Acronym Cheat Sheet

CSS	**Cascading Style Sheets** (17-Dec-1996/Rev. 11-Jan-1999) Style sheets define how content should be rendered (font, color, spacing) in a Web document. Each tag (<body>, <p>, …) can have its own style.
	CSS2 Cascading Style Sheets, Level 2 (12-May-1998) Builds on CSS. Media-specific style sheets that can be tailored to visual browsers, text readers, printers, braille devices, etc.
	CSS3 Cascading Style Sheets, Level 3 (in progress) Modularization and extension of CSS2. Modules include: Selectors, Box Model, Color, Text, Fonts, Backgrounds, Values and Units, Cascading and Inheritance.
DHTML	**Dynamic HTML** The mixture of JavaScript, HTML, DOM and CSS for dynamic pages (content created on the fly, navigation effects, pull-down and pop-up menus,…).
DOM	**Document Object Model** (Level 1: 1-Oct-1998; 2nd Ed. 29-Sept-2000) Platform-independent roadmap to the internal hierarchy of the elements in a document. It allows scripts to access and dynamically modify document content, structure, and style.
	DOM Level 2 Builds on DOM Level 1. Modules: Core, Views, Events, Style, Traversal, and Range.
	DOM Level 3 Builds on DOM Level 2. Modules: Core, Abstract Schemas, Load and Save, Events, XPath.
DTD	**Document Type Definition** The “grammar” for your markup language: the set of elements (tags), their attributes, and legal structure, to mark up a document for a particular application. Being replaced by: XML Schema
HTML	**HyperText Markup Language** (4.01: 24-Dec-1999) The standard publishing language of the World Wide Web. A usage of SGML.
ISO	**International Organization for Standardization** A worldwide federation of national standards bodies, one from each of over 100 countries. US member is ANSI (American National Standards Institute).
MIME	**Multipurpose Internet Mail Extensions** MIME (as in MIME content-type) defines a format and framework for representing a wide variety of data types in Internet applications.

RDF	**Resource Description Framework** XML-based syntax for metadata (information about information).	(22-Feb-1999)
SGML	**Standard Generalized Markup Language** ISO 8879 standard markup language for authors to describe the structure of their documents. HTML and XML are usages of SGML.	(1986)
SMIL	**Synchronized Multimedia Integration Language** Markup language for multimedia presentations.	(2.0, 5-June-2001)
SVG	**Scalable Vector Graphics** Markup language for 2D vector graphics presentations.	(1.0, 4-Sept-2001)
XHTML	**Extensible HTML** XHTML 1.0 is a reformulation of HTML 4.01 in XML.	(1.0, 26-Jan-2000)
XML	**Extensible Markup Language** Simplified version of SGML. Markup for the structure of a document. Author determines the tags necessary for the application.	(1.0, 2nd Ed. 6-May-2000)
XML Schema	**Extensible Markup Language Schema** Similar to the DTD, except written in XML. Schema also provides for specifying the datatype associated with an attribute.	(2-May-2001)
XPath	**XML Path** Language for how to reference a specific part of an XML document.	
XSL	**Extensible Stylesheet Language** To view an XML document, it needs to be formatted and styled for a particular browser or player. Styling instructions are organized in style sheets such as CSS and XSL. . XSL has more advanced styling than CSS: it can transform a document before displaying it. XSL consists of XSLT, XPath, and XSL FO.	(2.0: 15-Oct-2001)
XSL FO	**XSL Formatting Objects** Language for formatting, including page formatting, font size, font style, and font weight.	
XSLT	**XSL Transformations** Language for transforming an XML document into another.	

XML Basics for XHTML, SVG, and SMIL

§1. Background

1.1 Purpose of this Course

Web-based technologies such as HTML and JavaScript are often learned by cutting and pasting code from other documents. Eager to get to the "fun stuff", we sometimes don't take the time to understand how things really work. This XML Basics course has that information.

Two XML-based languages that are growing in popularity are the Synchronized Multimedia Integration Language (SMIL, pronounced "smile") and Scalable Vector Graphics (SVG). Both are standards maintained by the World Wide Web Consortium (W3C). Starting with a review of HTML, we'll see why the need for XML arose. We will cover the basics of XML, including well-formed documents and the DTD (document type definition). We will then look at simple examples of SMIL and SVG documents, both standalone and embedded in web pages.

1.2 History of Web Presentation

HTML, the HyperText Markup Language, is the current language for web page authoring. The concept of *markup* originates from text publishing. The publisher would annotate a manuscript with typesetting instructions such as layout, typeface and boldness. As electronic publishing came into being, HTML was created for formatting and hyperlinking electronic text documents.

The term *hypertext* was coined in 1965 by Ted Nelson (*A File Structure for the Complex, the Changing, and the Indeterminate*. 20th National Conference, New York, Association for Computing Machinery, 1965).

Hyper **Text** Markup Language

- Designed for text and text hyperlinking
- Consists of pre-defined markup tags

HTML -- Pros

- Easy
- Non-proprietary format
- Widespread acceptance

HTML's ease of use, non-proprietary format, and widespread acceptance has caused it to be used for multimedia purposes for which is was not designed. The markup tags are pre-defined, and may not suit your needs, but you try anyway. One common trick is to lay out your images and text by clever use of tables. Another fundamental problem is that web page content is intermixed with its presentation. Style sheets were added to help separate content from presentation.

Interactivity requires scripting (JavaScript, CGI), animated gifs, or programming (Java applets, Flash, Director).

HTML -- Cons

- Content is static, not dynamic
- Can't add new tags
- Content and presentation is intermixed

XML, Extensible Markup Language, was designed as a standard to take care of these limitations. An author defines the tags needed for the application at hand, and a model that describes every element that can appear in the documents. The focus is on the structure of the document, and the meaning of the content. A separate model, the XSL, specifies how the content shall be rendered. The same content can be presented in different ways as needed.

XML

- Author defines tags
- Focus on structure and meaning of content
- Separate model specified rendering

Summary of Markup History

1. SGML
   - ➢ Too complicated
2. Simplified SGML for text: HTML
   - ➢ Too restrictive
3. Simplified SGML for general data markup: XML
   - ✓ SVG
   - ✓ SMIL

1.3 W3C: World Wide Web Consortium **www.w3.org**

W3C are the keepers of the standards, founded in 1994 by Tim Berners-Lee, the inventor of the web as we know it today. Through working groups of industry experts, their mission is the development and maintenance of web standards.

Here are just a few of the relevant sites:

www.w3.org/MarkUp	HTML
www.w3.org/AudioVideo	SMIL
www.w3.org/XML	XML
www.w3.org/Style/CSS	Style sheets
www.w3.org/Style/XSL	XML Style sheets
www.w3.org/DOM/DOMTR	Document Object Model
www.w3.org/Graphics	Web Graphics
www.w3.org/Graphics/SVG	SVG

§2. HTML

2.1. Basic Structure of HTML Document

```
<!-- Basic web page -->
<html>
     <head>
          <title> Basic Web Page </title>
     </head>

     <body>
     </body>
</html>
```

<html> is the top-level element (the "root" element) that contains the entire document. It has two subsections: the **<head>** and the **<body>**.

<head> contains information about the document.

<title> contains the title of the document, shown in the titlebar of the browser window

<body> contains the content of the document.

Comments are delimited by **<!--** and **-->**

2.2. Work Flow

A. Editor

- Used to create the document.
- Pages can be created in a wide range of tools
  - Write raw HTML in a text editor such as Notepad or vi
  - WYSIWYG authoring tools that hide the HTML from you.

B. Browser

- Interprets the tags and renders the document.
- Presents you a view of the document.
- Current browsers are very lenient. They accept sloppy HTML, and consequently are large, slow, and difficult to update

2.3. Markup Terms

A. Element

- *name* and *content*
- Delimited by *start* and *end* tags
- An element can have 0, 1, or many attributes

Example:

```
<title> Intro to SMIL </title>
```

The name of the element is **`title`**.
The content is the text "**`Intro to SMIL`**".

B. Attribute

- A parameter to an element.
- An element can have 0, 1, or many attributes
- Specified in the start tag as *`name`* = "*`value`*"
- Enclose value in quotes

Example:

```
<table border = "0" >
```

The element name is **`table`**.
The name of the **`table`** attribute is **`border`**.
The value of **`border`** is `0`. (no border drawn around the table)

C. Empty Element

- All the content information is specified in the attributes.
- Instead of an end tag, close with a forward slash:

Example:

```
<img src = "hockey.jpg"  />
```

The element name is **`img`**.	image
The name of the **`img`** attribute is **`src`**.	source of image
The value of **`src`** is **`hockey.jpg`**.	filename of image

D. Container Tag

- The content is contained between a start tag and an end tag.

Examples:

```
<title> Mega-Widget Price List </title>
<h1> Section 1 </h1>
```

E. Parsing

Parsing is the act of scanning a document and interpreting the information based on the structure of the elements.

F. Rendering

Rendering is the act of presenting a view of the information in a document. The presentation is in the form most appropriate to the environment. Browsers usually present a visual page, but they can also present a spoken or printed version. Use the `alt` attribute to specify an alternative rendering for purely visual items such as images.

G. Structure vs Presentation

There are two kinds of tags in HTML: *structure* tags and *presentation* tags. Structure tags define anatomical information about the document; Presentation tags define the appearance of the document.

1. Structure tags

```
<head>    ...    </head>
<title>   ...    </title>
<body>    ...    </body>
```

2. Presentation tags

`<center>`	…	`</center>`	display centered
`<font>`	…	`</font>`	set the font
`<b>`	…	`</b>`	display in boldface

The **`font`** tag and many presentation-related attributes of other tags were deprecated in HTML 4.0 in favor of style sheets.

2.4. Style Sheets

- Define appearance.
- Separate structure of document from rendering instructions
- Useful for maintaining consistent look across large websites.

A. Specifing rules

- Rules have the following format

  ```
  selector { property:value }
  ```

- Cascading Style Sheets (CSS) defines the properties of the elements and their allowed values

- 3 Types of selectors:
  1. Set style for all occurrences of a given tag

     ```
     element { property:value }
     ```

     Example,

     ```
     h1 { font-weight:bold }
     ```

     Result: Everywhere in the document where an `h1` container tag is used, such as follows, the text will be rendered in boldface.

     ```
     <h1> Example 1 </h1>
     ```

  2. Set style for tag whose `class` attribute has a certain value

     ```
     element.classValue { property:value }
     ```

     Example,

     ```
     h3.r { color:red }
     ```

     Result: Everywhere in the document where an `h3` container tag is used with a `class` attribute whose value is "`r`", such as follows, the text will be rendered in red.

     ```
     <h3 class="r"> Example 2 </h3>
     ```

  3. Set style for tag whose `id` attribute has a certain value

     ```
     element#idValue { property:value }
     ```

     Example,

     ```
     p#note { font-style:italic }
     ```

     Result: Everywhere in the document where a `p` container tag is used with an `id` attribute whose value is "`note`", such as follows, the text will be rendered in red.

     ```
     <p id="note"> Example 3 </p>
     ```

B. Associating a style with a document

Style is specified in the `<head>` of the HTML document.

1. Complete definition contained in the header

```
<html>
    <head>
        <title> CSS Example 2.1 </title>
        <style type="text/css">
            h1     { font-weight:bold  }
            h2     { color:blue }
            h3.r   { color:red }
            p#note { font-style:italic }
        </style>
    </head>

    <body>
        <h1> Using CSS </h1>
        <h2> How to set the style </h2>
        <h3 class="r"> Set style for tag with 'class' attr</h3>
        <h3> Set style for all occurrences of a given tag </h3>
        <p id="note"> You can also use the 'id' attribute </p>
    </body>
</html>
```

2. Complete definition is contained in a separate file, and linked in the header. Useful for modifying the appearance of an entire website by changing just the stylesheet.

```
<html>
    <head>
        <title> CSS Example 2.2 </title>
        <link rel="stylesheet" type="text/css"
              href="ex2.2.css" />
    </head>
             …
</html>
```

   - File `ex2.2.css` contains:

     ```
     h1     { font-weight:bold  }
     h2     { color:blue }
     h3.r   { color:red }
     p#note { font-style:italic }
     ```

   - Attribute **`rel`** specifies the relationship that the **`href`** attribute has to the document. Here we specify that **`href`** links to a style sheet. (For other possible relationships see `http://www.w3.org/TR/html4/types.html#type-links`)

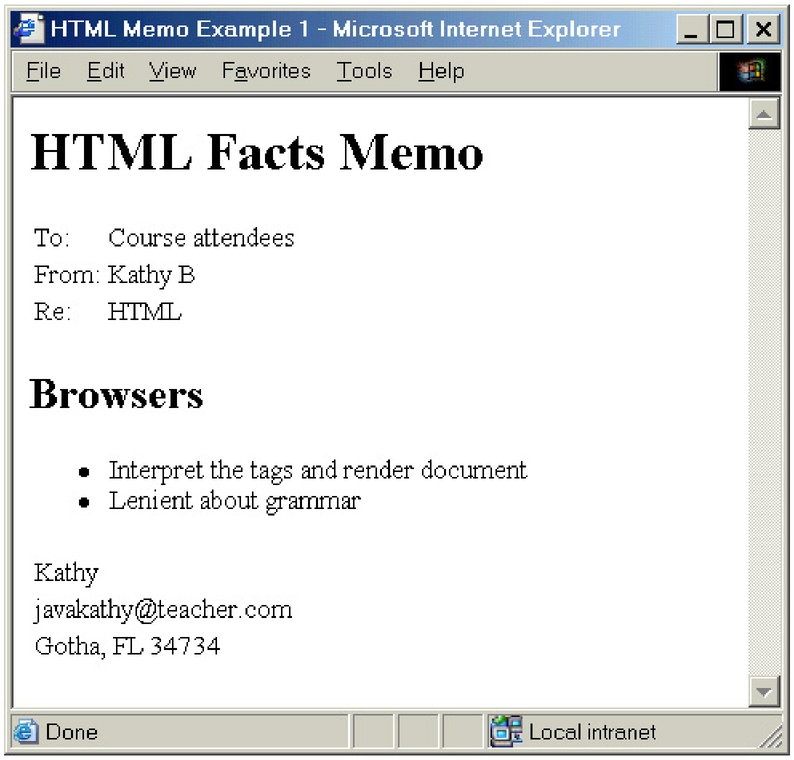
2.5. Examples

A. HTML_Ex/memo_ex1.html

```
<html>
    <head>
        <title> HTML Memo Example 1 </title>
    </head>

    <body>
        <h1> HTML Facts Memo </h1>
        <table border="0">
            <tr> <td> To:   </td> <td> Course attendees </td></tr>
            <tr> <td> From: </td> <td> Kathy B          </td></tr>
            <tr> <td> Re:   </td> <td> HTML             </td></tr>
        </table>

        <h2> Browsers </h2>
        <ul>
            <li> Interpret the tags and render document  </li>
            <li> Lenient about grammar                   </li>
        </ul>

        <table>
            <tr> <td> Kathy                  </td></tr>
            <tr> <td> javakathy@teacher.com </td></tr>
            <tr> <td> Gotha, FL 34734       </td></tr>
        </table>
    </body>
</html>
```

Description of tags:

`h1`	Heading level 1; Most important. Useful for title or main section.	
`h2`	Heading level 2. Useful for first subheading.	
`table`	Organize data in rows and columns.	
	`tr`	Table row.
	`td`	Table cell data
`ul`	Unordered list (items are bulleted, not numbered)	
	`li`	List item


HTML Memo Example 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Local intranet

HTML Facts Memo

To: Course attendees
From: Kathy B
Re: HTML

Browsers

- Interpret the tags and render document
- Lenient about grammar

Kathy
javakathy@teacher.com
Gotha, FL 34734

Figure 2-1: HTML Facts Memo

HTML_Ex/memo_ex2.html

```
<!-- Example 2 shows sloppy HTML... missing end tags, case changes.
Browser is able to compensate, and document renders just like
Example 1.
-->
<HTML>
    <head>
        <title> HTML Memo Example 2: Sloppy HTML </title>
    </head>

    <bOdY>
        <h1> HTML Facts Memo </h1>
        <TABLE border="0">
            <tr> <td> To:    <td> Course attendees
            <tr> <td> From:  <td> Kathy B
            <tr> <td> Re:    <td> HTML
        </table>

        <h2> Browsers </h2>
        <ul>
            <li> Interpret the tags and render document
            <li> Lenient about grammar
        </ul>

        <table>
            <tr> <td> Kathy
            <tr> <td> javakathy@teacher.com
            <tr> <td> Gotha, FL 34734
        </table>
    </body>
</html>
```

Q: Why is this important to know?

A: XML-based languages are very strict about syntax... they don't tolerate sloppiness. If you've gotten into any bad habits writing with HTML, you'll have to break them before moving on to SVG or SMIL.

B. HTML_Ex/memo_ex3.html

```
<!-- Example 3 is the same text as Example 1.
Style information is specified in the header -->

<html>
    <head>
        <title> HTML Memo Example 3 </title>
        <style type="text/css">
            table.contact { color: maroon;
                background: rgb(204,204,255); }
            h1 { font-weight: bold; font-size: 150%; color:blue }
            h2 { margin-left: 5%; margin-right: 10%;}
        </style>
    </head>

    <body>
        <h1> HTML Facts Memo </h1>
        <table border="0">
            <tr> <td> To:   </td> <td> Course attendees </td></tr>
            <tr> <td> From: </td> <td> Kathy B          </td></tr>
            <tr> <td> Re:   </td> <td> HTML             </td></tr>
        </table>

        <h2> Browsers </h2>
        <ul>
            <li> Interpret the tags and render document  </li>
            <li> Lenient about grammar                   </li>
        </ul>

        <table class="contact">
            <tr> <td> Kathy                  </td></tr>
            <tr> <td> javakathy@teacher.com </td></tr>
            <tr> <td> Gotha, FL 34734       </td></tr>
        </table>
    </body>
</html>
```

2.6. Quiz

Q1.1) Given

```
<h2> Accessories for sale </h2>
```

element `h2` is a container tag.

a) true
b) false

Q1.2) Given

```
<a href="prices.html">Prices </a>
```

element `a` is an empty element.

a) true
b) false

Q1.3) Given

```
<img src="house.jpg" height="300" width="300"
alt="house" />
```

the element name is:

a) `img`
b) `src`
c) `height`
d) `width`
e) `house`

Q1.4) Given

```
<hr align="center" />
```

the element `hr` is an empty element.

a) true
b) false

Q1.5) Given

```
<input type=”radio” name=”Q1” value=”true”>
```

the attributes are:

a) `input`
b) `type`
c) `radio`
d) `name`
e) `Q1`
f) `value`
g) `true`

2.7. HTML Summary

A. Advantages

- non-proprietary format
- easy

B. Disadvantages

- Structure, content, and presentation are intermixed. Style sheets help, but don't completely eliminate this.

- Browsers are bloated with error-checking to compensate for poorly-written HTML.

➢ No concern as to what kind of content the text is.

The memo signature in Example 2.5A contains a name, an email address, and some mailing address information. What if you wanted an automated way to extract that information into your address book? You can't do that easily if the document is in HTML... it's just text formatted in a `<table>`... it has no meaning, even though we can look at it and know it is address information.

If HTML had tags such as `<name>`, `<address>`, and `<email>`, then it would be easy to extract the information. The solution is to use XML to markup your data, then convert it to HTML for display.

§3. XML

3.1 Overview

- XML: Extensible Markup Language
  - No tags are predefined
  - Author creates the tags needed for the application
  - Focus is on *structure* and *meaning* of the content

A. Basic XML Document

- Must have exactly one top-level ("root") element that contains the rest

Example 1:

```
<smil>
   <!-- A smil document's root element is "smil"-->
</smil>
```

Example 2:

```
<svg>
   <!-- An svg document's root element is "svg"-->
</svg>
```

Example 3:

```
<html>
   <!-- An xhtml document's root element is "html"-->
</html>
```

B. XML Document Rules

- One top-level element that contains the rest
- Required start and end tags
  - Empty elements *must* be closed with a forward slash ("/")
- Must put attribute values in quotes
- Case-sensitive element and attribute names

  Not legal: <BODY> yada, yada, yada </body>

Note: XML applications adhere strictly to these rules!

C. Examples

Example 1: XML for memo signature from Example 2.5A

```
<contact>
     <name> Kathy B </name>
     <email> javakathy@teacher.com </email>
     <address location=”home”>
          <city> Gotha </ city >
          <state> FL </state>
          <zip> 34734 </zip>
     </address>
</contact>
```

Example 2: A client database, using `<contact>` from Example 1.

```
<clientDatabase>
     <lastUpdated>
          <mm>01</mm> <dd>05</dd> <yyyy>2002</yyyy>
     </lastUpdated>
     <contact>
     </contact>
          <name> Colin </name>
          <email> colin@company.com </email>
          <address location=”work”>
               <city> Orlando </ city >
               <state> FL </state>
               <zip> 32835 </zip>
          </address>
     <contact>
          <name> Mike </name>
          <email> mike@mikesCo.com </email>
          <address location=”work”>
               <city> Tampa </ city >
               <state> FL </state>
               <zip> 34720 </zip>
          </address>
     </contact>
</clientDatabase>
```

Example 3: An inventory database

```
<inventory>
     <dept id=”hardware”>
          <item sku=”1592” name=”PC” price=”1000” />
          <item sku=”1595” name=”hard drive” price=”350” />
     </dept>
     <dept id=”software”>
          <item sku=”281” name=”PhotoShow” price=”125” />
          <item sku=”299” name=”TaxHelp” price=”38.95” />
     </dept>
<inventory>
```

D. How does XML work?

If no tags are pre-defined, what determines a valid document?

- Answer (today): The Document Type Definition (DTD)
- Answer (coming soon): XML Schema

3.2 DTD: Document Type Definition

- The "grammar" for your markup language.
  - Lists every element that is allowed in a document
  - Details each element's attributes
  - Describes the legal structure of the document (which elements are allowed where).
- You create one for your application (such as the client database)
- W3C provides DTDs for HTML, XHTML, SMIL, SVG, and others

A. Specifying an Element

`<!ELEMENT` ***elementName*** **(*contentModel*) >**

The `<!ELEMENT` keyword begins an element declaration and the > character ends it. Between these are specified the element name and the content model.

