region, unless overridden on the `<smilText>` element (or on interior segmentation elements).

```
<region xml:id="label" top="5px" left="10%" width="80%" height="30px"
  textColor="red" textFontWeight="bold" textFontStyle="italic"
  textFontFamily="palatino, garamond, serif" />
```

9.6.2 Adding Explicit Timing Within Text

While the basic text control outlined in the previous set of example is useful, the insertion of intra-block timing provides one of the compelling uses of *smilText*. Intra-block timing is accomplished using the `<tev>` or `<clear>` elements, together with the `begin` or `next` attributes.

In the following fragment, a text string is presented in temporal chunks

```
...
<smilText dur="5s" region="label">
  The story of my dog Gretchen,
  <tev begin="1.5s"/>
  who was not only a beautiful animal
  <tev begin="3s"/>
  but also one of my best friends.
  <clear begin="4.5s"/>
</smilText>
...
```

In this example, each text fragment will be added incrementally to the region because of the effective value of the `textMode` attribute. If `textMode` had been set to `replace`, then each fragment's entry would first clear the region.

Let's suppose that we wanted to change the time at which the second fragment ("who was ...") appeared from 3 seconds after the start of the `<smilText>` object to 3.5 seconds. One impact may be that down-stream timing may also need to be changed. In order to make authoring easier, *smilText* allows the scheduling information for fragments to be specified as a relative value. This is done via the `next` attribute:

```
...
<smilText dur="5s" region="label">
  The story of my dog Gretchen,
  <tev next="1.5s"/>
  who was not only a beautiful animal
  <tev next="1.5s"/>
  but also one of my best friends.
  <clear begin="4.5s"/>
</smilText>
...
```

Each `next` attribute defines an interval that is relative to the previous `<tev>` or `<clear>` element in the object. (The first `next` is relative to the start of the `<smilText>` element.) *SmilText* allows `begin` and `next` timing to be mixed, but the actual schedule produced by a *smilText* object must always be strictly linear. If an absolute time is specified to a moment that has already occurred, the resolved time is set to the current instant. All subsequent relative times are based on the actual starting times of the time base.

In the examples up to this point, the `<smilText>` element has had a specific duration defined. This is not necessary, since the `<smilText>` object acts like a continuous media element. The total time of all of the temporal moments defined in the text object are used as the inherent duration of the `<smilText>` element.



T X T X T
T X T X T
T X T X T
T X T X T

9.6.3 Adding Event-Based Text Timing

Internal timing within a *smilText* object is useful for controlling the appearance of successive text strings. This is a powerful feature for many multimedia applications. Even more powerful is the ability of the appearance of one of the text fragments to act as a trigger for some other object in the presentation (such as a bit of audio). Similarly, it is sometimes useful to have an external object trigger the next text fragment within a block. *SmilText* supports both of these operations.

In the following fragment, we show how you can trigger a dog's bark (captured in the `woof.mp3` audio file) at the point that the third text fragment is about to be rendered:

```
...
<par>
  <smilText dur="5s" region="Label">
    The story of my dog Gretchen,
    <tev next="1.5s"/>
    who was not only a beautiful animal
    <tev xml:id="barkNow" next="1.5s"/>
    but also one of my best friends.
    <clear begin="4.5s"/>
  </smilText>
  <audio begin="barkNow.beginEvent" src="woof.mp3" region="audio" />
</par>
...
```

In this example, the start of the `<tev>` with ID `barkNow` resolves the begin time on the audio object. The bark and the last text line start together.