- The *elementName* is the name used in the tag.
- The *contentModel* lists the types of child elements that are acceptable in that element, and the order in which they can appear. Children can be…
  - other elements

    ```
    <!ELEMENT html (head?, body?)>
    ```

  - `#PCDATA`, which stands for "Parsed Character Data"

    Content is a character string which can contain embedded tags.

    ```
    <!ELEMENT title (#PCDATA)>
    ```

  - `EMPTY`, the keyword for *empty elements* (Elements that have their content specified only via attributes)

    ```
    <!ELEMENT img EMPTY>
    ```

- Occurrence indicators

`+`	element must appear one or several times
`*`	element can appear 0 or more times
`?`	element can appear once or not at all

- connectors

`,`	both elements to right and left of comma must appear, and in that order
`|`	either left element or right element can appear (not both)

<u>Element DTD: Example from SMIL</u>

```
<!ELEMENT smil (head?,body?)>
```

Means: a **`smil`** document will have zero or one **`head`** element, followed by zero or one **`body`** element.

legal documents

`<smil>`
`</smil>`

`<smil>`
`    <head>`
`   </head>`
`</smil>`

`<smil>`
`    <body>`
`    </body>`
`</smil>`

illegal	why illegal?

`<smil>`
`    <body> </body>`
`    <head> </head>`
`</smil>`
	`head` element must appear before the `body` element

`<smil>`
`    <head> </head>`
`    <head> </head>`
`</smil>`
	document can contain at most one `head` element

`<smil>`
`    <table>`
`    </table>`
`</smil>`
	`smil` element can only contain `head` or `body` element, not `table`

Element DTD: Advanced Example from HTML:

```
<!ELEMENT table
      (caption?, (col*|colgroup*), thead?, tfoot?,
      (tbody+|tr+))>

<!ELEMENT caption  %Inline;>
<!ELEMENT thead    (tr)+>
<!ELEMENT tfoot    (tr)+>
<!ELEMENT tbody    (tr)+>
<!ELEMENT colgroup (col)*>
<!ELEMENT col      EMPTY>
<!ELEMENT tr       (th|td)+>
<!ELEMENT th       %Flow;>
<!ELEMENT td       %Flow;>
```

legal

`<table>`
`    <caption> </caption>`
`    <col />`
`    <tfoot>`
`        <tr>`
`            <td> </td>`
`        </tr>`
`    </tfoot>`
`</table>`

Reminder: current browsers are very lenient and render illegal HTML by trying to figure out what you meant. So even though these examples are illegal syntax, chances are good your browser won't complain.

illegal	why illegal?

`<table>`
`</table>`
	table must contain one or more `tbody` or one or more `tr`.

`<table>`
`    <tr>`
`        <td> </td>`
`    </tr>`
`    <caption> </caption>`
`</table>`
	`caption` must come first

B. Specifying an Attribute

`<!ATTLIST` ***elementName attributeName attributeType defaultValue*** **`>`**

The `<!ATTLIST` keyword begins an attribute declaration and the > character ends it. Between these are specified the element name, attribute name, attribute type, and an optional default value.

- Attributes are parameters to an element.
- *attributeType*
    - `CDATA`
    - `ID`
    - `ENTITY`
    - `NMTOKEN`
    - enumerated list of tokens, separated by a `|`
- *defaultValue*
    - `#REQUIRED`
    - `#IMPLIED`
    - `#FIXED`
    - literal value

`CDATA`	Character Data. String attribute that cannot contain markup, but can contain character references such as `&lt`
`ID`	"identifier". An identifier is a unique name in the document.
`ENTITY`	A named reference. May be internal or external to the document.
`NMTOKEN`	A Name Token. Can contain letters, digits, and the punctuation dot, dash, underscore, and colon, but no white space.

`#REQUIRED`	Markup in the document must include a value for this attribute.
`#IMPLIED`	Application will provide a default value for this attribute if the markup does not provide one.
`#FIXED`	Application must use the value provided by the markup.
literal value	This value will be used if the markup does not provide a value for the attribute.

Attribute DTD: Example from XHTML

```
<!ATTLIST img
  %attrs;
  src         %URI;           #REQUIRED
  alt         %Text;          #REQUIRED
  name        NMTOKEN         #IMPLIED
  longdesc    %URI;           #IMPLIED
  height      %Length;        #IMPLIED
  width       %Length;        #IMPLIED
  usemap      %URI;           #IMPLIED
  ismap       (ismap)         #IMPLIED
  align       %ImgAlign;      #IMPLIED
  border      %Length;        #IMPLIED
  hspace      %Pixels;        #IMPLIED
  vspace      %Pixels;        #IMPLIED
>
```

[Note: `%attrs;` is an *entity*, which is described in the next section.]

legal
`<img src="face.jpg" alt="Clock face" />`

illegal	why illegal?
`<img src="face.jpg" alt="Clock face">`	Didn't close the empty element with "/>"
`<img src="hi.jpg" width="400" />`	`alt` element is required
`<img src="bar.jpg" alt="Nav bar" ismap="true" />`	The only value allowed for the attribute `ismap` is `ismap`

C. Specifying an Entity

`<!ENTITY %` *identifier value*>

The `<!ENTITY` keyword begins an entity declaration and the > character ends it. Between these are specified an identifier and a value.

- An entity is a definition.
- Anywhere that the identifier appears in the DTD, the identifier's value is substituted.
- Entities can be used to give a value a name that implies its intended use.
- Entities are useful for grouping commonly appearing attributes or attribute values under a single name.

Entity Examples from XHTML DTD:

In the attribute definition for `img` in the previous section, we saw the entities `%Length` and `%Text`, among others: They are defined as follows:

```
<!ENTITY % Text     "CDATA">

<!ENTITY % Length   "CDATA">
    <!-- nn for pixels or nn% for percentage length -->
```

We could have just specified `"CDATA"` as the value of attribute `height`. Instead, we use an entity to tell the user what type of value (a Length) we want `height` to have. [This is the best we can do with a DTD, where all we have to work with are character strings. Schemas will allow us to specify a data type, such as `nonNegativeInteger`.]

Entity Example from SVG

```
<!ENTITY % Boolean "(false | true)">
```

Most elements have an attribute **`externalResourcesRequired`** whose value is then specified as `%Boolean;`.

```
<!ELEMENT animateColor (%descTitleMetadata;%animateColorExt;) >
<!ATTLIST animateColor
  externalResourcesRequired %Boolean; #IMPLIED
      ...
>
```

Complicated Entity Example from SVG

```
<!-- This entity allows for at most one of desc, title and
metadata, supplied in any order -->

<!ENTITY % descTitleMetadata "(
     (
        ( desc,((title,metadata?)|(metadata,title?) )?) |
        ( title,((desc,metadata?)|(metadata,desc?)  )?) |
        ( metadata,((desc,title?)|(title,desc?)     )?)
     )? )" >
```

This example shows why comments are so important!

Element/Attribute/Entity Example from SMIL, "Metadata"

```
<!ENTITY % skip-attr "skip-content (true|false) 'true'">

<!ELEMENT meta EMPTY>
<!ATTLIST meta
        name    NMTOKEN #REQUIRED
        content CDATA   #REQUIRED
        %skip-attr;
>
```

legal
`<meta name="Author" content="Kathy B" />`
`<meta name="Author" content="Kathy B" skip-content="false" />`

illegal	why illegal?
`<meta name="Author" content="Kathy B" skip-content="yes" />`	`skip-content`'s allowed values are `true` or `false`
`<meta author="John" />`	`author` is not a valid attribute for `meta`

Element/Entity Example from SVG, "Filter Effects"

```
<!ELEMENT filter
          ( %descTitleMetadata;,
            (feBlend|feFlood|feColorMatrix|
             feComponentTransfer|feComposite|feConvolveMatrix|
             feDiffuseLighting|feDisplacementMap|
             feGaussianBlur|feImage|feMerge|feMorphology|
             feOffset|feSpecularLighting|feTile|feTurbulence|
             animate|set
             %filterExt;)*) >
```

legal

`<filter …  >`
    `<desc> </desc>`
    `<feOffset … />`
    `<feGaussianBlur … />`
    `<feMerge> … </feMerge>`
`</filter>`

illegal	why illegal?

`<filter …  >`
    `<filter … >`
    `</filter>`
`</filter>`
	`filter` is not a legal child element of `filter`

`<filter …  >`
    `<feOffset … />`
    `<title> </title>`
    `<feGaussianBlur … />`
    `<feMerge> … </feMerge>`
`</filter>`
	the entity `%descTitleMetadata` must be the *first* child element, not the second as shown here

D. Example: DTD for Memo Signature

```
<!ELEMENT contact (name,email*,address+)>

<!ELEMENT name      (#PCDATA) >
<!ELEMENT email     (#PCDATA) >
<!ATTLIST email     location (home|office|other) "office">

<!ELEMENT address   (street?,city,state,zip) >
<!ATTLIST address   location (home|office|other) "office">
<!ELEMENT street    (#PCDATA) >
<!ELEMENT city      (#PCDATA) >
<!ELEMENT state     (#PCDATA) >
<!ELEMENT zip       (#PCDATA) >
```

We've defined a `contact` that is composed of a `name`, any number of `email` addresses (optional), and at least one `address`. The `name` is a character string. The `email` is a character string that has a `location` attribute, which can be one of "`home`", "`office`", or "`other`".

An `address` consists of an optional `street`, and required `city`, `state`, and `zip`. As for `email`, `address` has a `location` attribute with the default of "`office`", which can be one of "`home`", "`office`", or "`other`". The elements `street`, `city`, `state`, and `zip` are character strings.

Thus the following XML is legal based on the DTD

```
<contact>
     <name> Kathy B </name>
     <email> javakathy@teacher.com </email>
     <address location="home">
          <city> Gotha </ city >
          <state> FL </state>
          <zip> 34734 </zip>
     </address>
</contact>
```

3.3 Prolog and Document Type Declaration

A. Prolog

- XML documents should begin with an XML Declaration which specifies the version of XML being used.

```
<?xml version="1.0"?>
```

- XML Declaration may specify the character encoding to identify the character set used in the document.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

B. Document Type Declaration

- The XML **document type declaration** contains or points to the grammar (markup declarations) for a class of documents. This grammar is called a document type definition, or **DTD**.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- The document type declaration can point to an external subset containing markup declarations, or can contain the grammar directly in an internal subset, or both. The full DTD for a document consists of both subsets taken together

3.4 Terms

A. Attribute

An attribute is a parameter to an element. An attribute's type and value range, including a possible default value, are defined in the DTD. The specification of the type is limited to a few types such as `CDATA`, `ID`, and `NMTOKEN`.

B. Element

An element is a document structuring unit. The element's content model and attributes are defined in the DTD.

C. Valid Document

A valid document has been verified against its associated DTD. The structure, elements, and attributes are consistent with the definitions in the DTD.

- Validating that your HTML markup is well-formed:

  http://validator.w3.org/

and that your Cascading Style Sheets are valid:

  http://jigsaw.w3.org/css-validator/

D. Well-formed Document

A document is well-formed when it is structured according to the rules of the XML Recommendation. These rules require that elements are delimited by start and end tags, and are nested properly within one another.

Wrong Nesting: `<center> <h1> Intro </center> </h1>`

Right Nesting: `<center> <h1> Intro </h1> </center>`

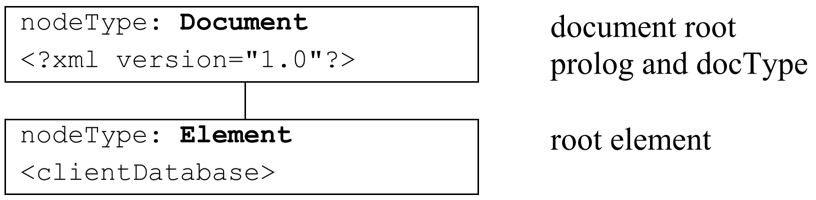
3.5 DOM: Document Object Model

- History
    - Dynamic HTML: style-sheets and scripts animate documents
    - javascript for Internet Explorer differs from Netscape because of different Document Object Models, creating web interoperability nightmares
- W3C's DOM is a standard API (Application Programming Interface) to the structure of documents
    - Defines platform- and language-independent programmatic interface to HTML and XML documents
    - Allows programmer to access and update content, structure, and style of document
    - Supports web interoperability (scripts will work on all browsers and servers that conform to the standard API)
    - Conforming implementations (Java, ECMAScript) allow programmers/script authors to access and manipulate parsed HTML and XML content
    - A parsed document: content is read in, analyzed, and placed in a **tree** structure.

- Hierarchy of DOM Tree **`Node`**s
    - The DOM root `Node` is a **`Document`** `Node`, which contains the prolog and document type declaration content.
    - **`Document`** has one child `Node`, an **`Element`** `Node` which contains the root element of the document.
    - Each **`Element`** `Node` in the tree has a **`NamedNodeMap`** of **`Attr`**s for all of its attributes.

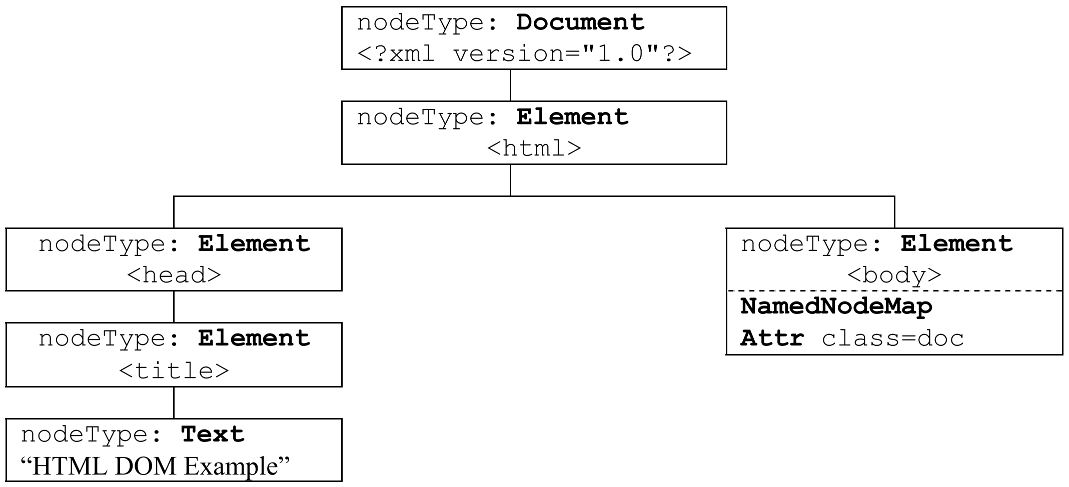
Example 1:

```
<?xml version="1.0"?>
<clientDatabase>
</clientDatabase>
```



Example 2:

```
<?xml version="1.0"?>
<html>
    <head>
        <title> HTML DOM Example</title>
    </head>
    <body class=”doc”>
        My XHTML Document
    </body>
</html>
```



3.6 Quizzes

A. XML Quiz 1

Q2.1) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil>
    <head>  </head>
    <body>  </body>
</smil>
```

a) true
b) false

Q2.2) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil id="slideshow">
    <body>
    </body>
</smil>
```

a) true
b) false

Q2.3) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil id="preview">
    <body>  </body>
    <head>  </head>
</smil>
```

a) true
b) false

Q2.4) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil id="demo">
    <head> <body> </head> </body>
</smil>
```

a) true
b) false

B. XML Quiz 2

Q3.1) Given the DTD fragment

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
          name     NMTOKEN   #REQUIRED
          content  CDATA     #REQUIRED >
```

the following markup is valid

```
<meta />
```

a) true
b) false

Q3.2) Given the DTD fragment

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
          name     NMTOKEN   #REQUIRED
          content  CDATA     #REQUIRED >
```

the following markup is valid

```
<meta name="author" content="Kathy" >
```

a) true
b) false

Q3.3) Given the DTD fragment

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
          name     NMTOKEN   #REQUIRED
          content  CDATA     #REQUIRED >
```

the following markup is valid

```
<meta content="javakathy" name="copyright"  />
```

a) true
b) false

C. XML Quiz 3

Q4.1) Given the DTD fragment

```
<!ENTITY % viewport-attrs "
          height           CDATA   #IMPLIED
          width            CDATA   #IMPLIED
          background-color CDATA   #IMPLIED
">
<!ELEMENT region EMPTY>
<!ATTLIST region
         %id-attr;
         %title-attr;
         %viewport-attrs;
         left    CDATA   "0"
         top     CDATA   "0"
         z-index CDATA   "0"
         fit (hidden|fill|meet|scroll|slice) "hidden"
         %skip-attr;     >
```

the name of the tag is

a) `viewport-attrs`
b) `region`
c) `hidden`

Q4.2) the attribute `fit` can have the value 0

a) `true`
b) `false`

Q4.3) the default value of `fit` is

a) `hidden`
b) `fill`
c) `meet`
d) `scroll`
e) `slice`

Q4.4) the following markup is valid

```
<region height="400" background-color="black" />
```

a) `true`
b) `false`

§4. XHTML: Extensible HTML

A. XHTML 1.0

- 26-Jan-2000 Reformulation of HTML 4.01 (1997) in XML
- Three DTDs are available
    - `xhtml1-strict.dtd`
        - Must adhere to strict XML rules, including separating content from layout.
        - Use with CSS
    - `xhtml1-transitional.dtd`
        - For use when migrating pages that might still be accessed by older browsers.
    - `xhtml1-frameset.dtd`
        - Use when partitioning browser window into frames
- Encouraged to use XML declaration as first line of document

  `<?xml version="1.0" encoding="UTF-8"?>`

- Place the appropriate `DOCTYPE` declaration in the document before the root element, depending on which DTD you are following:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "DTD/xhtml1-strict.dtd">


<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "DTD/xhtml1-transitional.dtd">


<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
     "DTD/xhtml1-frameset.dtd">
```

B. XHTML 1.1

- 31 May 2001: XHTML 1.1 Becomes a W3C Recommendation
- Reformulation of XHTML 1.0 using modules.
- `http://www.w3.org/TR/xhtml11/`

§5. XML Advanced Topics

A. XML Schemas

- Created because current DTD specification is limited.
    - Need to be able to easily specify a data range. For example, if you want an attribute to have a value between one and 100, you'd have to list them all:

      ```
      <!ATTLIST apt floorNum (1|2|3|4|5|……99|100)
      ```

    - No way to specify the datatype for data verification

<u>Example DTD and valid document:</u>

```
<!ELEMENT inventory (item)*>

<!ELEMENT item  (#PCDATA) >
<!ATTLIST item count CDATA #IMPLIED >

<inventory>
   <item count="115"> Apples </count>
   <item count="-2" > Oranges </count>
   <item count="1.2"> Grapes </count>
   <item count="joe"> Bananas </count>
</inventory
```

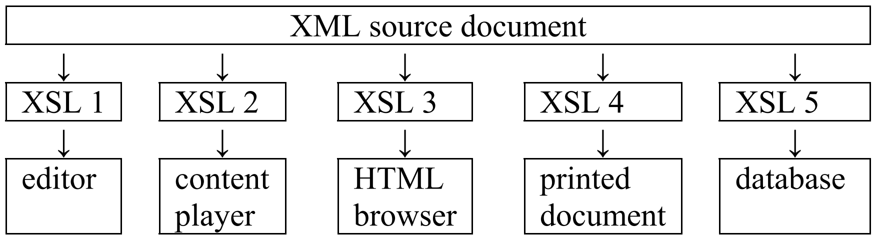
The XML is valid according to the DTD. But there is some data that is either obviously wrong ("joe") or probably wrong ("-2" and "1.2")… it would be nice to be able to specify a data type in the document grammar so that type-checking can be performed for us.

- 2-May-2001: XML Schema becomes a W3C Recommendation
- Schemas are basically a DTD expressed in XML
    - Includes data types
- http://www.w3.org/XML/Schema

B. How do you view an XML document?

- Question:
  If anyone can make up tags, how do you view a document? For example, how can Netscape render our document if it only renders HTML?

- Answer: XSL: Extensible Stylesheet Language
  Formatting instructions for your application's markup.



C. Namespaces

If everyone is free to make up their own tags, chances are two applications will come up with the same name but slightly different meanings (attributes and rendering). Solution is to use a *namespace*, the equivalent of giving your tags a first and last name. This way you can use the tags from the two applications in your own application, without the names clashing.

Give your tags a prefix which is declared in the document:

```
<contacts xmlns:jk ="http://javakathy.com/ns/addressBook/1.0">

       <jk:contact>
           <jk:name> Kathy B </jk:name>
           <jk:email> javakathy@teacher.com </jk:email>
           <jk:address location="home">
               <jk:city> Gotha </jk:city >
               <jk:state> FL </jk:state>
               <jk:zip> 34734 </jk:zip>
           </ jk:address>
       </jk:contact>

</contacts>
```

xmlns	XML Name Space
jk	Prefix we'll use when referring to elements and attributes defined in the DTD at the specified URL.

§6. Quiz Solutions

A. HTML Quiz Solution

Q1.1) Given

```
<h2> Accessories for sale </h2>
```

element `h2` is a container tag.

►a) true
b) false

Q1.1 is true. h2 has a start tag and an end tag, with the content in between.

Q1.2) Given

```
<a href="prices.html">Prices </a>
```

element `a` is an empty element.

a) true
►b) false

Q1.2 is false. The anchor, Prices, is not specified by an attribute. An empty element has all content specified by attributes

Q1.3) Given

```
<img src="house.jpg" height="300" width="300"
alt="house" />
```

the element name is:

►a) `img`
b) `src`
c) `height`
d) `width`
e) `house`

In Q1.3, The element name is the tag name, `img`.

Q1.4) Given

```
<hr align="center" />
```

the element `hr` is an empty element.

►a) true
b) false

Q4) is *true*. All of the content of element hr is specified by attributes. Notice the forward slash used to close empty elements.

Q1.5) Given

```
<input type="radio" name="Q1" value="true">
```

the attributes are:

a) `input`
►b) `type`
c) `radio`
►d) `name`
e) `Q1`
►f) `value`
g) `true`

The attributes are `type`, `name`, and `value`.

B. XML Quiz 1

Q2.1) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil>
    <head>  </head>
    <body>  </body>
</smil>
```

►a) true
b) false

Q2.1 is *true*. The attribute `id` is implied, not required, so it's okay to leave it out. Element `smil`'s sub-elements `head` and `body` are specified to appear once or not at all, with the `head` element before the `body` element..

Q2.2) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil id="slideshow">
    <body>
    </body>
</smil>
```

►a) true
b) false

Q2.2 is *true*. Element `smil` has the attribute `id`. The sub-element `head` must appear once or not at all, so leaving it out is valid.

Q2.3) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil id="preview">
    <body>  </body>
    <head>  </head>
</smil>
```

a) true
►b) false

Q2.3 is *false*. Element `smil`'s sub-elements appear in the wrong order.

Q2.4) Given the DTD fragment

```
<!ENTITY % id-attr "id ID #IMPLIED">
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil %id-attr;>
```

the following markup is valid

```
<smil id="demo">
    <head> <body> </head> </body>
</smil>
```

a) true
►b) false

Q2.4 is *false*. Element `smil's` sub-element tags are not nested properly

C. XML Quiz 2

Q3.1) Given the DTD fragment

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
          name     NMTOKEN   #REQUIRED
          content  CDATA     #REQUIRED >
```

the following markup is valid

```
<meta />
```

a) true
►b) false

Q3.1 is *false*.
Attributes name and content are required attributes.

Q3.2) Given the DTD fragment

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
          name     NMTOKEN   #REQUIRED
          content  CDATA     #REQUIRED >
```

the following markup is valid

```
<meta name="author" content="Kathy" >
```

a) true
►b) false

Q3.2 is *false*.
The required attributes, name and content, are correctly specified, but the empty element closing forward slash is missing.

Q3.3) Given the DTD fragment

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
          name     NMTOKEN   #REQUIRED
          content  CDATA     #REQUIRED >
```

the following markup is valid

```
<meta content="javakathy" name="copyright"  />
```

►a) true
b) false

Q3.3 is *true*.
The required attributes, name and content, are correctly specified (order doesn't matter), and the empty element tag is closed.

D. XML Quiz 3

Q4.1) Given the DTD fragment

```
<!ENTITY % viewport-attrs "
          height           CDATA   #IMPLIED
          width            CDATA   #IMPLIED
          background-color CDATA   #IMPLIED
">
<!ELEMENT region EMPTY>
<!ATTLIST region
         %id-attr;
         %title-attr;
         %viewport-attrs;
         left    CDATA   "0"
         top     CDATA   "0"
         z-index CDATA   "0"
         fit (hidden|fill|meet|scroll|slice) "hidden"
         %skip-attr;     >
```

the name of the tag is

a) `viewport-attrs`
►b) `region`
c) `hidden`

Q4.2) the attribute `fit` can have the value 0

a) `true`
►b) `false`

Q4.2 is *false*.
Attribute fit is defined as an enumerated type - the valid values are explicitly listed (enumerated) in the DTD.

Q4.3) the default value of `fit` is

►a) `hidden`
b) `fill`
c) `meet`
d) `scroll`
e) `slice`

Q4.4) the following markup is valid

```
<region height="400" background-color="black" />
```

►a) `true`
b) `false`

Q4.4 is *true*.
Region's attributes include the attributes specified in the entity %viewport-attrs, which contains the optional attributes height, width, and background-color.

§Appendix A: XHTML 1.0 Strict DTD

```
<!--
   Extensible HTML version 1.0 Strict DTD

   This is the same as HTML 4.0 Strict except for
   changes due to the differences between XML and SGML.

   Namespace = http://www.w3.org/1999/xhtml

   For further information, see: http://www.w3.org/TR/xhtml1

   Copyright (c) 1998-2000 W3C (MIT, INRIA, Keio),
   All Rights Reserved.

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

   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"

   $Revision: 1.14 $
   $Date: 2000/01/25 23:52:20 $

-->

<!--================ Character mnemonic entities =========================-->

<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin 1 for XHTML//EN"
   "xhtml-lat1.ent">
%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols for XHTML//EN"
   "xhtml-symbol.ent">
%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special for XHTML//EN"
   "xhtml-special.ent">
%HTMLspecial;
```

```
<!--================== Imported Names ====================================-->

<!ENTITY % ContentType "CDATA">          <!-- media type, as per [RFC2045] -->

<!ENTITY % ContentTypes "CDATA">         <!-- comma-separated list of media types, [RFC2045] -->

<!ENTITY % Charset "CDATA">              <!-- a character encoding, as per [RFC2045] -->

<!ENTITY % Charsets "CDATA">             <!-- space-separated list of char encodings, [RFC2045] -->

<!ENTITY % LanguageCode "NMTOKEN">       <!-- a language code, as per [RFC1766] -->

<!ENTITY % Character "CDATA">            <!-- a single character from [ISO10646] -->

<!ENTITY % Number "CDATA">               <!-- one or more digits -->

<!ENTITY % LinkTypes "CDATA">            <!-- space-separated list of link types -->

<!ENTITY % MediaDesc "CDATA">            <!-- single or comma-separated list of media descriptors -->

<!ENTITY % URI "CDATA">                  <!-- a Uniform Resource Identifier, see [RFC2396] -->

<!ENTITY % UriList "CDATA">              <!-- space separated list of Uniform Resource Identifiers -->

<!ENTITY % Datetime "CDATA">             <!-- date and time information. ISO date format -->

<!ENTITY % Script "CDATA">               <!-- script expression -->

<!ENTITY % StyleSheet "CDATA">           <!-- style sheet data -->

<!ENTITY % Text "CDATA">                 <!-- used for titles etc. -->

<!ENTITY % FrameTarget "NMTOKEN">        <!-- render in this frame -->

<!ENTITY % Length "CDATA">               <!-- nn for pixels or nn% for percentage length -->

<!ENTITY % MultiLength "CDATA">          <!-- pixel, percentage, or relative -->

<!ENTITY % MultiLengths "CDATA">         <!-- comma-separated list of MultiLength -->

<!ENTITY % Pixels "CDATA">               <!-- integer representing length in pixels -->


<!-- these are used for image maps -->

<!ENTITY % Shape "(rect|circle|poly|default)">

<!ENTITY % Coords "CDATA">               <!-- comma separated list of lengths -->
```

```
<!--==================== Generic Attributes ===============================-->

<!-- core attributes common to most elements-->
<!ENTITY % coreattrs
 "id          ID             #IMPLIED     <!-- document-wide unique id -->
  class       CDATA          #IMPLIED     <!-- space separated list of classes -->
  style       %StyleSheet;   #IMPLIED     <!-- associated style info -->
  title       %Text;         #IMPLIED"    <!-- advisory title/amplification -->
  >

<!-- internationalization attributes -->
<!ENTITY % i18n
 "lang        %LanguageCode; #IMPLIED     <!-- language code (backwards compatible) -->
  xml:lang    %LanguageCode; #IMPLIED     <!-- language code (as per XML 1.0 spec) -->
  dir         (ltr|rtl)      #IMPLIED"    <!-- direction for weak/neutral text -->
  >

<!-- attributes for common UI events-->
<!ENTITY % events
 "onclick     %Script;       #IMPLIED     <!-- a pointer button was clicked -->
  ondblclick  %Script;       #IMPLIED     <!-- a pointer button was double clicked -->
  onmousedown %Script;       #IMPLIED     <!-- a pointer button was pressed down -->
  onmouseup   %Script;       #IMPLIED     <!-- a pointer button was released -->
  onmouseover %Script;       #IMPLIED     <!-- a pointer was moved onto the element -->
  onmousemove %Script;       #IMPLIED     <!--  a pointer was moved onto the element -->
  onmouseout  %Script;       #IMPLIED     <!--  a pointer was moved away from the element --
>
  onkeypress  %Script;       #IMPLIED     <!-- a key was pressed and released -->
  onkeydown   %Script;       #IMPLIED     <!-- a key was pressed down -->
  onkeyup     %Script;       #IMPLIED"    <!-- a key was released -->
  >

<!-- attributes for elements that can get the focus -->
<!ENTITY % focus
 "accesskey   %Character;    #IMPLIED     <!-- accessibility key character -->
  tabindex    %Number;       #IMPLIED     <!-- position in tabbing order -->
  onfocus     %Script;       #IMPLIED     <!-- the element got the focus -->
  onblur      %Script;       #IMPLIED"    <!-- the element lost the focus -->
  >

<!ENTITY % attrs "%coreattrs; %i18n; %events;">

<!--==================== Text Elements ===================================-->

<!ENTITY % special      "br | span | bdo | object | img | map">

<!ENTITY % fontstyle    "tt | i | b | big | small">

<!ENTITY % phrase       "em | strong | dfn | code | q | sub | sup |
                         samp | kbd | var | cite | abbr | acronym">

<!ENTITY % inline.forms "input | select | textarea | label | button">

<!-- these can occur at block or inline level -->
<!ENTITY % misc         "ins | del | script | noscript">
```

```
<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;">

<!-- %Inline; covers inline or "text-level" elements -->
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">


<!--================== Block level elements ================================-->

<!ENTITY % heading      "h1|h2|h3|h4|h5|h6">
<!ENTITY % lists        "ul | ol | dl">
<!ENTITY % blocktext    "pre | hr | blockquote | address">

<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; | fieldset | table">

<!ENTITY % Block        "(%block; | form | %misc;)*">

<!-- %Flow;  mixes Block and  Inline and is used for list items etc. -->
<!ENTITY % Flow "(#PCDATA | %block; | form | %inline; | %misc;)*">


<!--================== Content models for exclusions =====================-->

<!-- a elements use %Inline; excluding a -->

<!ENTITY % a.content
   "(#PCDATA | %special; | %fontstyle; | %phrase; | %inline.forms; | %misc;)*">

<!-- pre uses %Inline excluding img, object, big, small, sup or sup -->

<!ENTITY % pre.content
   "(#PCDATA | a | br | span | bdo | map | tt | i | b |
      %phrase; | %inline.forms;)*">

<!-- form uses %Block; excluding form -->

<!ENTITY % form.content "(%block; | %misc;)*">

<!-- button uses %Flow; but excludes a, form and form controls -->

<!ENTITY % button.content
   "(#PCDATA | p | %heading; | div | %lists; | %blocktext; |
    table | %special; | %fontstyle; | %phrase; | %misc;)*">


<!--================ Document Structure ==================================-->

<!-- the namespace URI designates the document profile -->

<!ELEMENT html (head, body)>
<!ATTLIST html
  %i18n;
  xmlns       %URI;          #FIXED 'http://www.w3.org/1999/xhtml'
  >
```

```
<!--================= Document Head =======================================-->

<!ENTITY % head.misc "(script|style|meta|link|object)*">

<!-- content model is %head.misc; combined with a single
     title and an optional base element in any order -->

<!ELEMENT head (%head.misc;,
      ((title, %head.misc;, (base, %head.misc;)?) |
       (base, %head.misc;, (title, %head.misc;))))>

<!ATTLIST head
  %i18n;
  profile     %URI;          #IMPLIED
  >

<!-- The title element is not considered part of the flow of text.  It should be displayed, for example as the
     page header or window title. Exactly one title is required per document.
  -->
<!ELEMENT title (#PCDATA)>
<!ATTLIST title %i18n;>

<!-- document base URI -->

<!ELEMENT base EMPTY>
<!ATTLIST base
  href        %URI;          #IMPLIED
  >

<!-- generic metainformation -->
<!ELEMENT meta EMPTY>
<!ATTLIST meta
  %i18n;
  http-equiv  CDATA          #IMPLIED
  name        CDATA          #IMPLIED
  content     CDATA          #REQUIRED
  scheme      CDATA          #IMPLIED
  >

<!-- Relationship values can be used in principle:

   a)  for document specific toolbars/menus when used with the link element in document head e.g.
      start, contents, previous, next, index, end, help
   b)  to link to a separate style sheet (rel="stylesheet")
   c)  to make a link to a script (rel="script")
   d)  by stylesheets to control how collections of html nodes are rendered into printed documents
   e)  to make a link to a printable version of this document
     e.g. a PostScript or PDF version (rel="alternate" media="print")
-->
```

```
<!ELEMENT link EMPTY>
<!ATTLIST link
  %attrs;
  charset     %Charset;      #IMPLIED
  href        %URI;          #IMPLIED
  hreflang    %LanguageCode; #IMPLIED
  type        %ContentType;  #IMPLIED
  rel         %LinkTypes;    #IMPLIED
  rev         %LinkTypes;    #IMPLIED
  media       %MediaDesc;    #IMPLIED
  >
```

<!-- style info, which may include CDATA sections -->
```
<!ELEMENT style (#PCDATA)>
<!ATTLIST style
  %i18n;
  type        %ContentType;  #REQUIRED
  media       %MediaDesc;    #IMPLIED
  title       %Text;         #IMPLIED
  xml:space   (preserve)     #FIXED 'preserve'
  >
```

<!-- script statements, which may include CDATA sections -->
```
<!ELEMENT script (#PCDATA)>
<!ATTLIST script
  charset     %Charset;      #IMPLIED
  type        %ContentType;  #REQUIRED
  src         %URI;          #IMPLIED
  defer       (defer)        #IMPLIED
  xml:space   (preserve)     #FIXED 'preserve'
  >
```

<!-- alternate content container for non script-based rendering -->
```
<!ELEMENT noscript %Block;>
<!ATTLIST noscript
  %attrs;
  >
```

<!--=================== Document Body ====================================-->

```
<!ELEMENT body %Block;>
<!ATTLIST body
  %attrs;
  onload          %Script;   #IMPLIED
  onunload        %Script;   #IMPLIED
  >

<!ELEMENT div %Flow;>  <!-- generic language/style container -->
<!ATTLIST div
  %attrs;
  >
```

<!--=================== Paragraphs =======================================-->

```
<!ELEMENT p %Inline;>
<!ATTLIST p
  %attrs;
  >
```

```
<!--=================== Headings =========================================-->

<!--  There are six levels of headings from h1 (the most important) to h6 (the least important). -->

<!ELEMENT h1  %Inline;>
<!ATTLIST h1
   %attrs;
   >

<!ELEMENT h2 %Inline;>
<!ATTLIST h2
   %attrs;
   >

<!ELEMENT h3 %Inline;>
<!ATTLIST h3
   %attrs;
   >

<!ELEMENT h4 %Inline;>
<!ATTLIST h4
   %attrs;
   >

<!ELEMENT h5 %Inline;>
<!ATTLIST h5
   %attrs;
   >

<!ELEMENT h6 %Inline;>
<!ATTLIST h6
   %attrs;
   >

<!--=================== Lists ============================================-->

<!ELEMENT ul (li)+>                  <!-- Unordered list -->
<!ATTLIST ul
  %attrs;
  >

<!ELEMENT ol (li)+>                  <!-- Ordered (numbered) list -->
<!ATTLIST ol
  %attrs;
  >

<!ELEMENT li %Flow;>                 <!-- list item -->
<!ATTLIST li
  %attrs;
  >

<!ELEMENT dl (dt|dd)+>               <!-- definition lists -->
<!ATTLIST dl
  %attrs;
  >
```

```
<!ELEMENT dt %Inline;>                <!—use dt for definition list term -->
<!ATTLIST dt
  %attrs;
  >

<!ELEMENT dd %Flow;>                  <!—use dd for term's definition -->
<!ATTLIST dd
  %attrs;
  >

<!--==================== Address ============================================-->

<!ELEMENT address %Inline;>           <!-- information on author -->
<!ATTLIST address
  %attrs;
  >

<!--==================== Horizontal Rule ====================================-->

<!ELEMENT hr EMPTY>
<!ATTLIST hr
  %attrs;
  >

<!--==================== Preformatted Text ==================================-->

<!-- content is %Inline; excluding "img|object|big|small|sub|sup" -->

<!ELEMENT pre %pre.content;>
<!ATTLIST pre
  %attrs;
  xml:space (preserve) #FIXED 'preserve'
  >

<!--==================== Block-like Quotes ==================================-->

<!ELEMENT blockquote %Block;>
<!ATTLIST blockquote
  %attrs;
  cite        %URI;          #IMPLIED
  >

<!--==================== Inserted/Deleted Text ==============================-->
<!--
 ins/del are allowed in block and inline content, but it is inappropriate to include block content within an
ins element occurring in inline content.
-->
<!ELEMENT ins %Flow;>
<!ATTLIST ins
  %attrs;
  cite        %URI;          #IMPLIED
  datetime    %Datetime;     #IMPLIED
  >
```

```
<!ELEMENT del %Flow;>
<!ATTLIST del
  %attrs;
  cite        %URI;          #IMPLIED
  datetime    %Datetime;     #IMPLIED
  >

<!--=================== The Anchor Element ================================-->
<!-- content is %Inline; except that anchors shouldn't be nested -->

<!ELEMENT a %a.content;>
<!ATTLIST a
  %attrs;
  charset     %Charset;      #IMPLIED
  type        %ContentType;  #IMPLIED
  name        NMTOKEN        #IMPLIED
  href        %URI;          #IMPLIED
  hreflang    %LanguageCode; #IMPLIED
  rel         %LinkTypes;    #IMPLIED
  rev         %LinkTypes;    #IMPLIED
  accesskey   %Character;    #IMPLIED
  shape       %Shape;        "rect"
  coords      %Coords;       #IMPLIED
  tabindex    %Number;       #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  >

<!--===================== Inline Elements ================================-->

<!ELEMENT span %Inline;>                  <!-- generic language/style container -->
<!ATTLIST span
  %attrs;
  >

<!ELEMENT bdo %Inline;>                   <!-- I18N BiDi over-ride -->
<!ATTLIST bdo
  %coreattrs;
  %events;
  lang        %LanguageCode; #IMPLIED
  xml:lang    %LanguageCode; #IMPLIED
  dir         (ltr|rtl)      #REQUIRED
  >

<!ELEMENT br EMPTY>                       <!-- forced line break -->
<!ATTLIST br
  %coreattrs;
  >

<!ELEMENT em %Inline;>                    <!-- emphasis -->
<!ATTLIST em %attrs;>

<!ELEMENT strong %Inline;>                <!-- strong emphasis -->
<!ATTLIST strong %attrs;>
```

```
<!ELEMENT dfn %Inline;>                          <!-- definitional -->
<!ATTLIST dfn %attrs;>

<!ELEMENT code %Inline;>                         <!-- program code -->
<!ATTLIST code %attrs;>

<!ELEMENT samp %Inline;>                         <!-- sample -->
<!ATTLIST samp %attrs;>

<!ELEMENT kbd %Inline;>                          <!-- something user would type -->
<!ATTLIST kbd %attrs;>

<!ELEMENT var %Inline;>                          <!-- variable -->
<!ATTLIST var %attrs;>

<!ELEMENT cite %Inline;>                         <!-- citation -->
<!ATTLIST cite %attrs;>

<!ELEMENT abbr %Inline;>                         <!-- abbreviation -->
<!ATTLIST abbr %attrs;>

<!ELEMENT acronym %Inline;>                      <!-- acronym -->
<!ATTLIST acronym %attrs;>

<!ELEMENT q %Inline;>                            <!-- inlined quote -->
<!ATTLIST q
  %attrs;
  cite        %URI;          #IMPLIED
  >

<!ELEMENT sub %Inline;>                          <!-- subscript -->
<!ATTLIST sub %attrs;>

<!ELEMENT sup %Inline;>                          <!-- superscript -->
<!ATTLIST sup %attrs;>

<!ELEMENT tt %Inline;>                           <!-- fixed pitch font -->
<!ATTLIST tt %attrs;>

<!ELEMENT i %Inline;>                            <!-- italic font -->
<!ATTLIST i %attrs;>

<!ELEMENT b %Inline;>                            <!-- bold font -->
<!ATTLIST b %attrs;>

<!ELEMENT big %Inline;>                          <!-- bigger font -->
<!ATTLIST big %attrs;>

<!ELEMENT small %Inline;>                        <!-- smaller font -->
<!ATTLIST small %attrs;>
```

```
<!--==================== Object ======================================-->
<!--
     object is used to embed objects as part of HTML pages.  param elements should precede other
     content. Parameters  can also be expressed as attribute/value pairs on the object element itself when
     brevity is desired.
-->
<!ELEMENT object (#PCDATA | param | %block; | form | %inline; | %misc;)*>
<!ATTLIST object
  %attrs;
  declare     (declare)       #IMPLIED
  classid     %URI;           #IMPLIED
  codebase    %URI;           #IMPLIED
  data        %URI;           #IMPLIED
  type        %ContentType;   #IMPLIED
  codetype    %ContentType;   #IMPLIED
  archive     %UriList;       #IMPLIED
  standby     %Text;          #IMPLIED
  height      %Length;        #IMPLIED
  width       %Length;        #IMPLIED
  usemap      %URI;           #IMPLIED
  name        NMTOKEN         #IMPLIED
  tabindex    %Number;        #IMPLIED
  >

<!—
      param is used to supply a named property value.  In XML it would seem natural to follow RDF and
     support an abbreviated syntax where the param elements are replaced by attribute value pairs on the
     object start tag.
-->
<!ELEMENT param EMPTY>
<!ATTLIST param
  id          ID              #IMPLIED
  name        CDATA           #IMPLIED
  value       CDATA           #IMPLIED
  valuetype   (data|ref|object) "data"
  type        %ContentType;   #IMPLIED
  >

<!--=================== Images ===========================================-->
<!--
     To avoid accessibility problems for people who aren't able to see the image, you should provide a text
     description using the alt and longdesc attributes. In addition, avoid the use of server-side image maps.
     Note that in this DTD there is no name attribute. That is only available in the transitional and frameset
     DTD.
-->
<!ELEMENT img EMPTY>
<!ATTLIST img
  %attrs;
  src         %URI;           #REQUIRED
  alt         %Text;          #REQUIRED
  longdesc    %URI;           #IMPLIED
  height      %Length;        #IMPLIED
  width       %Length;        #IMPLIED
  usemap      %URI;           #IMPLIED
  ismap       (ismap)         #IMPLIED
  >
```

```
<!-- usemap points to a map element which may be in this document
  or an external document, although the latter is not widely supported -->

<!--=================== Client-side image maps ============================-->

<!-- These can be placed in the same document or grouped in a
     separate document although this isn't yet widely supported -->

<!ELEMENT map ((%block; | form | %misc;)+ | area+)>
<!ATTLIST map
  %i18n;
  %events;
  id          ID             #REQUIRED
  class       CDATA          #IMPLIED
  style       %StyleSheet;   #IMPLIED
  title       %Text;         #IMPLIED
  name        NMTOKEN        #IMPLIED
  >

<!ELEMENT area EMPTY>
<!ATTLIST area
  %attrs;
  shape       %Shape;        "rect"
  coords      %Coords;       #IMPLIED
  href        %URI;          #IMPLIED
  nohref      (nohref)       #IMPLIED
  alt         %Text;         #REQUIRED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  >

<!--================ Forms ===============================================-->
<!ELEMENT form %form.content;>   <!-- forms shouldn't be nested -->

<!ATTLIST form
  %attrs;
  action          %URI;          #REQUIRED
  method          (get|post)     "get"
  enctype         %ContentType;  "application/x-www-form-urlencoded"
  onsubmit        %Script;       #IMPLIED
  onreset         %Script;       #IMPLIED
  accept          %ContentTypes; #IMPLIED
  accept-charset  %Charsets;     #IMPLIED
  >

<!--  Each label must not contain more than ONE field  Label elements shouldn't be nested. -->
<!ELEMENT label %Inline;>
<!ATTLIST label
  %attrs;
  for         IDREF          #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  >
```

```
<!ENTITY % InputType
  "(text | password | checkbox | radio | submit | reset |
    file | hidden | image | button)"
   >
```