Since we don't know how long the bark audio lasts, it may be interesting to have the completion of the audio trigger the last text fragment in our example. This is accomplished in the following code example:

```
...
<par>
  <smilText region="Label">
    The story of my dog Gretchen,
    <tev next="1.5s"/>
    who was not only a beautiful animal
    <tev xml:id="barkNow" next="1.5s"/>
    <tev begin="woof.endEvent" />
    but also one of my best friends.
    <clear xml:id="sighNow" begin="4.5s"/>
  </smilText>
  <audio xml:id="woof" begin="barkNow.beginEvent" src="woof.mp3" ... />
  <audio begin="sighNow.beginEvent" src="sigh.mp3" ... />
</par>
...
```

As before, the `woof.mp3` audio is triggered by the second `<tev>`, but now a third `<tev>` waits for the audio to end before continuing. Once the last fragment has been displayed, a final audio object is triggered with a sympathetic sigh.

The use of event-based timing is only allowed when the `<smilText>` element if the companion objects are encoded in the same SMIL file. On `<tev>` and `<clear>`, events may only be used with the `begin` attribute.

9.6.4 Adding Text Motion

Text motion in *smilText* is accomplished by selecting a motion-based value in the **textMode** attribute. These are: *crawl*, *scroll* and *jump*. All three values make use of the concepts of *primary text direction* and *secondary text direction*. The primary text direction is the direction of the normal flow of successive text characters in a sentence, while the secondary text direction is the direction used to add new primary-direction lines. In this book, the primary text direction is left-to-right, and the secondary direction is top-to-bottom.

When text is crawled or scrolled in *smilText*, it is always in a direction that is opposite to the primary/secondary writing direction. Figure 9-1 tries to capture this on a static page for the following crawling text example:

```
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0" baseProfile="Language">
  <head>
    <layout>
      <root-layout width="500" height="60"/>
      <region xml:id="crawlText" left="0" top="0" width="100%" height="100%"
        textMode="crawl" />
    </layout>
  </head>
  <body>
    <body>
      <smilText region="crawlText" dur="11s">
        The story of my dog Gretchen,
        who was not only a beautiful animal,
        but was also one of my best friends.
      </smilText>
    </body>
  </smil>
```

At the start of the motion, all of the text positioning defaults are applied, so that the text string is positioned at the top-left side of the region. The text duration is defined as 11 seconds, during which all motion takes place. The motion ends when the last characters have entered from the right — these last characters do not progress across the screen.

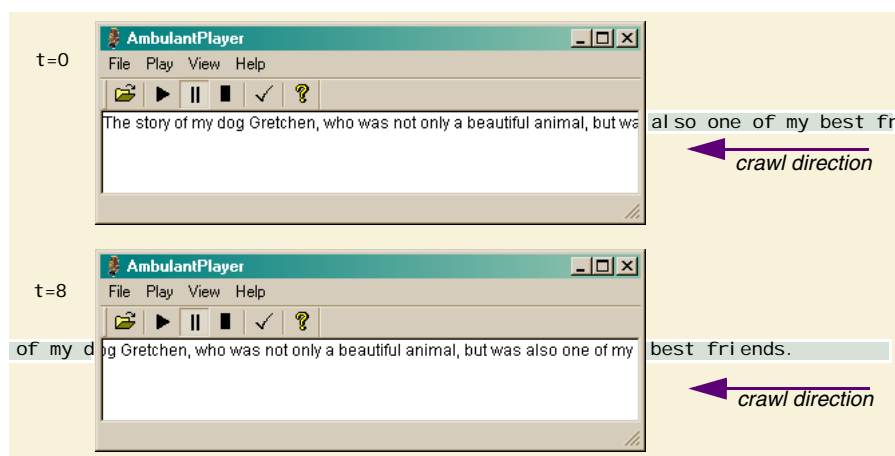


Figure 9-1. Positioning crawling text.

The initial placement of text at the left edge and the final placement at the right edge can be changed using the **textConceal** attribute. By setting the value to *initial*, the text is initially placed outside the region (logically, to the right) and then crawled in. By setting the value to *final*, the text fully exits the region before the duration ends. A value of both combines these effects.