<!-- the `name` attribute is required for all but `submit & reset` -->

```
<!ELEMENT input EMPTY>     <!-- form control -->
<!ATTLIST input
  %attrs;
  type        %InputType;    "text"
  name        CDATA          #IMPLIED
  value       CDATA          #IMPLIED
  checked     (checked)      #IMPLIED
  disabled    (disabled)     #IMPLIED
  readonly    (readonly)     #IMPLIED
  size        CDATA          #IMPLIED
  maxlength   %Number;       #IMPLIED
  src         %URI;          #IMPLIED
  alt         CDATA          #IMPLIED
  usemap      %URI;          #IMPLIED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  onselect    %Script;       #IMPLIED
  onchange    %Script;       #IMPLIED
  accept      %ContentTypes; #IMPLIED
  >

<!ELEMENT select (optgroup|option)+>  <!-- option selector -->
<!ATTLIST select
  %attrs;
  name        CDATA          #IMPLIED
  size        %Number;       #IMPLIED
  multiple    (multiple)     #IMPLIED
  disabled    (disabled)     #IMPLIED
  tabindex    %Number;       #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  onchange    %Script;       #IMPLIED
  >

<!ELEMENT optgroup (option)+>   <!-- option group -->
<!ATTLIST optgroup
  %attrs;
  disabled    (disabled)     #IMPLIED
  label       %Text;         #REQUIRED
  >

<!ELEMENT option (#PCDATA)>     <!-- selectable choice -->
<!ATTLIST option
  %attrs;
  selected    (selected)     #IMPLIED
  disabled    (disabled)     #IMPLIED
  label       %Text;         #IMPLIED
  value       CDATA          #IMPLIED
  >
```

```
<!ELEMENT textarea (#PCDATA)>     <!-- multi-line text field -->
<!ATTLIST textarea
  %attrs;
  name        CDATA          #IMPLIED
  rows        %Number;       #REQUIRED
  cols        %Number;       #REQUIRED
  disabled    (disabled)     #IMPLIED
  readonly    (readonly)     #IMPLIED
  tabindex    %Number;       #IMPLIED
  accesskey   %Character;    #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED
  onselect    %Script;       #IMPLIED
  onchange    %Script;       #IMPLIED
  >

<!--
     The fieldset element is used to group form fields. Only one legend element
   should occur in the content and if present should only be preceded by
   whitespace.
-->
<!ELEMENT fieldset (#PCDATA | legend | %block; | form | %inline; | %misc;)*>
<!ATTLIST fieldset
  %attrs;
  >

<!ELEMENT legend %Inline;>     <!-- fieldset label -->
<!ATTLIST legend
  %attrs;
  accesskey   %Character;    #IMPLIED
  >

<!--
 Content is %Flow; excluding a, form and form controls
-->
<!ELEMENT button %button.content;>  <!-- push button -->
<!ATTLIST button
  %attrs;
  name        CDATA                    #IMPLIED
  value       CDATA                    #IMPLIED
  type        (button|submit|reset)    "submit"
  disabled    (disabled)               #IMPLIED
  tabindex    %Number;                 #IMPLIED
  accesskey   %Character;              #IMPLIED
  onfocus     %Script;                 #IMPLIED
  onblur      %Script;                 #IMPLIED
  >
```

```
<!--======================= Tables =======================================-->
<!-- Derived from IETF HTML table standard, see [RFC1942] -->
<!--
  The border attribute sets the thickness of the frame around the table. The default units are screen
  pixels.

  The frame attribute specifies which parts of the frame around the table should be rendered. The values
  are not the same as CALS to avoid a name clash with the valign attribute.
-->

<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">

<!--
 The rules attribute defines which rules to draw between cells:

 If rules is absent then assume:
    "none" if border is absent or border="0" otherwise "all"
-->

<!ENTITY % TRules "(none | groups | rows | cols | all)">

<!-- horizontal placement of table relative to document -->
<!ENTITY % TAlign "(left|center|right)">

<!-- horizontal alignment attributes for cell contents

  char       alignment char, e.g. char=':'
  charoff    offset for alignment char
-->
<!ENTITY % cellhalign
  "align       (left|center|right|justify|char) #IMPLIED
   char        %Character;     #IMPLIED
   charoff     %Length;        #IMPLIED"
  >

<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cellvalign
  "valign      (top|middle|bottom|baseline) #IMPLIED"
  >

<!ELEMENT table
     (caption?, (col*|colgroup*), thead?, tfoot?, (tbody+|tr+))>
<!ELEMENT caption  %Inline;>
<!ELEMENT thead    (tr)+>
<!ELEMENT tfoot    (tr)+>
<!ELEMENT tbody    (tr)+>
<!ELEMENT colgroup (col)*>
<!ELEMENT col      EMPTY>
<!ELEMENT tr       (th|td)+>
<!ELEMENT th       %Flow;>
<!ELEMENT td       %Flow;>
```

```
<!ATTLIST table
  %attrs;
  summary     %Text;         #IMPLIED
  width       %Length;       #IMPLIED
  border      %Pixels;       #IMPLIED
  frame       %TFrame;       #IMPLIED
  rules       %TRules;       #IMPLIED
  cellspacing %Length;       #IMPLIED
  cellpadding %Length;       #IMPLIED
  >

<!ENTITY % CAlign "(top|bottom|left|right)">

<!ATTLIST caption
  %attrs;
  >
```

<!-- colgroup groups a set of col elements. It allows you to group several semantically related columns together.
-->

```
<!ATTLIST colgroup
  %attrs;
  span            %Number;       "1"
  width           %MultiLength;  #IMPLIED
  %cellhalign;
  %cellvalign;
  >
```

<!--
col elements define the alignment properties for cells in one or more columns.

The width attribute specifies the width of the columns, e.g.

width=64	width in screen pixels
width=0.5*	relative width of 0.5

The span attribute causes the attributes of one col element to apply to more than one column.
-->

```
<!ATTLIST col
  %attrs;
  span            %Number;       "1"
  width           %MultiLength;  #IMPLIED
  %cellhalign;
  %cellvalign;
  >
```

<!--
Use thead to duplicate headers when breaking table across page boundaries, or for static headers when tbody sections are rendered in scrolling panel.

Use tfoot to duplicate footers when breaking table across page boundaries, or for static footers when tbody sections are rendered in scrolling panel.

Use multiple tbody sections when rules are needed between groups of table rows.
-->

```
<!ATTLIST thead
  %attrs;
  %cellhalign;
  %cellvalign;
  >

<!ATTLIST tfoot
  %attrs;
  %cellhalign;
  %cellvalign;
  >

<!ATTLIST tbody
  %attrs;
  %cellhalign;
  %cellvalign;
  >

<!ATTLIST tr
  %attrs;
  %cellhalign;
  %cellvalign;
  >
```

<!-- Scope is simpler than headers attribute for common tables -->

```
<!ENTITY % Scope "(row|col|rowgroup|colgroup)">
```

<!-- th is for headers, td for data and for cells acting as both -->

```
<!ATTLIST th
  %attrs;
  abbr        %Text;         #IMPLIED
  axis        CDATA          #IMPLIED
  headers     IDREFS         #IMPLIED
  scope       %Scope;        #IMPLIED
  rowspan     %Number;       "1"
  colspan     %Number;       "1"
  %cellhalign;
  %cellvalign;
  >

<!ATTLIST td
  %attrs;
  abbr        %Text;         #IMPLIED
  axis        CDATA          #IMPLIED
  headers     IDREFS         #IMPLIED
  scope       %Scope;        #IMPLIED
  rowspan     %Number;       "1"
  colspan     %Number;       "1"
  %cellhalign;
  %cellvalign;
  >
```

Intro to SMIL
Synchronized Multimedia Integration Language

SMIL 2.0 errata at
http://www.w3.org/2001/07/REC-SMIL20-20010731-errata#E31

§1. Background

1.1 Purpose of this Course

SMIL (Synchronized Multimedia Integration Language) is an XML-based language that allows authors to write interactive multimedia presentations. An author can specify the temporal behavior of a presentation, associate hyperlinks with media objects and describe the layout of the presentation. SMIL is a standard by the World Wide Web Consortium (W3C). Participants in this course will learn the syntax of SMIL to author their own interactive presentations.

1.2 What is SMIL good for?

- Position content anywhere in your layout
- Synchronize the timing of the elements
- Display media to suit end-user's language, bit-rate, screen size, etc
- SMIL 2.0 is organized into modules which can be plugged into other XML-based languages. For example, SVG incorporates the SMIL 2 Animation Module.

§2. SMIL Players

PC/Macintosh

RealOnePlayer
  http://www.real.com

QuickTime
  http://www.apple.com/quicktime

GR*i*NS
  http://www.oratrix.com

Unix

RealPlayer Community Supported Player
  http://realforum.real.com/cgi-bin/unixplayer/wwwthreads.pl
  http://proforma.real.com/real/player/unix/unix.html

2.1 Embedded Documents

RealOnePlayer

Example: Display smil document from inside an html page, using RealOnePlayer plugin.

```
<OBJECT ID=RAOCX
        CLASSID="clsid:CFCDAA03-8BE4-11cf-B84B-0020AFBBCCFA"
        height="400" width="460">
      <param name="controls" value="ImageWindow" />
      <param name="autostart" value="true" />
      <param name="src" value="ex2_region.smil" />
      <embed height="400" width="460" controls="ImageWindow"
          src="ex2_region.smil"
          type="audio/x-pn-realaudio-plugin"
          autostart="true" />
</OBJECT>
```

QuickTime Player

```
<OBJECT CLASSID="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
        width="460" height="418"
        CODEBASE="http://www.apple.com/qtactivex/qtplugin.cab">
          <PARAM name="SRC" VALUE="ex2_region.smil">
          <PARAM name="AUTOPLAY" VALUE="true">
          <PARAM name="CONTROLLER" VALUE="false">
          <embed src="ex2_region.smil"
                 width="460" height="418"
                 type="video/quicktime"
                 autoplay="true" controller="true" loop="false"
       pluginspage="http://www.apple.com/quicktime/download" />
<OBJECT>
```

Information on the QuickTime Active-X Plug-in required for Internet Explorer 6 is located at `http://www.apple.com/quicktime/products/tutorials/activex.html`.

To check if you have the plug-in installed, see `http://www.apple.com/quicktime/download/qtcheck/`.

2.2 Standalone document

Example: Document is a ".smil" file viewed directly, by RealOnePlayer or QuickTime Player.

```
<smil>
    <!-- content -->
</smil>
```

§3. Basic SMIL Document Structure

3.1 Basic Structure of SMIL Document

```
<!-- DOCTYPE and DTD information -->
<smil>
   <head>
      <!-- information about presentation -->
      <meta name="Author" content="Kathy B" />
      <layout>
         <!-- main window information -->
         <!-- subregion height, width, & location -->
      </layout>
   </head>

   <body>
      <!-- multimedia presentation -->
   </body>
</smil>
```

`<smil>` is the top-level element that contains the entire document. Just as with HTML, it has 2 subsections: `<head>` and `<body>`. The `<head>` section contains information about the presentation. The `<body>` section contains the content of the presentation.

Comments are delimited by **`<!--`** and **`-->`** as in HTML and XML

3.2 Document Type Declaration (DOCTYPE)

The Document Type Declaration (`DOCTYPE`) goes before the top-level element. For SMIL 1.0, the `DOCTYPE` is

```
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
   "http://www.w3.org/TR/REC-smil/SMIL10.dtd">
```

§4. `<head>`

4.1 <layout>

A. Window Layout

a) `root-layout`

- Main window for your presentation
- Attributes

1. `width`, `height`	In pixels
2. `background-color`	(optional)

- Examples

```
<root-layout width="640" height="480" />

<root-layout width="640" height="480"
         background-color="black" />
```

b) `region`

- Sub-areas within main window where we'll place our media
- Origin (0,0) is at top left corner of main window.
- Attributes

1. `width`, `height`	In pixels, or as % of main window
2. `top`, `left`	In pixels, or as % of main window
3. `id`	Name by which we'll refer to region
4. `background-color`	(optional)
5. `z-index`	(optional)
6. `fit`	(optional)

(1) Exact position

Specify top left corner in pixels from main top left corner.

```
<region id="photo_region"
        width="640" height="480"
        top="20" left="80"
        background-color="black" />
```

(2) Relative position

Specify top left corner as % from main top left corner.

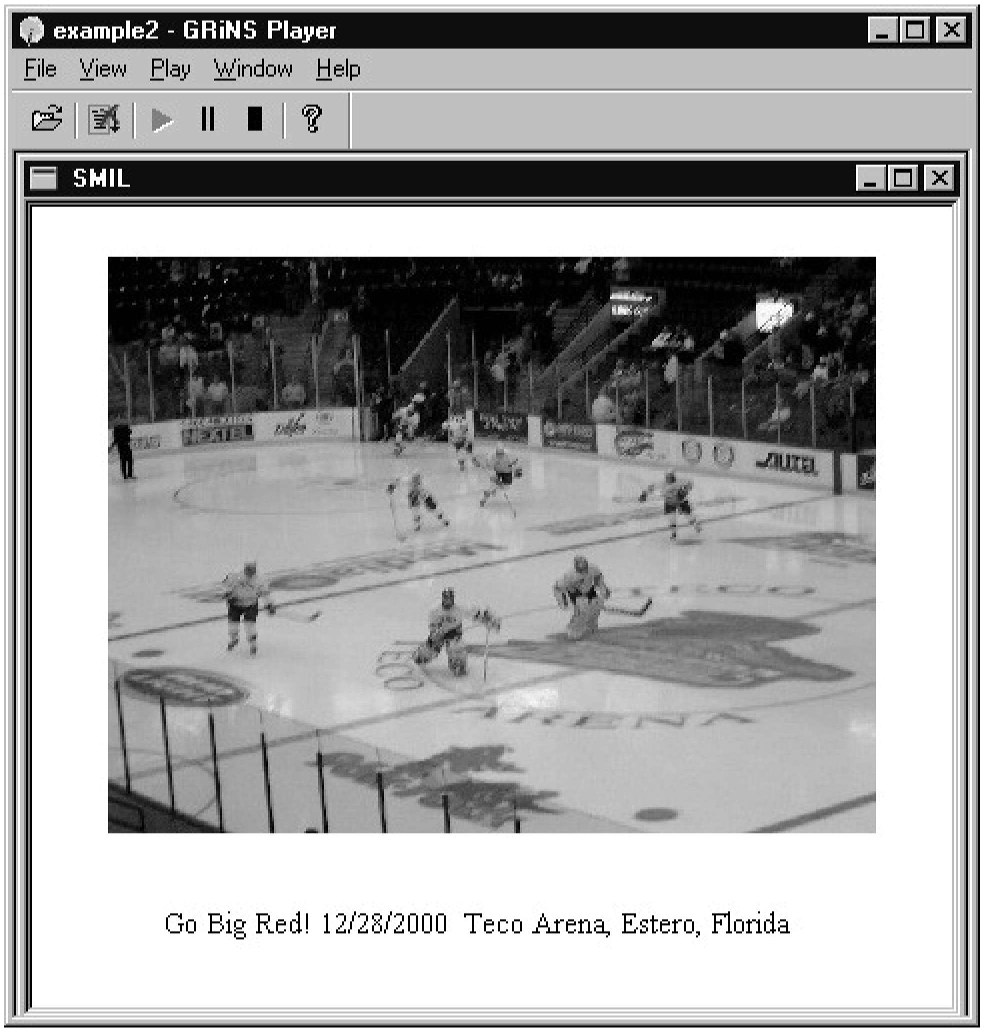
```
<region id="titlebar"
        width="200" height="100"
        top="25%" left="50%" />
```

Region starts a quarter of the way down from the top of main window, and halfway over from the left.

SMIL Ex/ex2/ex2_region.smil

An image and a caption displayed in their own regions.

```
<smil>
  <head>
      <meta name="Author" content="Kathy Barshatzky" />
      <meta name="Copyright" content="javakathy.com" />
      <layout>
         <root-layout width="460" height="400" />
         <region id="photo_region"
                    width="384" height="288" top="25" left="38" />
         <region id="caption_region"
                    width="340" height="40" top="350" left="60" />
      </layout>
  </head>
  <body>
      <par>
         <img src="hockeyTeco.jpg" alt="hockey game image"
             region="photo_region" dur="indefinite" />
         <text src="caption2.txt" alt="Teco Arena, Dec 2000"
             region="caption_region" dur="indefinite" />
      </par>
  </body>
</smil>
```



Go Big Red! 12/28/2000 Teco Arena, Estero, Florida

(3) Overlapping regions

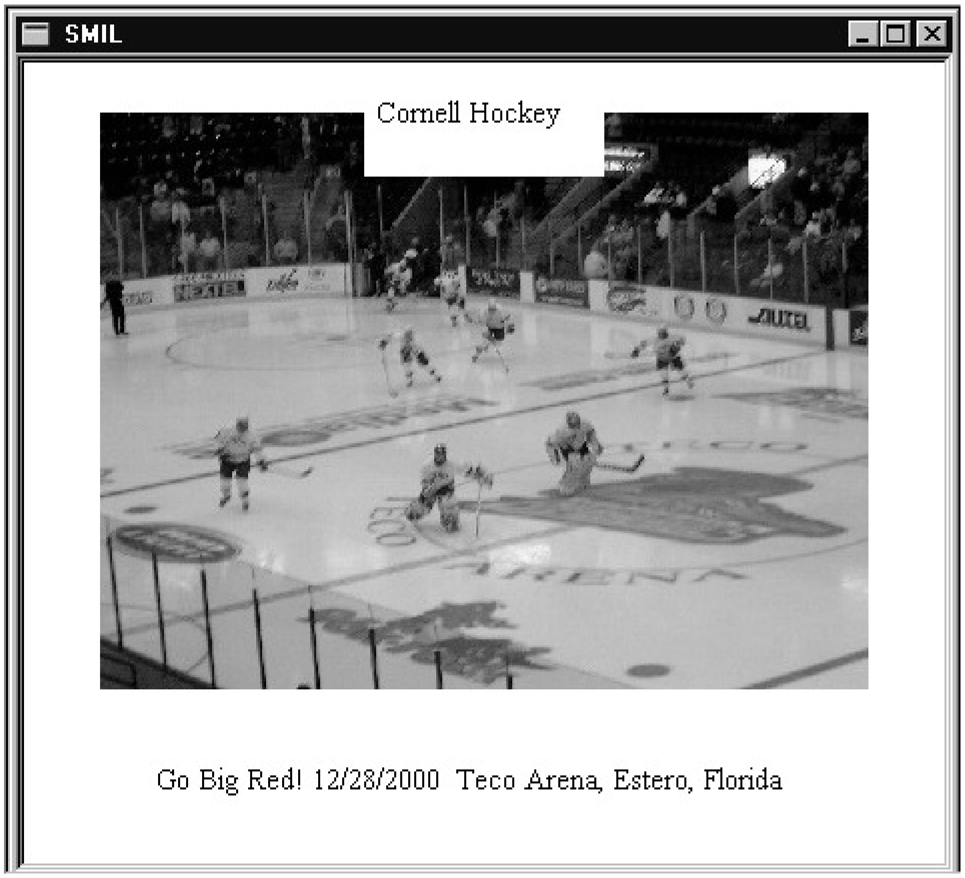
You are allowed to overlap regions. The **`z-index`** attribute, which is optional and has a value of 0 by default, determines which region will be on top.

- Region with the greater `z-index` goes on top
- If two or more regions have the same z-index, the first region encountered goes on the bottom, and subsequent regions are placed on top.

SMIL Ex/ex3/ex3a_overlap.smil

```
<!-- In this example, the title region is given a z-index
greater than the default.  Thus during rendering, it will
be placed over any overlapping regions of lower z-index.
-->
<smil>
  <head>
      <meta name="Author" content="Kathy Barshatzky" />
      <meta name="Copyright" content="javakathy.com" />
      <layout>
         <root-layout width="460" height="400" />
         <region id="photo_region"
            width="384" height="288" top="25" left="38" />
         <region id="caption_region"
            width="340" height="40" top="350" left="60" />
         <region id="title"
            width="120" height="40" top="17" left="170"
            background-color="white"
            z-index="10" />
      </layout>
  </head>
  <body>
      <par>
         <text src="title.txt" alt="Cornell Hockey"
             region="title" dur="indefinite"/>
         <img src="hockeyTeco.jpg" alt="hockey image"
             region="photo_region" dur="indefinite" />
         <text src="caption2.txt"
             region="caption_region" dur="indefinite"/>
      </par>
  </body>
</smil>
```

`SMIL_Ex/ex3/ex3a_overlap.smil` Screen Capture



The region with the title is placed over the region with the image.

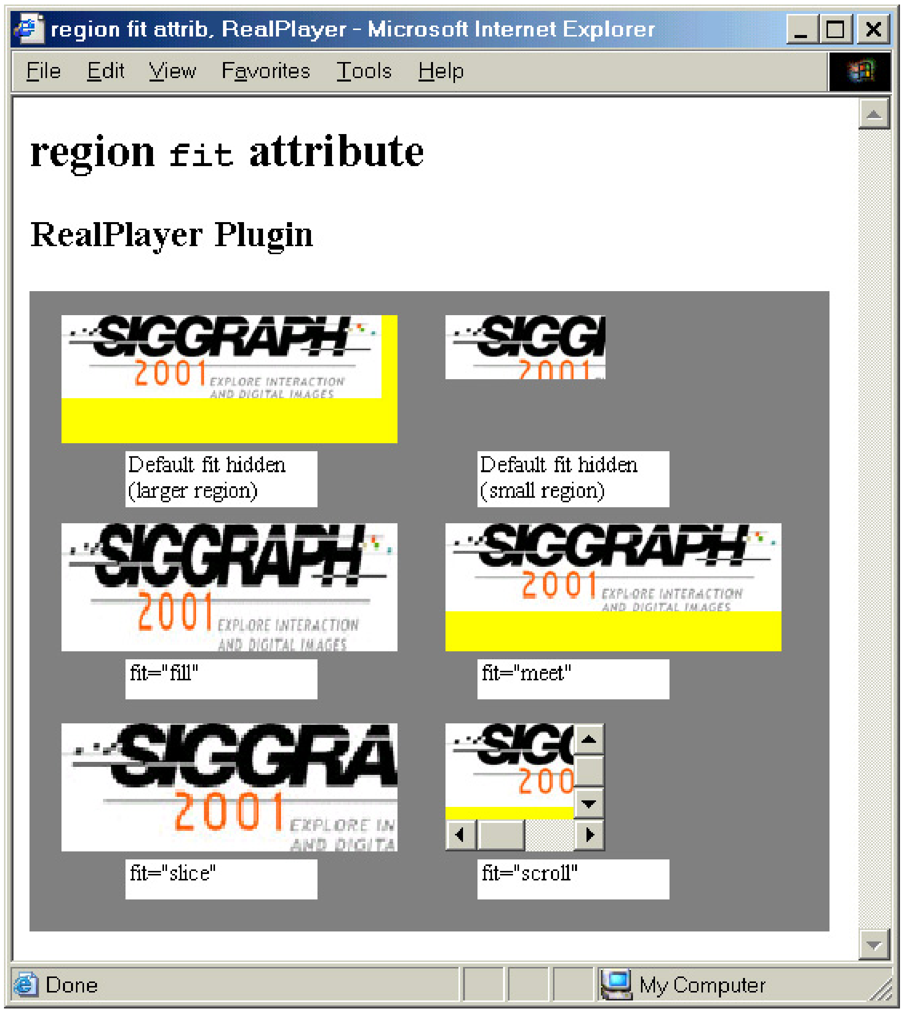
(4) `fit` attribute

The `fit` attribute determines if and how the media, such as an image, is stretched to fill its region. The default value is `hidden`.

`hidden`	image placed unstretched at the top left corner of the region. Any portion of the image outside of the region is clipped.
`fill`	stretches (disproportionately) the image to fit the region
`meet`	stretches the image proportionately until it meets one of the image boundaries
`slice`	stretches the image proportionally to fill the entire region. Any part of the image outside of the region is clipped.
`scroll`	scrollbars appear if the image is larger than its region.

<u>**`SMIL_Ex/ex4/fit_example.smil`** Screen Capture</u>

The regions have a yellow background so that you can tell their size and location.


region fit attrib, RealPlayer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

region `fit` attribute

RealPlayer Plugin

Default fit hidden (larger region)

Default fit hidden (small region)

fit="fill"

fit="meet"

fit="slice"

fit="scroll"

SMIL Ex/ex4/fit example.smil

```
<!--
region's fit attribute
-->
<smil>
  <head>
      <meta name="Author" content="Kathy Barshatzky" />
      <meta name="Copyright" content="javakathy.com" />
      <layout>
         <root-layout width="500" height="400"
                      background-color="gray"/>

         <region id="reg1"
            width="210" height="80" top="15" left="20"
            background-color="yellow" />
         <region id="reg1_txt" background-color="white"
            width="120" height="35" top="100" left="60" />

         <region id="reg2"
            width="100" height="40" top="15" left="260"
            background-color="yellow" />
         <region id="reg2_txt" background-color="white"
            width="120" height="35" top="100" left="280" />

         <region id="reg3" fit="fill"
            width="210" height="80" top="145" left="20"
            background-color="yellow" />
         <region id="reg3_txt" background-color="white"
            width="120" height="25" top="230" left="60" />

         <region id="reg4" fit="meet"
            width="210" height="80" top="145" left="260"
            background-color="yellow" />
         <region id="reg4_txt" background-color="white"
            width="120" height="25" top="230" left="280" />

         <region id="reg5" fit="slice"
            width="210" height="80" top="270" left="20"
            background-color="yellow" />
         <region id="reg5_txt" background-color="white"
            width="120" height="25" top="355" left="60" />

         <region id="reg6" fit="scroll"
            width="100" height="80" top="270" left="260"
            background-color="yellow" />
         <region id="reg6_txt" background-color="white"
            width="120" height="25" top="355" left="280" />

      </layout>
  </head>
  <body>
      <par>
         <img src="sigLogo2001.jpg" alt="siggraph 2001 logo"
             region="reg1" dur="indefinite" />
         <text src="fit_default1.txt" alt="fit default"
             region="reg1_txt" dur="indefinite" />

         <img src="sigLogo2001.jpg" alt=" siggraph 2001 logo"
             region="reg2" dur="indefinite" />
         <text src="fit_default2.txt" alt="fit default"
             region="reg2_txt" dur="indefinite" />

         <img src="sigLogo2001.jpg" alt=" siggraph 2001 logo"
             region="reg3" dur="indefinite" />
         <text src="fit_fill.txt" alt="fit fill"
             region="reg3_txt" dur="indefinite" />

         <img src="sigLogo2001.jpg" alt=" siggraph 2001 logo"
             region="reg4" dur="indefinite" />
         <text src="fit_meet.txt" alt="fit meet"
             region="reg4_txt" dur="indefinite" />

         <img src="sigLogo2001.jpg" alt=" siggraph 2001 logo"
             region="reg5" dur="indefinite" />
         <text src="fit_slice.txt" alt="fit slice"
             region="reg5_txt" dur="indefinite" />

         <img src="sigLogo2001.jpg" alt=" siggraph 2001 logo"
             region="reg6" dur="indefinite" />
         <text src="fit_scroll.txt" alt="fit scroll"
             region="reg6_txt" dur="indefinite" />
      </par>
  </body>
</smil>
```

B. Layout types

a) default

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

http://www.w3.org/TR/REC-smil/#layout-elements

SMIL basic layout uses the formatting properties defined by CSS2 (Cascading Style Sheets Level 2) to control the layout of media object elements.

b) CSS2

CSS2 can be used as an alternative to the basic layout.

```
<layout type="text/css">
    [region="r"] { top: 20px; left: 20px }
</layout>
```

http://www.w3.org/TR/REC-smil/#layout

4.2 <meta>

- Attributes

  1. **`name`** is the *identifier* for the information specified by `content`

  2. **`content`** provides detailed information about the topic specified by name.

  Examples:

```
<meta name="Author" content="Kathy Barshatzky" />
<meta name="Copyright" content="javakathy.com" />
```

§5. <body>

5.1 Media Object Elements

`animation`	time-based function of a target element
`audio`	audio clip, such as wav, mp3
`img`	still image, such as gif, jpg
`ref`	generic media reference
`text`	unformatted or html text
`textstream`	streaming text
`video`	video clip, such as Real movie, avi, mpg, QuickTime

Not all players support all possible media types. A complete list of MIME types is in the appendix for reference.

continuous media	media objects with an intrinsic duration, such as video
discrete media	media objects without an intrinsic duration, such as text

5.2 Text Media Types

<text>

type = "text/plain"
Plain, unformatted text in a `.txt` file.

type = "text/html"
Text in html format in a `.html` file.
(Not currently supported by RealPlayer or QuickTime Player.)

Example: Contents of the file `caption3.html`:

```
<html>
   <head>
      <title>Caption 3</title>
   </head>

   <body bgcolor="red" >
      <p>
      <b>Go Big Red!</b> 12/28/2000  Teco Arena, Estero, Florida
      </p>
   </body>
</html>
```

<textstream>

`textstream` supports streaming text, such as RealText.
See the References section for links to more information.

5.3 Synchronization

- Two possible synchronizations:

`<par>`	parallel	Media executed at the same time
`<seq>`	sequential	Media executed one after the other

These can be nested. For example, you can have two sequences running in parallel:

```
<par>
        <seq>
        ...
        </seq>

        <seq>
        ...
        </seq>
</par>
```

- Attributes

1. **`dur`** duration Length of time that the media is visible and playing.

```
<img dur ="indefinite" ... />
<text dur ="6s" ... />
```

Discrete media objects, such as text and images, should be given an appropriate duration, or they may vanish quickly from the window. The previous examples all had `dur = "indefinite"` so that the photo and caption would remain present.

2. **`begin`** delay Length of time that the media waits before playing.

```
<text begin="3s" ... />
```

3. **`end`** delay Time after which the media stops playing.

```
<text end="5s" ... />
```

SMIL Ex/ex6/tour.smil

This presentation plays a movie with subtitles. There are three elements playing in parallel: the video, the title, and the subtitles. The subtitles are a sequence of text files, each with the given duration.

```
<smil>
   <head>
      <meta name="Author" content="Kathy Barshatzky" />
      <meta name="Copyright" content="javakathy.com" />
      <layout>
         <root-layout width="380" height="340" />
         <region id="title"
                  width="300" height="50" left="50" top="12" />
         <region id="movie"
                  width="340" height="242" left="20" top="50"  />
         <region id="caption"
                  width="300" height="40" left="40" top="310" />
      </layout>
   </head>

   <body>
      <par>
         <text src="title.txt" region="title" dur="indefinite" />
         <video src="CircleVisionTour.avi" region="movie"
                alt="Circle Vision Tour" />
         <seq>
            <text region="caption"
                  src="caption1.txt" dur="7s"/>
            <text region="caption"
                  src="caption2.txt" dur="8s"/>
            <text region="caption"
                  src="caption3.txt" dur="14s" />
            <text region="caption"
                  src="caption4.txt" dur="8s" />
            <text region="caption"
                  src="cap5nook.txt" dur="8s" />
            <text region="caption"
                  src="cap6kitchen.txt" dur="6s" />
             <text region="caption"
                  src="cap7diningRm.txt" dur="6s" />
             <text region="caption"
                  src="cap8livingRm.txt" dur="6s" />
         </seq>
      </par>
   </body>

</smil>
```

5.4 Adapting the Presentation

`<switch>`

Adapting a presentation to the end-user's system based on

```
system-bitrate
system-captions
system-language
system-overdub-or-caption
system-required
system-screen-size
system-screen-depth
```

A set of child test attributes are placed within the **`<switch>`** tags. The first match is executed. A match is the first child whose test attributes all evaluate to TRUE.

A complete description of the test attributes can be found at:
http://www.w3.org/TR/REC-smil/#test

For example, your presentation could play the audio track in different languages based on the user's preferred language:

```
...
<switch>
   <audio src="salesPitch-french" system-language="fr"/>
   <audio src=" salesPitch -english" system-language="en"/>
</switch>
```

If the user's preferred language is French, the French audio will play. If the user's preferred language is English, the English audio will play.

SMIL Ex/ex7/ex7_switch.smil

```
<!-- Select the resolution of the image to show based on the resolution
of the end-viewer's system.
-->
<smil>
  <head>
      <meta name="Author" content="Kathy Barshatzky" />
      <meta name="Copyright" content="javakathy.com" />
      <layout>
         <root-layout width="1050" height="820" />
         <region id="photo_region"
                  width="1024" height="768" top="60" left="38" />
         <region id="caption_region"
                    width="340" height="40" top="10" left="60" />
      </layout>
  </head>
  <body>
      <par>
         <text src="roof.txt"
              region="caption_region" type="text/plain"
              dur="indefinite"/>
         <switch>
             <img system-screen-size="1280X1024" src="rt149_1024.jpg"
                   alt="roof truss" region="photo_region"
                   dur="indefinite" />
             <img system-screen-size="1024X768"  src="rt149_640.jpg"
                   alt="roof truss" region="photo_region"
                   dur="indefinite" />
             <img system-screen-size="640X480"  src="rt149_160.jpg"
                   alt="roof truss" region="photo_region"
                   dur="indefinite" />
         </switch>

      </par>
  </body>
</smil>
```

5.5 Hyperlinking

A *link* has two ends, called *anchors*. The link starts at the source anchor and points to the destination anchor.

<a href=”*xx*”>	sets up an anchor to a *complete* media object
<anchor>	sets up an anchor to a *portion of a* media object spatial or temporal subparts

Ex: Linking two text buttons to two other presentations:

```
<a href="presentation2.smil">
    <text region="r_btn2" src="button2.txt" dur="indefinite" />
</a>

<a href="presentation3.smil">
    <text region="r_btn3" src="button3.txt" dur="indefinite" />
</a>
```

REC-smil-19980615

Synchronized Multimedia Integration Language (SMIL) 1.0 Specification

W3C Recommendation 15-June-1998

This version:
http://www.w3.org/TR/1998/REC-smil-19980615
Latest version:
http://www.w3.org/TR/REC-smil
Previous version:
http://www.w3.org/TR/1998/PR-smil-19980409

About this Document

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

- Stephan Bugaj, Lucent/Bell Labs
- Dick Bulterman, CWI
- Bruce Butterfield, RealNetworks
- Wo Chang, NIST
- Guy Fouquet, Alcatel
- Christian Gran, GMD
- Mark Hakkinen, The Productivity Works
- Lynda Hardman, CWI
- Peter Hoddie, Apple
- Klaus Hofrichter, GMD
- Philipp Hoschka, W3C
- Jack Jansen, CWI
- George Kerscher, DAISY Consortium
- Rob Lanphier, RealNetworks
- Nabil Layaïda, INRIA
- Stephanie Leif, RealNetworks
- Sjoerd Mullender, CWI
- Didier Pillet, CNET/DSM
- Anup Rao, Netscape
- Lloyd Rutledge, CWI
- Patrick Soquet, Havas
- Warner ten Kate, Philips
- Jacco van Ossenbruggen, CWI
- Michael Vernick, Lucent/Bell Labs
- Jin Yu, DEC

Acknowledgments: In addition to the working group members, the following people contributed to the SMIL effort: Bert Bos (W3C), Dan Connolly (W3C), Patrick Deunhouwer (Philips), Martin Dürst (W3C), Al Gilman, Håkon Lie (W3C), Chris Lilley (W3C), Curtis Reynolds (RealNetworks), Michael Riesman, Curtis Reynolds (RealNetworks), Henning Schulzrinne (Columbia University) and Koga Youichirou (W3C).

Editor: Philipp Hoschka, W3C (hoschka@w3.org)

Abstract

This document specifies version 1 of the Synchronized Multimedia Integration Language (SMIL 1.0, pronounced "smile"). SMIL allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can

1. describe the temporal behavior of the presentation
2. describe the layout of the presentation on a screen
3. associate hyperlinks with media objects

This specification is structured as follows: Section 1 presents the specification approach. Section 2 defines the "smil" element. Section 3 defines the elements that can be contained in the head part of a SMIL document. Section 4 defines the elements that can be contained in the body part of a SMIL document. In particular, this Section defines the time model used in SMIL. Section 5 describes the SMIL DTD.

Status of this Document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Comments on this Recommendation may be sent to the public mailing list www-smil@w3.org.

Available languages

The English version of this specification is the only normative version. However, for translations in other languages see http://www.w3.org/AudioVideo/SMIL/translations.

Errata

The list of known errors in this specification is available at http://www.w3.org/AudioVideo/SMIL/errata.

Table of Contents

1 Specification Approach

SMIL documents are XML 1.0 documents [XML10]. The reader is expected to be familiar with the concepts and terms defined in XML 1.0.

This specification does not rely on particular features defined in URLs that cannot potentially be expressed using URNs. Therefore, the more generic term URI [URI] is used throughout the specification.

The syntax of SMIL documents is defined by the DTD in Section 5.2. The syntax of an attribute value that cannot be defined using the DTD notation is defined together with the first element using an attribute that can contain the attribute value. The syntax of such attribute values is defined using the Extended Backus-Naur Form (EBNF) defined in the XML 1.0 specification.

An element definition is structured as follows: First, all attributes of the element are defined in alphabetical order. An attribute is defined in the following way: If the attribute is used by an element for the first time in the specification, the semantics of the attribute are defined. If the attribute has already been used by another element, the specification refers to the definition of the attribute in the first element that used it. The definition of element attributes is followed by the definition of any attribute values whose syntax cannot be defined using the DTD notation. The final section in an element definition specifies the element content.

2 The `smil` Element

Element Attributes

The **`smil`** element can have the following attribute:

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

Element Content

The **`smil`** element can contain the following children:

`body`	Defined in Section 4.1
`head`	Defined in Section 3.1

3 The Document Head

3.1 The `head` Element

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

Element Attributes

The `head` element can have the following attribute:

`id`	Defined in Section 2

Element Content

The `head` element can contain the following children:

`layout`	Defined in Section 3.2
`meta`	Defined in Section 3.4
`switch`	Defined in Section 4.3

The `head` element may contain any number of `meta` elements and either a `layout` element or a `switch` element.

3.2 The `layout` Element

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

If a document contains no **`layout`** element, the positioning of the body elements is implementation-dependent.

A SMIL document can contain multiple alternative layouts by enclosing several layout elements within a **`switch`** element (defined in Section 4.3). This can be used for example to describe the document's layout using different layout languages.

The following example shows how CSS2 can be used as alternative to the SMIL basic layout language (defined in Section 3.3):

```
<smil>
   <head>
      <switch>
         <layout type="text/css">
            [region="r"] { top: 20px; left: 20px }
         </layout>
         <layout>
            <region id="r" top="20" left="20" />
         </layout>
      </switch>
   </head>
   <body>
      <seq>
         <img region="r" src="http://www.w3.org/test" dur="10s" />
      </seq>
   </body>
</smil>
```

(note that in this example, both layout alternatives result in the same layout)

Element Attributes

`id`	Defined in Section 2
`type`	This attribute specifies which layout language is used in the layout element. If the player does not understand this language, it must skip all content up until the next "`</layout>`" tag. The default value of the type attribute is "**`text/smil-basic-layout`**".

Element Content

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

`region`	Defined in Section 3.3.1
`root-layout`	Defined in Section 3.3.2

If the **`type`** attribute of the **`layout`** element has another value, the element contains character data.

3.3 SMIL Basic Layout Language

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

SMIL basic layout only controls the layout of media object elements (defined in Section 4.2.3). It is illegal to use SMIL basic layout for other SMIL elements.

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

Fixed Property Values

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

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

Note that as a result of these definitions, all absolutely positioned elements (`animation`, `img`, `ref`, `text`, `textstream` and `video`) are contained within a single containing block defined by the content content edge of the root element (`smil`).

Default Values

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

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

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

3.3.1 The `region` Element

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

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

```
<smil>
   <head>
      <layout>
         <region id="a" top="5" />
      </layout>
   </head>
   <body>
         <text region="a" src="text.html" dur="10s" />
   </body>
</smil>
```

Element Attributes

The **`region`** element can have the following attributes:

`background-color`

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

`fit`

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

`fill`

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

`hidden`

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

`meet`

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

`scroll`

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

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

The default value of **`fill`** is **`hidden`**.

`height`
The use and definition of this attribute are identical to the "height" property in the CSS2 specification. Attribute values can be "percentage" values, and a variation of the "length" values defined in CSS2. For "length" values, SMIL basic layout only supports pixel units as defined in CSS2. It allows to leave out the "px" unit qualifier in pixel values (the "px" qualifier is required in CSS2).

`id`
Defined in Section 2

A region element is applied to a positionable element by setting the **`region`** attribute of the positionable element to the **`id`** value of the region.

The **`id`** attribute is required for **`region`** elements.

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

The default value is zero.

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

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

If the value of the **`skip-content`** attribute is "**`true`**", and one of the cases above apply, the content of the element is ignored. If the value is "**`false`**", the content of the element is processed.

The default value for **`skip-content`** is "**`true`**".

`title`
This attribute offers advisory information about the element for which it is set. Values of the title attribute may be rendered by user agents in a variety of ways. For instance, visual browsers frequently display the title as a "tool tip" (a short message that appears when the pointing device pauses over an object).

It is strongly recommended that all **`region`** elements have a **`title`** attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

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

  The default value is zero.

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

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

- If two boxes generated by elements A and B have the same stack level, then
    1. If the display of an element A starts later than the display of an element B, the box of A is stacked on top of the box of B (temporal order).
    2. If the display of the elements starts at the same time, and an element A occurs later in the SMIL document text than an element B, the box of A is stacked on top of the box of B (document tree order as defined in CSS2).

Element Content

region is an empty element.

3.3.2 The `root-layout` Element

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

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

Element Attributes

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

`background-color`	Defined in Section 3.3.1
`height`	Defined in Section 3.3.1 Sets the height of the root element. Only length values are allowed.
`id`	Defined in Section 2
`skip-content`	Defined in Section 3.3.1
`title`	Defined in Section 3.3.1
`width`	Defined in Section 3.3.1 Sets the width of the root element. Only length values are allowed.

Element Content

`root-layout` is an empty element.

3.4 The `meta` Element

The **`meta`** element can be used to define properties of a document (e.g., author, expiration date, a list of key words, etc.) and assign values to those properties. Each **`meta`** element specifies a single property/value pair.

Element Attributes

The **`meta`** element can have the following attributes:

`content`	This attribute specifies the value of the property defined in the **`meta`** element. The **`content`** attribute is required for **`meta`** elements.
`id`	Defined in Section 2
`name`	This attribute identifies the property defined in the **`meta`** element. The **`name`** attribute is required for **`meta`** elements.
`skip-content`	Defined in Section 3.3.1

The list of properties is open-ended. This specification defines the following properties:

`base`	The value of this property determines the base URI for all relative URIs used in the document.
`pics-label` **`PICS-Label`**	The value of this property specifies a valid rating label for the document as defined by PICS [PICS].
`title`	The value of this property contains the title of the presentation.

Element Content

`meta` is an empty element.

4 The Document Body

4.1 The **body** Element

The **body** element contains information that is related to the temporal and linking behavior of the document. It implicitly defines a **seq** element (defined in Section 4.2.2, see Section 4.2.4 for a definition of the temporal semantics of the **body** element).

Element Attributes

The **body** element can have the following attribute:

id	Defined in Section 2

Element Content

The **body** element can contain the following children:

a	Defined in Section 4.5.1
animation	Defined in Section 4.2.3
audio	Defined in Section 4.2.3
img	Defined in Section 4.2.3
par	Defined in Section 4.2.1
ref	Defined in Section 4.2.3
seq	Defined in Section 4.2.2
switch	Defined in Section 4.3
text	Defined in Section 4.2.3
textstream	Defined in Section 4.2.3
video	Defined in Section 4.2.3

4.2 Synchronization Elements

4.2.1 The **par** Element

The children of a **par** element can overlap in time. The textual order of appearance of children in a par has no significance for the timing of their presentation.