In Figure 9-1, the text is placed along the top edge of the region. This placement can be changed using the **textPlace** attribute. A value of *start* puts the text at the beginning of the secondary text position — the top of the block for top-to-bottom text. A value of *center* puts it in the middle of the region, while the *end* value puts it at the bottom. If the **textMode** were set to *scroll*, we would get similar behavior, but then along the left/right secondary direction.

The example above does not define a specific movement rate. The default value of *auto* is assumed for the **textRate** attribute. The movement of the text could be increased or decreased by assigning a higher or slower text rate, although for most practical situations, the *auto* value should be sufficient.

It is difficult to give nice visual representations of moving text in a printed book. You are encouraged to download a *smilText* player (such as *Ambulant*) and try various combinations of the motion attributes. The book's web site has a series of motion demos to get you started.

9.6.5 Reusing Styles

In order to make the application of a common set of styles more convenient, *smilText* defines a styling container that can be placed within a document **<head>** section:

```
<smil xmlns="http://www.w3.org/ns/SMIL" version="3.0" baseProfile="Language" >
<head>
  <textStyling>
    <textStyle id="headlines"
      textColor="blue" textBackgroundColor="orange"
      textAlign="center" textFontFamily="sansSerif"
      textFontSize="large" textFontWeight="bold" />
  </textStyling>
</head>
<body>
  ...
  <smilText begin="10s" dur="3s" textStyle="headlines" >
    A Geometric Tour of New York
  </smilText>
</body>
</smil >
```

In this example, the pre-defined style *headlines* could be applied to any *smilText* element or used within any ****. The aggregated text styles could also be applied to the **<region>** definition, to set default styling for all content rendered in that region.

9.7 Using smilText as an External Format

In addition to the embedded version of *smilText*, a complete **<smilText>** element could be placed in an external file and then played by a compliant user

agent. This could be a standard SMIL 3.0 player, or it could be a browser plug-in. (The book's web site will point you to plug-in implementations.)

The primary advantage of using *smilText* as an external format is that the text content can be bound to the presentation at document run-time, rather than at document author time. The text can be automatically generated based on information on the presentation user, or it can be dynamically updated from a streaming source. This points to a second advantage of using *smilText* constructs in an external file: the format lends itself nicely to content streaming.

9.8 Summary and Conclusion

One of the new areas of functionality within SMIL 3.0 is the support for in-line text content. This content can consist of simple text, or it can contain embedded timing commands or text motion attributes.

SmilText builds on a number of standard XML technologies, such as XSL and the emerging DFXP. It also retains many of the advantages of Real's *RealText* format, which has been one of the standard formats for real-world timed text.

The goal of *smilText* has been to balance styling complexity with rendering and scheduling efficiency. For this reason, you won't find tables or embedded media commands within a *smilText* block. Even the timing constructs `<tev>` and `<clear>` were deliberately selected so as not to give the impression that full SMIL timing was available within a `<smilText>` object. Time will tell if this approach gains acceptance.

9.9 Further Resources

XML

Extensible Markup Language (XML) 1.1, Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, François Yergeau and John Cowan (Eds.), W3C Recommendation, 04 February 2004.

<http://www.w3.org/TR/2004/REC-xml11-20040204/>.

XSL

Extensible Stylesheet Language (XSL) Version 1.1, Anders Berglund (Ed.), W3C Recommendation, 5 December 2006. <http://www.w3.org/TR/xsl1/>.

DFXP

Timed Text Distribution Format Transfer Profile (DFXP), Glenn Adams (Ed.), Information available at: <http://www.w3.org/AudioVideo/TT/>.



T X T X T
T X T X T
T X T X T
T X T X T



<http://www.springer.com/978-3-540-78546-0>

SMIL 3.0

Flexible Multimedia for Web, Mobile Devices and Daisy
Talking Books

Bulterman, D.C.A.; Rutledge, L.W.

2009, Approx. 535 p., Hardcover

ISBN: 978-3-540-78546-0