Element Attributes

The **par** element can have the following attributes: **abstract**, **author**, **begin**, **copyright**, **dur**, **end**, **endsync**, **id**, **region**, **system-bitrate**, **system-captions**, **system-language**, **system-overdub-or-caption**, **system-required**, **system-screen-size**, **system-screen-depth**, **title**:

abstract	A brief description of the content contained in the element.
author	The name of the author of the content contained in the element.
begin	This attribute specifies the time for the explicit begin of an element. See Section 4.2.4 for a definition of its semantics.

The attribute can contain the following two types of values:

delay-value

A delay value is a *clock-value* measuring presentation time. Presentation time advances at the speed of the presentation. It behaves like the timecode shown on a counter of a tape-deck. It can be stopped, decreased or increased either by user actions, or by the player itself.

The semantics of a delay value depend on the element's first ancestor that is a synchronization element (i.e. ancestors that are "**a**" or "**switch**" elements are ignored):

- If this ancestor is a **par** element, the value defines a delay from the effective begin of that element (see Figure 4.1).
- If this ancestor is a **seq** element (defined in Section 4.2.2), the value defines a delay from the effective end of the first lexical predecessor that is a synchronization element (see Figure 4.2).

event-value

The element begins when a certain event occurs (see Figure 4.3). Its value is an *element-event* (see Definition below).

The element generating the event must be "in scope". The set of "in scope" elements S is determined as follows:

1. Take all children from the element's first ancestor that is a synchronization element and add them to S.
2. Remove all "a" and "switch" elements from S. Add the children of all "a" elements to S, unless they are "switch" elements.

The resulting set S is the set of "in scope" elements.

```
<par>
  <audio id="a" begin="6s" src="audio" />
</par>
```

```
      par
|-----------------|

   6s      a
<----->|-----------|
```

Figure 4.1: Using a delay value within a "`par`" element

```
<seq>
  <audio src="audio1" />
  <audio begin="5s" src="audio2" />
</seq>
```

```
   audio     5s     audio
|---------|<---->|---------|
```

Figure 4.2: Using a delay value within a "`seq`" element

```
<par>
  <audio id="a" begin="6s" ... />
  <img  begin="id(a)(4s)" ... />
</par>
```

```
      par
|-----------------|

   6s      a
<---->|-----------|
       4s
      <-->
          img
        |-------|
```

Figure 4.3: Synchronization attribute with element event value

---

`copyright`	The copyright notice of the content contained in the element.
`dur`	This attribute specifies the explicit duration of an element. See Section 4.2.4 for a definition of its semantics. The attribute value can be a clock value, or the string **`indefinite`**.
`end`	This attribute specifies the explicit end of an element. See Section 4.2.4 for a definition of its semantics. The attribute can contain the same types of attribute values as the **`begin`** attribute.

endsync For a definition of the semantics of this attribute, see Section 4.2.4. The attribute can have the following values:

- **first** For a definition of the semantics of this value, see Section 4.2.4.
- **id-ref** This attribute value has the following syntax:

  ```
  id-ref ::= "id(" id-value ")"
  ```

  where "id-value" must be a legal XML identifier.
  For a definition of the semantics of this value, see Section 4.2.4.
- **last** For a definition of the semantics of this value, see Section 4.2.4.

The default value of **endsync** is **last**.

id Defined in Section 2

region This attribute specifies an abstract rendering surface (either visual or acoustic) defined within the layout section of the document. Its value must be an XML identifier. If no rendering surface with this id is defined in the layout section, the values of the formatting properties of this element are determined by the default layout.

The **region** attribute on **par** elements cannot be used by the basic layout language for SMIL defined in this specification. It is added for completeness, since it may be required by other layout languages.

repeat For a definition of the semantics of this attribute, see Section 4.2.4. The attribute value can be an integer, or the string **indefinite**. The default value is **1**.

system-bitrate	Defined in Section 4.4
system-captions	Defined in Section 4.4
system-language	Defined in Section 4.4
system-overdub-or-caption	Defined in Section 4.4
system-required	Defined in Section 4.4
system-screen-size	Defined in Section 4.4
system-screen-depth	Defined in Section 4.4

title Defined in Section 3.3.1

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

Note on Synchronization between Children

The accuracy of synchronization between the children in a parallel group is implementation-dependent. Take the example of synchronization in case of playback delays, i.e. the behavior when the **par** element contains two or more continuous media types such as audio or video, and one of them experiences a delay. A player can show the following synchronization behaviors:

hard synchronization
The player synchronizes the children in the **par** element to a common clock (see Figure 4.4 a)).

soft synchronization
Each child of the **par** element has its own clock, which runs independently of the clocks of other children in the **par** element (see Figure 4.4 b)).

```
   audio
|----...------|
   video
|----...------|
```

```
   audio
|----------|
   video
|----...--|
```

a) hard synchronization: Delay in video: Either the audio is stopped, or some video frames are dropped. The exact behavior is implementation-dependent

```
   audio
|----------|
   video
|----...------|
```

b) soft synchronization

Figure 4.4: Effect of a delay on playout schedule for players using different synchronization policies

---

Attribute Values

clock-value

Clock values have the following syntax:

```
Clock-val           ::= Full-clock-val | Partial-clock-val |
                         Timecount-val
Full-clock-val      ::= Hours ":" Minutes ":" Seconds ("."
                         Fraction)?
Partial-clock-val   ::= Minutes ":" Seconds ("." Fraction)?
Timecount-val       ::= Timecount ("." Fraction)?
                         ("h" | "min" | "s" | "ms")? ; default is "s"
Hours               ::= 2DIGIT; 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]
```

The following are examples of legal clock values:

- Full clock value: 02:30:03 = 2 hours, 30 minutes and 3 seconds
- Partial clock value: 02:33 = 2 minutes and 33 seconds
- Timecount values:
  3h = 3 hours
  45min = 45 minutes
  30s = 30 seconds
  5ms = 5 milliseconds

A fraction x with n digits represents the following value: x * 1/10**n

Examples:

00.5s = 5 * 1/10 seconds = 500 milliseconds
00:00.005 = 5 * 1/1000 seconds = 5 milliseconds

element-event value

An *element event* value specifies a particular event in a synchronization element. An element event has the following syntax:

```
Element-event      ::= "id(" Event-source ")(" Event ")"
Event-source       ::= Id-value
Event              ::= "begin" | Clock-val | "end"
```

The following events are defined:

`begin`

This event is generated at an element's effective begin.
Example use: `begin="id(x)(begin)"`

clock-val

This event is generated when a clock associated with an element reaches a particular value. This clock starts at 0 at the element's effective begin. For `par` and `seq` elements, the clock gives the presentation time elapsed since the effective begin of the element. For media object elements, the semantics are implementation-dependent. The clock may either give presentation time elapsed since the effective begin, or it may give the media time of the object. The latter may differ from the presentation time that elapsed since the object's display was started e.g. due to rendering or network delays, and is the recommended approach.
It is an error to use a clock value that exceeds the value of the effective duration of the element generating the event.

Example use: `begin="id(x)(45s)"`

`end`

This event is generated at the element's effective end.
Example use: `begin="id(x)(end)"`

Element Content

The **`par`** element can contain the following children:

`a`	Defined in Section 4.5.1
`animation`	Defined in Section 4.2.3
`audio`	Defined in Section 4.2.3
`img`	Defined in Section 4.2.3
`par`	Defined in Section 4.2.1
`ref`	Defined in Section 4.2.3
`seq`	Defined in Section 4.2.2
`switch`	Defined in Section 4.3
`text`	Defined in Section 4.2.3
`textstream`	Defined in Section 4.2.3
`video`	Defined in Section 4.2.3

All of these elements may appear multiple times as direct children of a **`par`** element.

4.2.2 The **seq** Element

The children of a **seq** element form a temporal sequence.

Attributes

The **seq** element can have the following attributes:

abstract	Defined in Section 4.2.1
author	Defined in Section 4.2.1
begin	Defined in Section 4.2.1
copyright	Defined in Section 4.2.1
dur	Defined in Section 4.2.1
end	Defined in Section 4.2.1
id	Defined in Section 2
region	Defined in Section 4.2.1

The **region** attribute on **seq** elements cannot be used by the basic layout language for SMIL defined in this specification. It is added for completeness, since it may be required by other layout languages.

repeat	Defined in Section 4.2.1
system-bitrate	Defined in Section 4.4
system-captions	Defined in Section 4.4
system-language	Defined in Section 4.4
system-overdub-or-caption	Defined in Section 4.4
system-required	Defined in Section 4.4
system-screen-size	Defined in Section 4.4
system-screen-depth	Defined in Section 4.4
title	Defined in Section 3.3.1

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

Element Content

The **seq** element can contain the following children:

a	Defined in Section 4.5.1
animation	Defined in Section 4.2.3
audio	Defined in Section 4.2.3
img	Defined in Section 4.2.3
par	Defined in Section 4.2.1
ref	Defined in Section 4.2.3
seq	Defined in Section 4.2.2
switch	Defined in Section 4.3
text	Defined in Section 4.2.3
textstream	Defined in Section 4.2.3
video	Defined in Section 4.2.3

4.2.3 Media Object Elements:
The **ref**, **animation**, **audio**, **img**, **video**, **text** and **textstream** Elements

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

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

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

When playing back a media object, the player must not derive the exact type of the media object from the name of the media object element. Instead, it must rely solely on other sources about the type, such as type information contained in the "type" attribute, or the type information communicated by a server or the operating system.

Authors, however, should make sure that the group into which of the media object falls (animation, audio, img, video, text or textstream) is reflected in the element name. This is in order to increase the readability of the SMIL document. When in doubt about the group of a media object, authors should use the generic "ref" element.

Element Attributes

Media object elements can have the following attributes:

abstract	Defined in Section 4.2.1
alt	For user agents that cannot display a particular media-object, this attribute specifies alternate text. It is strongly recommended that all media object elements have an **alt** attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.
author	Defined in Section 4.2.1
begin	Defined in Section 4.2.1
clip-begin	The **clip-begin** attribute specifies the beginning of a sub-clip of a continuous media object as offset from the start of the media object. Values in the **clip-begin** attribute have the following syntax:

```
Clip-time-value   ::= Metric "=" ( Clock-val | Smpte-val )
Metric            ::= Smpte-type | "npt"
Smpte-type        ::= "smpte" | "smpte-30-drop" | "smpte-25"
Smpte-val         ::= Hours ":" Minutes ":" Seconds
                      [ ":" Frames [ "." Subframes ]]
Hours             ::= 2DIGIT
Minutes           ::= 2DIGIT
Seconds           ::= 2DIGIT
Frames            ::= 2DIGIT
Subframes         ::= 2DIGIT
```

The value of this attribute consists of a metric specifier, followed by a time value whose syntax and semantics depend on the metric specifier. The following formats are allowed:

SMPTE Timestamp

SMPTE time codes [SMPTE] can be used for frame-level access accuracy. The metric specifier can have the following values:

`smpte`
`smpte-30-drop`

These values indicate the use of the "SMPTE 30 drop" format with 29.97 frames per second. The "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute.

`smpte-25`

The "frames" field in the time specification can assume the values 0 through 24.

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

```
clip-begin="smpte=10:12:33:20"
```

Normal Play Time

Normal Play Time expresses time in terms of SMIL clock values. The metric specifier is "`npt`", and the syntax of the time value is identical to the syntax of SMIL clock values. Examples:

```
clip-begin="npt=123.45s"
clip-begin="npt=12:05:35.3"
```

`clip-end`	The **`clip-end`** attribute specifies the end of a sub-clip of a continuous media object (such as audio, video or another presentation) that should be played. It uses the same attribute value syntax as the clip-begin attribute. If the value of the **`clip-end`** attribute exceeds the duration of the media object, the value is ignored, and the clip end is set equal to the effective end of the media object.
`copyright`	Defined in Section 4.2.1
`dur`	Defined in Section 4.2.1
`end`	Defined in Section 4.2.1
`fill`	For a definition of the semantics of this attribute, see Section 4.2.4. The attribute can have the values **`remove`** and **`freeze`**.
`id`	Defined in Section 2
`longdesc`	This attribute specifies a link (URI) to a long description of a media object. This description should supplement the short description provided using the alt attribute. When the media-object has associated anchors, this attribute should provide information about the anchor's contents.
`region`	Defined in Section 4.2.1
`src`	The value of the src attribute is the URI of the media object.
`system-bitrate`	Defined in Section 4.4
`system-captions`	Defined in Section 4.4
`system-language`	Defined in Section 4.4

`system-overdub-or-caption`	Defined in Section 4.4
`system-required`	Defined in Section 4.4
`system-screen-size`	Defined in Section 4.4
`system-screen-depth`	Defined in Section 4.4
`title`	Defined in Section 3.3.1 It is strongly recommended that all media object elements have a `title` attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.
`type`	MIME type of the media object referenced by the `src` attribute.

Element Content

Media Object Elements can contain the following element:

`anchor`	Defined in Section 4.5.2

4.2.4 SMIL Time Model

4.2.4.1 Time Model Values

In the following discussion, the term "element" refers to synchronization elements only.

For each element we define the implicit, explicit, desired, and effective begin, duration, and end.

The effective begin/duration/end specify what the reader of the document will perceive.

The implicit, explicit, and desired values are auxiliary values used to define the effective values.

The rules for calculating each of these values for the elements defined in SMIL 1.0 are described in the next section.

1. Each element in SMIL has an *implicit begin*.
2. Each element can be assigned an *explicit begin* by adding a "begin" attribute to the element:

   ```
   begin = "value of explicit-begin"
   ```

   It is an error if the explicit begin is earlier than the implicit begin of the element.

3. Each element in SMIL has an *implicit end*.
4. Each element can be assigned an *explicit end* by adding an "end" attribute to the element:

   ```
   end = "value of explicit-end"
   ```

5. The *implicit duration* of an element is the difference between the implicit end and the implicit begin.
6. Each element in SMIL can be assigned an *explicit duration* by adding a "dur" attribute to the element:

   ```
   dur = "value of explicit-duration"
   ```

7. The *desired begin* of an element is equal to the explicit begin if one is given, otherwise the desired begin is equal to the implicit begin.
8. Each element has a *desired end*.
9. The *desired duration* of an element is the difference between the desired end and the desired begin.
10. Each element has an *effective begin*.
11. Each element has an *effective end*. (Note: the effective end of a child element can never be later than the effective end of its parent.)
12. The *effective duration* of an element is the difference between the effective end and the effective begin.

4.2.4.2 Determining Time Model Values for SMIL 1.0 Elements

This section defines how time model values are calculated for the synchronization elements of SMIL 1.0 in cases that are not covered by the rules in Section 4.2.4.1.

Determining the *implicit begin* of an element

- The implicit begin of the first child of the "body" element is when the document starts playing. When this is falls outside the scope of this document.
- The implicit begin of a child of a "par" element is equal to the effective begin of the "par" element.
- The implicit begin of the first child of a "seq" element is equal to the effective begin of the "seq" element.
- The implicit begin of any other child of a "seq" element is equal to the desired end time of the previous child of the "seq" element.

Determining the *implicit end* of an element

The first description that matches the element is the one that is to be used:

- An element with a "repeat" attribute with value "indefinite" has an implicit end immediately after its effective begin.
- An element with a "repeat" attribute with a value other than "indefinite" has an implicit end equal to the implicit end of a seq element with the stated number of copies of the element without "repeat" attribute as children.
- A media object element referring to a continuous media object has an implicit end equal to the sum of the effective begin of the element and the intrinsic duration of the media object.
- A media object element referring to a discrete media object such as text or image has an implicit end immediately after its effective begin.
- A "seq" element has an implicit end equal to the desired end of its last child.
- A "par" element has an implicit end that depends on the value of the "endsync" attribute. The implicit end is equal to the sum of the effective begin of the "par" element and the implicit duration which is derived as follows:
    - If the value of the "endsync" attribute is "last", or if the "endsync" attribute is missing, the implicit duration of the "par" element is the maximum of the desired durations of its children.
    - If the value of the "endsync" attribute is "first", the implicit duration of the "par" element is the minimum of the desired durations of its children.
    - If the value of the "endsync" attribute is an id-ref, the implicit duration of the "par" element is equal to the desired duration of the child referenced by the "id-ref".

Determining the *desired end* of an element

- If the element has both an explicit duration and an explicit end, the desired end is the minimum of:
    - the sum of the desired begin and the explicit duration; and
    - the explicit end.
- If the element has an explicit duration but no explicit end, the desired end is the sum of the desired begin and the explicit duration.
- If the element has an explicit end but no explicit duration, the desired end is equal to the explicit end
- Otherwise, the desired end is equal to the implicit end.

Determining the *desired begin* of an element

The desired begin of an element is determined by using rule 7 in Section 4.2.4.1.

Determining the *effective begin* of an element

The *effective begin* of an element is equal to the desired begin of the element, unless the effective end of the parent element is earlier than this time, in which case the element is not shown at all.

Determining the *effective end* of an element

- The effective end of the last child of the body element is player-dependent. The effective end is at least as late as the desired end, but whether it is any later is implementation-dependent.
- The effective end of the child of a `par` element can be derived as follows:
    - If the child has a `fill` attribute, and the value of the `fill` attribute is "freeze", the effective end of the child element is equal to the effective end of the parent.
      The last state of the element is retained on the screen until the effective end of the element.
    - If the child has a `fill` attribute, and the value of the `fill` attribute is "remove", the effective end of the child element is the minimum of the effective end of the parent and the desired end of the child element.
    - If the child element has no `fill` attribute, the effective end of the child depends on whether or not the child has an explicit duration or end.
        - If the child has an explicit duration or end, the effective end is determined as if the element had a `fill` attribute with value "remove".
        - If the child has neither an explicit duration nor an explicit end, the effective end is determined as if the element had a `fill` attribute with value "freeze".
- The effective end of the last child of a `seq` element is derived in the same way as the effective end of a child of a `par` element.
- The effective end of any other child of a `seq` element can be derived as follows:
    - If the child has a `fill` attribute, and the value of the `fill` attribute is "freeze", the effective end of the child element is equal to the effective begin of the next element
    - If the child has a `fill` attribute, and the value of the `fill` attribute is "remove", the effective end of the child element is the minimum of the effective begin of the next element and the desired end of the next child element.
    - If the child element has no `fill` attribute, the effective end of the child depends on whether or not the child has an explicit duration or end.
        - If the child has an explicit duration or end, the effective end is determined as if the element had a fill attribute with value "remove".
        - If the child has neither an explicit duration nor an explicit end, the effective end is determined as if the element had a fill attribute with value "freeze".

4.3 The **switch** Element

The **switch** element allows an author to specify a set of alternative elements from which only one acceptable element should be chosen. An element is acceptable if the element is a SMIL 1.0 element, the media-type can be decoded, and all of the test-attributes (see Section 4.4) of the element evaluate to `true`.

An element is selected as follows: the player evaluates the elements in the order in which they occur in the switch element. The first acceptable element is selected at the exclusion of all other elements within the switch.

Thus, authors should order the alternatives from the most desirable to the least desirable. Furthermore, authors should place a relatively fail-safe alternative as the last item in the <switch> so that at least one item within the switch is chosen (unless this is explicitly not desired). Implementations should NOT arbitrarily pick an object within a **<switch>** when test-attributes for all fail.

Note that http URIs provide for content-negotiation, which may be an alternative to using the "switch" element in some cases.

Attributes

The switch element can have the following attributes:

id	Defined in Section 2
title	Defined in Section 3.3.1 It is strongly recommended that all **switch** elements have a **title** attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Element Content

If the **switch** element is used as a direct or indirect child of a **body** element, it can contain the following children:

a	Defined in Section 4.5.1
animation	Defined in Section 4.2.3
audio	Defined in Section 4.2.3
img	Defined in Section 4.2.3
par	Defined in Section 4.2.1
ref	Defined in Section 4.2.3
seq	Defined in Section 4.2.2
switch	Defined in Section 4.3
text	Defined in Section 4.2.3
textstream	Defined in Section 4.2.3
video	Defined in Section 4.2.3

All of these elements may appear multiple times as children of a **switch** element.

If the **switch** element is used within a **head** element, it can contain the following child:

layout	Defined in Section 3.2

Multiple **layout** elements may occur within the **switch** element.

4.4 Test Attributes

This specification defines a list of test attributes that can be added to any synchronization element, and that test system capabilities and settings. Conceptually, these attributes represent boolean tests. When one of the test attributes specified for an element evaluates to "false", the element carrying this attribute is ignored.

Within the list below, the concept of "user preference" may show up. User preferences are usually set by the playback engine using a preferences dialog box, but this specification does not place any restrictions on how such preferences are communicated from the user to the SMIL player.

The following test attributes are defined in SMIL 1.0:

`system-bitrate`

This attribute specifies the approximate bandwidth, in bits per second available to the system. The measurement of bandwidth is application specific, meaning that applications may use sophisticated measurement of end-to-end connectivity, or a simple static setting controlled by the user. In the latter case, this could for instance be used to make a choice based on the users connection to the network. Typical values for modem users would be 14400, 28800, 56000 bit/s etc. Evaluates to "true" if the available system bitrate is equal to or greater than the given value. Evaluates to "false" if the available system bitrate is less than the given value.
The attribute can assume any integer value greater than 0. If the value exceeds an implementation-defined maximum bandwidth value, the attribute always evaluates to "false".

`system-captions`

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

`system-language`

The attribute value is a comma-separated list of language names as defined in [RFC1766].

Evaluates to "`true`" if one of the languages indicated by user preferences exactly equals one of the languages given in the value of this parameter, or if one of the languages indicated by

user preferences exactly equals a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-".

Evaluates to "`false`" otherwise.

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

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

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

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

```
<audio src="foo.rm" system-language="mi, en"/>
```

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

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

`system-overdub-or-caption`
This attribute is a setting which determines if users prefer overdubbing or captioning when the option is available. The attribute can have the values "caption" and "overdub". Evaluates to "true" if the user preference matches this attribute value. Evaluates to "false" if they do not match.

`system-required`
This attribute specifies the name of an extension. Evaluates to "true" if the extension is supported by the implementation, otherwise, this evaluates to "false". In a future version of SMIL, this attribute value will be an XML namespace [NAMESPACES].

`system-screen-size`
Attribute values have the following syntax:
`screen-size-val ::= screen-height"X"screen-width`
Each of these is a pixel value, and must be an integer value greater than 0. Evaluates to "true" if the SMIL playback engine is capable of displaying a presentation of the given size. Evaluates to "false" if the SMIL playback engine is only capable of displaying a smaller presentation.

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

Examples

1) Choosing between content with different bitrate

In a common scenario, implementations may wish to allow for selection via a **`system-bitrate`** parameter on elements. The media player evaluates each of the "choices" (elements within the switch) one at a time, looking for an acceptable bitrate given the known characteristics of the link between the media player and media server.

```
...
<par>
  <text .../>
  <switch>
    <par system-bitrate="40000">
    ...
    </par>
    <par system-bitrate="24000">
    ...
    </par>
    <par system-bitrate="10000">
    ........
    </par>
  </switch>
</par>
...
```

2) Choosing between audio resources with different bitrate

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

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

3) Choosing between audio resources in different languages

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

```
...
<switch>
   <audio src="joe-audio-french" system-language="fr"/>
   <audio src="joe-audio-english" system-language="en"/>
</switch>
...
```

4) Choosing between content written for different screens

In the following example, the presentation contains alternative parts designed for screens with different resolutions and bit-depths. Depending on the particular characteristics of the screen, the player can choose one of the alternatives.

```
...
<par>
  <text .../>
  <switch>
    <par system-screen-size="1280X1024" system-screen-depth="16">
    ........
    </par>
    <par system-screen-size="640X480" system-screen-depth="32">
    ...
    </par>
    <par system-screen-size="640X480" system-screen-depth="16">
    ...
    </par>
  </switch>
</par>
...
```

5) Distinguishing caption tracks from stock tickers

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

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

6) Choosing the language of overdub and caption tracks

In the following example, a French-language movie is available with English, German, and Dutch overdub and caption tracks. The following SMIL segment expresses this, and switches on the alternatives that the user prefers.

```
...
<par>
  <switch>
    <audio src="movie-aud-en.rm" system-language="en"
                system-overdub-or-caption="overdub"/>
    <audio src="movie-aud-de.rm" system-language="de"
                system-overdub-or-caption="overdub"/>
    <audio src="movie-aud-nl.rm" system-language="nl"
                system-overdub-or-caption="overdub"/>
     <!-- French for everyone else -->
     <audio src="movie-aud-fr.rm"/>
   </switch>
   <video src="movie-vid.rm"/>
   <switch>
     <textstream src="movie-caps-en.rtx" system-language="en"
                 system-overdub-or-caption="caption"/>
     <textstream src="movie-caps-de.rtx" system-language="de"
                system-overdub-or-caption="caption"/>
     <textstream src="movie-caps-nl.rtx" system-language="nl"
                 system-overdub-or-caption="caption"/>
     <!-- French captions for those that really want them -->
     <textstream src="movie-caps-fr.rtx" system-captions="on"/>
   </switch>
</par>
...
```

4.5 Hyperlinking Elements

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

SMIL provides only for in-line link elements. Links are limited to uni-directional single-headed links (i.e. all links have exactly one source and one destination resource). All links in SMIL are actuated by the user.

Handling of Links in Embedded Documents

Due to its integrating nature, the presentation of a SMIL document may involve other (non-SMIL) applications or plug-ins. For example, a SMIL browser may use an HTML plug-in to display an embedded HTML page. Vice versa, an HTML browser may use a SMIL plug-in to display a SMIL document embedded in an HTML page.

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

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

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

Addressing

SMIL supports name fragment identifiers and the '#' connector. This means that SMIL supports locators as currently used in HTML (e.g. it uses locators of the form "`http://foo.com/some/path#anchor1`").

Linking to SMIL Fragments

A locator that points to a SMIL document may contain a fragment part (e.g. `http://www.w3.org/test.smi#par1`). The fragment part is an id value that identifies one of the elements within the referenced SMIL document. If a link containing a fragment part is followed, the presentation should start as if the user had fast-forwarded the presentation represented by the destination document to the effective begin of the element designated by the fragment.

The following special cases can occur:

1. The element addressed by the link has a "**`repeat`**" attribute.
    1. If the value of the **`repeat`** attribute is N, all N repetitions of the element are played.
    2. If the value of the **`repeat`** attribute is "**`indefinite`**", playback ends according to the rules defined for **`repeat`** value **`indefinite.`**
2. The element addressed by the link is contained within another element that contains a **`repeat`** attribute.
    1. If the value of the **`repeat`** attribute is N, playback starts at the beginning of the element addressed by the link, followed by N-1 repetitions of the element containing the **`repeat`** attribute.

   2. If the value of the **repeat** attribute is "**indefinite** ", playback starts at the beginning of the element addressed by the link. Playback ends according to the rules defined for repeat value "**indefinite** ".
3. The element addressed by the link is content of a **switch** element: It is forbidden to link to elements that are the content of **switch** elements.

4.5.1 The a Element

The functionality of the **a** element is very similar to the functionality of the **a** element in HTML 4.0 [HTML40] . SMIL adds an attribute "show" that controls the temporal behavior of the source when the link is followed. For synchronization purposes, the **a** element is transparent, i.e. it does not influence the synchronization of its child elements. **a** elements may not be nested. An **a** element must have an **href** attribute.

Attributes

The **a** element can have the following attributes:

id Defined in Section 2

href This attribute contains the URI of the link's destination.
The **href** attribute is required for **a** elements.

show This attribute controls the behavior of the source document containing the link when the link is followed. It can have one of the following values:

replace:	The current presentation is paused at its current state and is replaced by the destination resource. If the player offers a history mechanism, the source presentation resumes from the state in which it was paused when the user returns to it.
new:	The presentation of the destination resource starts in a new context, not affecting the source resource.
pause:	The source presentation is paused at its current state, and the destination resource starts in a new context. When the display of the destination resource ends, the source presentation resumes from the state in which it was paused.

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

title Defined in Section 3.3.1

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

Element Content

The **a** element can contain the following children:

animation	Defined in Section 4.2.3
audio	Defined in Section 4.2.3
img	Defined in Section 4.2.3
par	Defined in Section 4.2.1
ref	Defined in Section 4.2.3

seq	Defined in Section 4.2.2
switch	Defined in Section 4.3
text	Defined in Section 4.2.3
textstream	Defined in Section 4.2.3
video	Defined in Section 4.2.3

Examples

Example 1

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

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

In the example, the second line can be replaced by a reference to any valid subtree of an SMIL presentation.

Example 2

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

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

For example, this allows a SMIL player to spawn off an HTML browser.

Example 3

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

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

Example 4

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

```
Presentation A:

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


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

      ...
```

```
<seq>
  <video src="rtsp://foo.com/graph.imf"/>
  <par>
    <video src="rtsp://foo.com/timbl.rm" region="l_window"/>
    <video id="next" src="rtsp://foo.com/v1.rm" region="r_window"/>
           ^^^^^^^^^
    <text src="rtsp://foo.com/caption1.html" region="l_2_title"/>
    <text src="rtsp://foo.com/caption2.rtx" region="r_2_title"/>
  </par>
</seq>
...
```

4.5.2 The `anchor` Element

The functionality of the "a" element is restricted in that it only allows associating a link with a complete media object. HTML image maps have demonstrated that it is useful to associate links with spatial subparts of an object. The anchor element realizes similar functionality for SMIL:

1. The anchor element allows associating a link destination to spatial and temporal subparts of a media object, using the "href" attribute (in contrast, the "a" element only allows associating a link with a complete media object).
2. The anchor element allows making a subpart of the media object the destination of a link, using the "id" attribute.
3. The anchor element allows breaking up an object into spatial subparts, using the "coords" attribute.
4. The anchor element allows breaking up an object into temporal subparts, using the "begin" and "end" attributes. The values of the begin and end attributes are relative to the beginning of the media object.

Attributes

The **`anchor`** element can have the following attributes:

`begin` Defined in Section 4.2.1

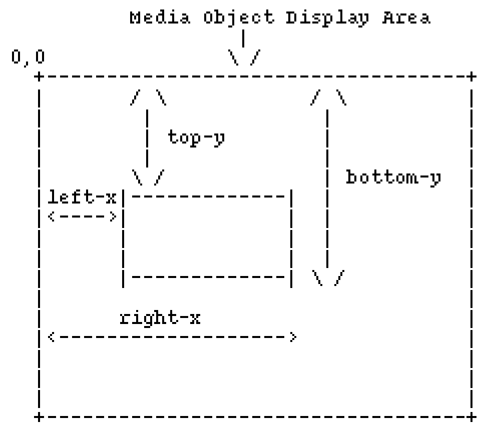
`coords`

The value of this attribute specifies a rectangle within the display area of a visual media object. Syntax and semantics of this attribute are similar to the coords attribute in HTML image maps, when the link is associated with a rectangular shape. The rectangle is specified by four length values: The first two values specify the coordinates of the upper left corner of the rectangle.The second two values specify the coordinates of the lower right corner of the rectangle. Coordinates are relative to the top left corner of the visual media object (see Figure 4.5). If a coordinate is specified as a percentage value, it is relative to the total width or height of the media object display area.

An attribute with an erroneous coords value is ignored (right-x smaller or equal to left-x, bottom-y smaller or equal to top-y). If the rectangle defined by the coords attribute exceeds the area covered by the media object, exceeding height and width are clipped at the borders of the media object.

Values of the coords attribute have the following syntax:

```
coords-value ::= left-x "," top-y "," right-x "," bottom-y
```



D

Figure 4.5: Semantics of "coords" attribute

---

end	Defined in Section 4.2.1
id	Defined in Section 2
show	Defined in Section 4.5.1
skip-content	Defined in Section 3.3.1
title	Defined in Section 3.3.1 It is strongly recommended that all anchor elements have a `title` attribute with a meaningful description. Authoring tools should ensure that no element can be introduced into a SMIL document without this attribute.

Examples

1) Associating links with spatial subparts

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

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

2) Associating links with temporal subparts

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

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

3) Jumping to a subpart of an object

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

```
Presentation A:

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


Presentation B:

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

4) Combining different uses of links

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

```
Presentation A:

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


Presentation B:

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

5 SMIL DTD

5.1 Relation to XML

A SMIL 1.0 document may optionally contain a document type declaration, which names the document type definition (DTD) in use for the document. For SMIL, the document type declaration should look as follows (the double quotes can be replaced by single quotes):

```
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
     "http://www.w3.org/TR/REC-smil/SMIL10.dtd">
```

The XML 1.0 specification provides a way to extend the DTD using the <!DOCTYPE> element, for instance to add a new set of entity definitions. Authors must not use this feature with SMIL since many SMIL players will not support it.

The following is illegal in SMIL:

```
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
     "http://www.w3.org/TR/REC-smil/SMIL10.dtd" [
<!ENTITY % AcmeCorpSymbols PUBLIC
     "-//Acme Corp//ENTITIES Corporate Symbols//EN"
     "http://www.acme.com/corp_symbols.xml"
>
%AcmeCorpSymbols;
]>
```

5.2 DTD

```
<!--

    This is the XML document type definition (DTD) for SMIL 1.0.

    Date: 1998/06/15 08:56:30

    Authors:
        Jacco van Ossenbruggen <jrvosse@cwi.nl>
        Sjoerd Mullender       <sjoerd@cwi.nl>

    Further information about SMIL is available at:

          http://www.w3.org/AudioVideo/
-->

<!-- Generally useful entities -->
<!ENTITY % id-attr "id ID #IMPLIED">
<!ENTITY % title-attr "title CDATA #IMPLIED">
<!ENTITY % skip-attr "skip-content (true|false) 'true'">
<!ENTITY % desc-attr "
        %title-attr;
        abstract        CDATA   #IMPLIED
        author          CDATA   #IMPLIED
        copyright       CDATA   #IMPLIED
">
```

```
<!--=================== SMIL Document ===============================-->
<!--
     The root element SMIL contains all other elements.
-->
<!ELEMENT smil (head?,body?)>
<!ATTLIST smil
        %id-attr;
>

<!--=================== The Document Head ===========================-->
<!ENTITY % layout-section "layout|switch">

<!ENTITY % head-element "(meta*,((%layout-section;), meta*))?">

<!ELEMENT head %head-element;>
<!ATTLIST head %id-attr;>

<!--=================== Layout Element ==============================-->
<!--
     Layout contains the region and root-layout elements defined by
     smil-basic-layout or other elements defined an external layout
     mechanism.
-->
<!ELEMENT layout ANY>
<!ATTLIST layout
        %id-attr;
        type CDATA      "text/smil-basic-layout"
>

<!--=================== Region Element ==============================-->
<!ENTITY % viewport-attrs "
        height              CDATA    #IMPLIED
        width               CDATA    #IMPLIED
        background-color    CDATA    #IMPLIED
">

<!ELEMENT region EMPTY>
<!ATTLIST region
        %id-attr;
        %title-attr;
        %viewport-attrs;
        left                CDATA    "0"
        top                 CDATA    "0"
        z-index             CDATA    "0"
        fit                 (hidden|fill|meet|scroll|slice)    "hidden"
        %skip-attr;
>

<!--=================== Root-layout Element =========================-->
<!ELEMENT root-layout EMPTY>
<!ATTLIST root-layout
        %id-attr;
        %title-attr;
        %viewport-attrs;
        %skip-attr;
>
```

```
<!--==================== Meta Element ================================-->
<!ELEMENT meta EMPTY>
<!ATTLIST meta
        name    NMTOKEN #REQUIRED
        content CDATA   #REQUIRED
        %skip-attr;
>

<!--==================== The Document Body ==========================-->
<!ENTITY % media-object
"audio|video|text|img|animation|textstream|ref">
<!ENTITY % schedule "par|seq|(%media-object;)">
<!ENTITY % inline-link "a">
<!ENTITY % assoc-link "anchor">
<!ENTITY % link "%inline-link;">
<!ENTITY % container-content "(%schedule;)|switch|(%link;)">
<!ENTITY % body-content "(%container-content;)">

<!ELEMENT body (%body-content;)*>
<!ATTLIST body %id-attr;>

<!--================ Synchronization Attributes =====================-->
<!ENTITY % sync-attributes "
        begin   CDATA   #IMPLIED
        end     CDATA   #IMPLIED
">

<!--================== Switch Parameter Attributes ===================-->
<!ENTITY % system-attribute "
        system-bitrate              CDATA                #IMPLIED
        system-language             CDATA                #IMPLIED
        system-required             NMTOKEN              #IMPLIED
        system-screen-size          CDATA                #IMPLIED
        system-screen-depth         CDATA                #IMPLIED
        system-captions             (on|off)             #IMPLIED
        system-overdub-or-caption   (caption|overdub)    #IMPLIED
">

<!--================== Fill Attribute ================================-->
<!ENTITY % fill-attribute "
        fill     (remove|freeze)     'remove'
">

<!--=================== The Parallel Element ========================-->
<!ENTITY % par-content "%container-content;">
<!ELEMENT par    (%par-content;)*>
<!ATTLIST par
        %id-attr;
        %desc-attr;
        endsync CDATA           "last"
        dur     CDATA           #IMPLIED
        repeat  CDATA           "1"
        region  IDREF           #IMPLIED
        %sync-attributes;
        %system-attribute;
>
```

```
<!--================= The Sequential Element =======================-->
<!ENTITY % seq-content "%container-content;">
<!ELEMENT seq    (%seq-content;)*>
<!ATTLIST seq
        %id-attr;
        %desc-attr;
        dur     CDATA           #IMPLIED
        repeat  CDATA           "1"
        region  IDREF           #IMPLIED
        %sync-attributes;
        %system-attribute;
>

<!--================== The Switch Element =========================-->
<!-- In the head, a switch may contain only layout elements,
     in the body, only container elements. However, this
     constraint cannot be expressed in the DTD (?), so
     we allow both:
-->
<!ENTITY % switch-content "layout|(%container-content;)">
<!ELEMENT switch (%switch-content;)*>
<!ATTLIST switch
        %id-attr;
        %title-attr;
>

<!--================= Media Object Elements =======================-->
<!-- SMIL only defines the structure. The real media data is
     referenced by the src attribute of the media objects.
-->

<!-- Furthermore, they have the following attributes as defined
     in the SMIL specification:
-->
<!ENTITY % mo-attributes "
        %id-attr;
        %desc-attr;
        region     IDREF           #IMPLIED
        alt        CDATA           #IMPLIED
        longdesc   CDATA           #IMPLIED
        src        CDATA           #IMPLIED
        type       CDATA           #IMPLIED
        dur        CDATA           #IMPLIED
        repeat     CDATA           '1'
        %fill-attribute;
        %sync-attributes;
        %system-attribute;
">
```

```
<!--
     Most info is in the attributes, media objects are empty or
     contain associated link elements:
-->
<!ENTITY % mo-content "(%assoc-link;)*">
<!ENTITY % clip-attrs "
        clip-begin      CDATA   #IMPLIED
        clip-end        CDATA   #IMPLIED
">

<!ELEMENT ref           %mo-content;>
<!ELEMENT audio         %mo-content;>
<!ELEMENT img           %mo-content;>
<!ELEMENT video         %mo-content;>
<!ELEMENT text          %mo-content;>
<!ELEMENT textstream    %mo-content;>
<!ELEMENT animation     %mo-content;>

<!ATTLIST ref           %mo-attributes; %clip-attrs;>
<!ATTLIST audio         %mo-attributes; %clip-attrs;>
<!ATTLIST video         %mo-attributes; %clip-attrs;>
<!ATTLIST animation     %mo-attributes; %clip-attrs;>
<!ATTLIST textstream    %mo-attributes; %clip-attrs;>
<!ATTLIST text          %mo-attributes;>
<!ATTLIST img           %mo-attributes;>

<!--================= Link Elements =================================-->

<!ENTITY % smil-link-attributes "
        %id-attr;
        %title-attr;
        href            CDATA                   #REQUIRED
        show            (replace|new|pause)     'replace'
">


<!--================= Inline Link Element ============================-->
<!ELEMENT a (%schedule;|switch)*>
<!ATTLIST a
        %smil-link-attributes;
>


<!--================= Associated Link Element ========================-->
<!ELEMENT anchor EMPTY>
<!ATTLIST anchor
        %skip-attr;
        %smil-link-attributes;
        %sync-attributes;
        coords          CDATA                   #IMPLIED
>
```

References

[CSS2]
"Cascading Style Sheets, level 2", B. Bos, H. Lie, C. Lilley, I. Jacobs, 12 May 1998.
Available at http://www.w3.org/TR/REC-CSS2/.

[HTML40]
"HTML 4.0 Specification", D. Raggett, A. Le Hors, I. Jacobs, 24 April 1998.
Available at http://www.w3.org/TR/REC-html40.

[ISO/IEC 10646]
ISO (International Organization for Standardization). ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

[NAMESPACES]
"Namespaces in XML", T. Bray, D. Hollander, A. Layman, 27 March 1998
W3C working draft. Available at http://www.w3.org/TR/WD-xml-names.

[PICS]
"PICS 1.1 Label Distribution -- Label Syntax and Communication Protocols", 31 October 1996, T. Krauskopf, J. Miller, P. Resnick, W. Trees
Available at http://www.w3.org/TR/REC-PICS-labels-961031

[RFC1738]
"Uniform Resource Locators", T. Berners-Lee, L. Masinter, and M. McCahill, December 1994.
Available at ftp://ftp.isi.edu/in-notes/rfc1738.txt.

[RFC1766]
"Tags for the Identification of Languages", H. Alvestrand, March 1995.
Available at ftp://ftp.isi.edu/in-notes/rfc1766.txt.

[RFC1808]
"Relative Uniform Resource Locators", R. Fielding, June 1995.
Available at ftp://ftp.isi.edu/in-notes/rfc1808.txt.

[RFC2045]
"Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed and N. Borenstein, November 1996.
Available at ftp://ftp.isi.edu/in-notes/rfc2045.txt. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

[SMPTE]
"Time and Control Codes for 24, 25 or 30 Frame-Per-Second Motion-Picture Systems - RP 136-1995". Society of Motion Picture & Television Engineers.

[URI]
"Uniform Resource Identifiers (URI): Generic Syntax and Semantics", T. Berners-Lee, R. Fielding, L. Masinter, 4 March 1998.
Available at http://www.ics.uci.edu/pub/ietf/uri/draft-fielding-uri-syntax-02.txt. This is a work in progress that is expected to update [RFC1738] and [RFC1808].

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

Appendix

Extending SMIL 1.0

(non-normative)

In the future, SMIL 1.0 may be extended by another W3C recommendation, or by private extensions.

For these extensions, it is recommended that the following rules are obeyed:

- All elements introduced in extensions must have a "skip-content" attribute (defined in Section 3.3.1) if it should be possible that their content is processed by SMIL 1.0 players.
- Private extensions must be introduced using the syntax of the XML namespace specification.

It is recommended that SMIL 1.0 players are prepared to handle documents that contain extension that obey these two rules.

Extensions should be handled using an XML namespace mechanism, once such a mechanism becomes a W3C recommendation. In the rest of the section, the syntax and semantics for XML namespaces defined in the W3C note [NAMESPACE] will be used for demonstration purposes only.

The following cases can occur:

1. The document contains a namespace declaration for the SMIL 1.0 specification that defines an empty prefix. In this case, non-SMIL 1.0 elements and attributes are only allowed in a document if they are declared using an XML namespace. The document may not contain a document type declaration for SMIL 1.0. If it does, it is invalid.

   In the following example, the element "new:a" is a legal extension. The elements "`mytags:a`" and "`b`" are syntax errors, since they are not declared using an XML namespace.

```
<?xml:namespace ns="http://www.acme.com/new-smil" prefix="new" ?>
<?xml:namespace ns="http://www.w3.org/TR/PR-smil" ?>
<smil>
  <body>
     <par>
        <new:a>
          ...
        </new:a>
        <mytags:a ... />
            ...
        </mytags:a>
       <b>
         ...
       </b>
     </par>
   </body>
 </smil>
```

2. The document contains no document type declaration, it contains a document type declaration for a SMIL version higher than 1.0, or it contains a namespace declaration for a SMIL specification with a version higher than 1.0. For a SMIL 1.0 player to be able to recognize such a namespace declaration, it is recommended that the URI of future SMIL versions starts with

`http://www.w3.org/TR/REC-smil`, and is followed by more characters which may for example be a version number.

In this case, a SMIL 1.0 player should assume that it is processing a SMIL document with a version number higher than 1.0.

The following cases can occur:

Unknown element

Unknown elements are ignored

An unknown element may contain content that consists of SMIL 1.0 elements. Whether such content is ignored or processed depends on the value of the "skip-content" attribute. If the attribute is set to "true", or the attribute is absent, the content is not processed. If it is set to "false", the content is processed.

Content in Element that was declared "Empty"

A future version of SMIL may allow content in elements that are declared as "empty" in SMIL 1.0. Whether this content is ignored or not depends on the value of the "skip-content" attribute of the formerly empty element. If the attribute is set to "true", the content is not processed. If it is set to "false", the content is processed.

Unknown Attribute
Unknown attributes are ignored.

Unknown Attribute Value
Attributes with unknown attribute values are ignored.

3. The document contains a document type declaration for SMIL 1.0. In this case, it may not contain any non-SMIL 1.0 elements, even if they are declared using XML namespaces. This is because such extensions would render the document invalid.

Using SMIL 1.0 as an Extension

When the XML namespace mechanism is used to include SMIL elements and attributes in other XML-based documents, it is recommended to use the following namespace identifier:
`http://www.w3.org/TR/REC-smil`

SMIL 2.0 DTDs

Contents

A. SMIL 2.0 Driver File

```
<!-- ...................................................................... -->
<!-- SMIL 2.0 DTD  ......................................................... -->
<!-- file: SMIL20.dtd
-->
<!-- SMIL 2.0 DTD

      This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

 This is the driver file for the SMIL 2.0 DTD.

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

     PUBLIC "-//W3C//DTD SMIL 2.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL20.dtd"

-->

<!ENTITY % NS.prefixed "IGNORE" >
<!ENTITY % SMIL.prefix "" >

<!-- Define the Content Model -->
<!ENTITY % smil-model.mod
    PUBLIC "-//W3C//ENTITIES SMIL 2.0 Document Model 1.0//EN"
           "smil-model-1.mod" >

<!-- Modular Framework Module  ................................... -->
<!ENTITY % smil-framework.module "INCLUDE" >
<![%smil-framework.module;[
<!ENTITY % smil-framework.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN"
            "smil-framework-1.mod" >
%smil-framework.mod;]]>

<!-- The SMIL 2.0 Profile includes the following sections:
             C. The SMIL Animation Module
             D. The SMIL Content Control Module
             G. The SMIL Layout Module
             H. The SMIL Linking Module
             I. The SMIL Media Object Module
             J. The SMIL Metainformation Module
             K. The SMIL Structure Module
             L. The SMIL Timing and Synchronization Module
             M. Integrating SMIL Timing into other XML-Based Languages
             P. The SMIL Transition effects Module

             The SMIL Streaming Media Object Module is optional.
-->
```

```
<!--
<!ENTITY % smil-streamingmedia.model "IGNORE">
<![%smil-streamingmedia.model;[
  <!ENTITY % smil-streaming-mod
    PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Streaming Media Objects//EN"
    "SMIL-streamingmedia.mod">
  %smil-streaming-mod;
]]>
-->

<!ENTITY % SMIL.anim-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Animation//EN"
  "SMIL-anim.mod">
<!ENTITY % SMIL.control-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Content Control//EN"
  "SMIL-control.mod">
<!ENTITY % SMIL.layout-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Layout//EN"
  "SMIL-layout.mod">
<!ENTITY % SMIL.link-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Linking//EN"
  "SMIL-link.mod">
<!ENTITY % SMIL.media-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Media Objects//EN"
  "SMIL-media.mod">
<!ENTITY % SMIL.meta-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Metainformation//EN"
  "SMIL-metainformation.mod">
<!ENTITY % SMIL.struct-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Structure//EN"
  "SMIL-struct.mod">
<!ENTITY % SMIL.timing-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Timing//EN"
  "SMIL-timing.mod">
<!ENTITY % SMIL.transition-mod
  PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Transition//EN"
  "SMIL-transition.mod">

%SMIL.struct-mod;
%SMIL.anim-mod;
%SMIL.control-mod;
%SMIL.meta-mod;
%SMIL.layout-mod;
%SMIL.link-mod;
%SMIL.media-mod;
%SMIL.timing-mod;
%SMIL.transition-mod;

<!-- end of SMIL20.dtd -->
```

B. SMIL 2.0 Modular Framework Module

```
<!-- ...................................................................... -->
<!-- SMIL 2.0 Modular Framework Module  .................................. -->
<!-- file: smil-framework-1.mod

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Modular Framework 1.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/smil-framework-1.mod"
-->

<!-- Modular Framework

     This required module instantiates the modules needed
     to support the SMIL 2.0 modularization model, including:

        +  datatypes
        +  namespace-qualified names
        +  common attributes
        +  document model
-->

<!ENTITY % smil-datatypes.module "INCLUDE" >
<![%smil-datatypes.module;[
<!ENTITY % smil-datatypes.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Datatypes 1.0//EN"
            "smil-datatypes-1.mod" >
%smil-datatypes.mod;]]>

<!ENTITY % smil-qname.module "INCLUDE" >
<![%smil-qname.module;[
<!ENTITY % smil-qname.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Qualified Names 1.0//EN"
            "smil-qname-1.mod" >
%smil-qname.mod;]]>

<!ENTITY % smil-attribs.module "INCLUDE" >
<![%smil-attribs.module;[
<!ENTITY % smil-attribs.mod
     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Common Attributes 1.0//EN"
            "smil-attribs-1.mod" >
%smil-attribs.mod;]]>

<!ENTITY % smil-model.module "INCLUDE" >
<![%smil-model.module;[
<!-- A content model MUST be defined by the driver file -->
%smil-model.mod;]]>
<!-- end of smil-framework-1.mod -->
```

B.1. SMIL 2.0 Common Attributes Module

```
<!-- ...................................................................... -->
<!-- SMIL 2.0 Common Attributes Module  .................................. -->
<!-- file: smil-attribs-1.mod

      This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Common Attributes 1.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/smil-attribs-1.mod"

     ....................................................................... -->

<!-- Common Attributes

     This module declares the common attributes for the SMIL DTD Modules.
-->

<!ENTITY % SMIL.pfx "">

<!ENTITY % id.attrib
 "%SMIL.pfx;id           ID                     #IMPLIED"
>

<!ENTITY % class.attrib
 "%SMIL.pfx;class        CDATA                  #IMPLIED"
>

<!ENTITY % title.attrib
 "%SMIL.pfx;title        %Text.datatype;        #IMPLIED"
>

<!ENTITY % longdesc.attrib
 "%SMIL.pfx;longdesc     %URI.datatype;         #IMPLIED"
>

<!ENTITY % alt.attrib
 "%SMIL.pfx;alt          %Text.datatype;        #IMPLIED"
>

<!ENTITY % SMIL.Accessibility.attrib "
 %longdesc.attrib;
 %alt.attrib;
">

<!ENTITY % Core.extra.attrib "" >
<!ENTITY % Core.attrib "
  xml:base              %URI.datatype;          #IMPLIED
  %id.attrib;
  %class.attrib;
  %title.attrib;
  %SMIL.Accessibility.attrib;
  %Core.extra.attrib;
">

<!ENTITY % I18n.extra.attrib "" >
<!ENTITY % I18n.attrib "
  xml:lang            %LanguageCode.datatype;   #IMPLIED
  %I18n.extra.attrib;"
>

<!ENTITY % SMIL.Description.attrib "
 %SMIL.pfx;abstract        %Text.datatype;      #IMPLIED
 %SMIL.pfx;author          %Text.datatype;      #IMPLIED
 %SMIL.pfx;copyright       %Text.datatype;      #IMPLIED
">

<!ENTITY % SMIL.tabindex.attrib "
 %SMIL.pfx;tabindex        %Number.datatype;    #IMPLIED
">

<!-- ================== BasicLayout ======================================= -->
<!ENTITY % SMIL.regionAttr.attrib "
 %SMIL.pfx;region          CDATA                #IMPLIED
">

<!ENTITY % SMIL.fill.attrib "
 %SMIL.pfx;fill (remove|freeze|hold|transition|auto|default) 'default'
">

<!ENTITY % SMIL.fillDefault.attrib "
 %SMIL.pfx;fillDefault (remove|freeze|hold|transition|auto|inherit) 'inherit'
">

<!-- ================== HierarchicalLayout ================================ -->
<!ENTITY % SMIL.backgroundColor.attrib "
 %SMIL.pfx;backgroundColor     CDATA    #IMPLIED
">
<!ENTITY % SMIL.backgroundColor-deprecated.attrib "
 %SMIL.pfx;background-color     CDATA    #IMPLIED
">

<!ENTITY % SMIL.Sub-region.attrib "
 %SMIL.pfx;top     CDATA    'auto'
 %SMIL.pfx;bottom  CDATA    'auto'
 %SMIL.pfx;left    CDATA    'auto'
 %SMIL.pfx;right   CDATA    'auto'
 %SMIL.pfx;height  CDATA    'auto'
 %SMIL.pfx;width   CDATA    'auto'
 %SMIL.pfx;z-index CDATA    #IMPLIED
">

<!ENTITY % SMIL.fit.attrib "
 %SMIL.pfx;fit             (hidden|fill|meet|scroll|slice)   #IMPLIED
">
```

```
<!-- ================ Registration Point attribute for media elements
============ -->
<!-- integrating language using HierarchicalLayout must include regPoint   -->
<!-- attribute on media elements for regPoint elements to be useful        -->

<!ENTITY % SMIL.regPointAttr.attrib "
 %SMIL.pfx;regPoint  CDATA    #IMPLIED
">

<!ENTITY % SMIL.regAlign.attrib "
 %SMIL.pfx;regAlign  (topLeft|topMid|topRight|midLeft|center|
                     midRight|bottomLeft|bottomMid|bottomRight) #IMPLIED
">

<!ENTITY % SMIL.RegistrationPoint.attrib "
 %SMIL.regPointAttr.attrib;
 %SMIL.regAlign.attrib;
">

<!--=================== Content Control =======================-->
<!-- customTest Attribute, do not confuse with customTest element! -->
<!ENTITY % SMIL.customTestAttr.attrib "
      %SMIL.pfx;customTest              IDREF          #IMPLIED
">

<!-- ========================= SkipContentControl Module
========================= -->
<!ENTITY % SMIL.skip-content.attrib "
      %SMIL.pfx;skip-content        (true|false)      'true'
">

<!-- Content Control Test Attributes -->

<!ENTITY % SMIL.Test.attrib "
      %SMIL.pfx;systemBitrate              CDATA                #IMPLIED
      %SMIL.pfx;systemCaptions             (on|off)             #IMPLIED

      %SMIL.pfx;systemLanguage             CDATA                #IMPLIED
      %SMIL.pfx;systemOverdubOrSubtitle    (overdub|subtitle)   #IMPLIED
      %SMIL.pfx;systemRequired             CDATA                #IMPLIED
      %SMIL.pfx;systemScreenSize           CDATA                #IMPLIED
      %SMIL.pfx;systemScreenDepth          CDATA                #IMPLIED
      %SMIL.pfx;systemAudioDesc            (on|off)             #IMPLIED
      %SMIL.pfx;systemOperatingSystem      NMTOKEN              #IMPLIED
      %SMIL.pfx;systemCPU                  NMTOKEN              #IMPLIED
      %SMIL.pfx;systemComponent            CDATA                #IMPLIED

      %SMIL.pfx;system-bitrate             CDATA                #IMPLIED
      %SMIL.pfx;system-captions            (on|off)             #IMPLIED
      %SMIL.pfx;system-language            CDATA                #IMPLIED
      %SMIL.pfx;system-overdub-or-caption  (overdub|caption)    #IMPLIED
      %SMIL.pfx;system-required            CDATA                #IMPLIED
      %SMIL.pfx;system-screen-size         CDATA                #IMPLIED
      %SMIL.pfx;system-screen-depth        CDATA                #IMPLIED
">
```

```
<!-- SMIL Animation Module  =================================================== -->
<!ENTITY % SMIL.BasicAnimation.attrib "
  %SMIL.pfx;values             CDATA                       #IMPLIED
  %SMIL.pfx;from               CDATA                       #IMPLIED
  %SMIL.pfx;to                 CDATA                       #IMPLIED
  %SMIL.pfx;by                 CDATA                       #IMPLIED
">

<!-- SMIL Timing Module  ====================================================== -->
<!ENTITY % SMIL.BasicInlineTiming.attrib "
  %SMIL.pfx;dur                %TimeValue.datatype;    #IMPLIED
  %SMIL.pfx;repeatCount        %TimeValue.datatype;    #IMPLIED
  %SMIL.pfx;repeatDur          %TimeValue.datatype;    #IMPLIED
  %SMIL.pfx;begin              %TimeValue.datatype;    #IMPLIED
  %SMIL.pfx;end                %TimeValue.datatype;    #IMPLIED
">

<!ENTITY % SMIL.MinMaxTiming.attrib "
  %SMIL.pfx;min                %TimeValue.datatype;    '0'
  %SMIL.pfx;max                %TimeValue.datatype;    'indefinite'
">

<!ENTITY % SMIL.BasicInlineTiming-deprecated.attrib "
  %SMIL.pfx;repeat             %TimeValue.datatype;    #IMPLIED
">

<!ENTITY % SMIL.endsync.attrib "
  %SMIL.pfx;endsync            CDATA                       'last'
">

<!-- endsync has a different default when applied to media elements -->
<!ENTITY % SMIL.endsync.media.attrib "
  %SMIL.pfx;endsync            CDATA                       'media'
">

<!ENTITY % SMIL.TimeContainerAttributes.attrib "
  %SMIL.pfx;timeAction         CDATA                       #IMPLIED
  %SMIL.pfx;timeContainer      CDATA                       #IMPLIED
">

<!ENTITY % SMIL.RestartTiming.attrib "
  %SMIL.pfx;restart  (always|whenNotActive|never|default)  'default'
">

<!ENTITY % SMIL.RestartDefaultTiming.attrib "
  %SMIL.pfx;restartDefault  (inherit|always|never|whenNotActive)  'inherit'
">

<!ENTITY % SMIL.SyncBehavior.attrib "
  %SMIL.pfx;syncBehavior  (canSlip|locked|independent|default)  'default'
  %SMIL.pfx;syncTolerance %TimeValue.datatype;                  'default'
">

<!ENTITY % SMIL.SyncBehaviorDefault.attrib "
  %SMIL.pfx;syncBehaviorDefault  (canSlip|locked|independent|inherit)  'inherit'
  %SMIL.pfx;syncToleranceDefault  %TimeValue.datatype;                 'inherit'
```

```
">

<!ENTITY % SMIL.SyncMaster.attrib "
  %SMIL.pfx;syncMaster        (true|false)            'false'
">

<!-- ================== Time Manipulations ================================== -->
<!ENTITY % SMIL.TimeManipulations.attrib "
  %SMIL.pfx;accelerate        %Number.datatype;       '0'
  %SMIL.pfx;decelerate        %Number.datatype;       '0'
  %SMIL.pfx;speed             %Number.datatype;       '1.0'
  %SMIL.pfx;autoReverse       (true|false)            'false'
">

<!-- ================== Media Objects ======================================= -->
<!ENTITY % SMIL.MediaClip.attrib "
  %SMIL.pfx;clipBegin         CDATA                   #IMPLIED
  %SMIL.pfx;clipEnd           CDATA                   #IMPLIED
">
<!ENTITY % SMIL.MediaClip.attrib.deprecated "
  %SMIL.pfx;clip-begin        CDATA                   #IMPLIED
  %SMIL.pfx;clip-end          CDATA                   #IMPLIED
">

<!-- ================== Streaming Media ==================================== -->
<!ENTITY % SMIL.Streaming-media.attrib "
  %SMIL.pfx;port              CDATA                   #IMPLIED
  %SMIL.pfx;rtpformat         CDATA                   #IMPLIED
  %SMIL.pfx;transport         CDATA                   #IMPLIED
">

<!ENTITY % SMIL.Streaming-timecontainer.attrib "
  %SMIL.pfx;control           CDATA                   #IMPLIED
">


<!-- ================== Transitions Media ================================== -->
<!ENTITY % SMIL.Transition.attrib "
 %SMIL.pfx;transIn            CDATA                   #IMPLIED
 %SMIL.pfx;transOut           CDATA                   #IMPLIED
">

<!-- end of smil-attribs-1.mod -->
```

B.2. SMIL 2.0 Document Model Module

```
<!-- ======================================================================  -->
<!-- SMIL 2.0 Document Model Module ========================================  -->
<!-- file: smil-model-1.mod

     This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author: Warner ten Kate, Jacco van Ossenbruggen, Aaron Cohen
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ENTITIES SMIL 2.0 Document Model 1.0//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/smil-model-1.mod"

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

<!--
        This file defines the SMIL 2.0 Language Document Model.
        All attributes and content models are defined in the second
        half of this file.  We first start with some utility definitions.
        These are mainly used to simplify the use of Modules in the
        second part of the file.

-->

<!-- ================== Util: Head ========================================= -->
<!ENTITY % SMIL.head-meta.content       "%SMIL.metadata.qname;">
<!ENTITY % SMIL.head-layout.content     "%SMIL.layout.qname;
                                       | %SMIL.switch.qname;">
<!ENTITY % SMIL.head-control.content    "%SMIL.customAttributes.qname;">
<!ENTITY % SMIL.head-transition.content "%SMIL.transition.qname;+">

<!--=================== Util: Body - Content Control ======================= -->
<!ENTITY % SMIL.content-control "%SMIL.switch.qname; | %SMIL.prefetch.qname;">
<!ENTITY % SMIL.content-control-attrs "%SMIL.Test.attrib;
                                       %SMIL.customTestAttr.attrib;
                               %SMIL.skip-content.attrib;">

<!--=================== Util: Body - Animation ========================= -->
<!ENTITY % SMIL.animation.elements "%SMIL.animate.qname;
                                    | %SMIL.set.qname;
                                    | %SMIL.animateMotion.qname;
                                    | %SMIL.animateColor.qname;">

<!--=================== Util: Body - Media ========================= -->

<!ENTITY % SMIL.media-object "%SMIL.audio.qname;
                              | %SMIL.video.qname;
                              | %SMIL.animation.qname;
                              | %SMIL.text.qname;
                              | %SMIL.img.qname;
                              | %SMIL.textstream.qname;
                              | %SMIL.ref.qname;
                              | %SMIL.brush.qname;
                              | %SMIL.animation.elements;">

<!--================== Util: Body - Timing ================================ -->
<!ENTITY % SMIL.BasicTimeContainers.class "%SMIL.par.qname;
                                          | %SMIL.seq.qname;">

<!ENTITY % SMIL.ExclTimeContainers.class "%SMIL.excl.qname;">

<!ENTITY % SMIL.timecontainer.class   "%SMIL.BasicTimeContainers.class;
                                       |%SMIL.ExclTimeContainers.class;">

<!ENTITY % SMIL.timecontainer.content "%SMIL.timecontainer.class;
                                     | %SMIL.media-object;
                                     | %SMIL.content-control;
                                     | %SMIL.a.qname;">

<!ENTITY % SMIL.smil-basictime.attrib "
 %SMIL.BasicInlineTiming.attrib;
 %SMIL.BasicInlineTiming-deprecated.attrib;
 %SMIL.MinMaxTiming.attrib;
">

<!ENTITY % SMIL.timecontainer.attrib "

 %SMIL.BasicInlineTiming.attrib;
 %SMIL.BasicInlineTiming-deprecated.attrib;
 %SMIL.MinMaxTiming.attrib;
 %SMIL.RestartTiming.attrib;
 %SMIL.RestartDefaultTiming.attrib;
 %SMIL.SyncBehavior.attrib;
 %SMIL.SyncBehaviorDefault.attrib;
 %SMIL.fillDefault.attrib;
">

<!-- ================================================================== -->
<!-- ================================================================== -->
<!-- ================================================================== -->

<!--
     The actual content model and attribute definitions for each module
     sections follow below.
-->

<!-- ================= Content Control =================================== -->
<!ENTITY % SMIL.BasicContentControl.module  "INCLUDE">
<!ENTITY % SMIL.CustomTestAttributes.module "INCLUDE">
<!ENTITY % SMIL.PrefetchControl.module      "INCLUDE">
<!ENTITY % SMIL.skip-contentControl.module   "INCLUDE">

<!ENTITY % SMIL.switch.content "((%SMIL.timecontainer.class;
                               | %SMIL.media-object;
                               | %SMIL.content-control;
                               | %SMIL.a.qname;
                               | %SMIL.area.qname;
                              | %SMIL.anchor.qname;)*
                              | %SMIL.layout.qname;*)">

<!ENTITY % SMIL.switch.attrib "%SMIL.Test.attrib; %SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.prefetch.attrib "
  %SMIL.timecontainer.attrib;
  %SMIL.MediaClip.attrib;
  %SMIL.MediaClip.attrib.deprecated;
  %SMIL.Test.attrib;
  %SMIL.customTestAttr.attrib;
  %SMIL.skip-content.attrib;
">

<!ENTITY % SMIL.customAttributes.attrib  "%SMIL.Test.attrib; %SMIL.skip-
content.attrib;">
<!ENTITY % SMIL.customTest.attrib    "%SMIL.skip-content.attrib;">

<!-- ================== Animation ========================================== -->
<!ENTITY % SMIL.BasicAnimation.module "INCLUDE">

<!-- choose targetElement or XLink: -->
<!ENTITY % SMIL.animation-targetElement "INCLUDE">
<!ENTITY % SMIL.animation-XLinkTarget   "IGNORE">

<!ENTITY % SMIL.animate.content "EMPTY">
<!ENTITY % SMIL.animateColor.content "EMPTY">
<!ENTITY % SMIL.animateMotion.content "EMPTY">
<!ENTITY % SMIL.set.content "EMPTY">

<!ENTITY % SMIL.animate.attrib        "%SMIL.skip-content.attrib;
%SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.animateColor.attrib   "%SMIL.skip-content.attrib;
%SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.animateMotion.attrib  "%SMIL.skip-content.attrib;
%SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.set.attrib            "%SMIL.skip-content.attrib;
%SMIL.customTestAttr.attrib;">

<!-- ================== Layout ============================================= -->
<!ENTITY % SMIL.BasicLayout.module        "INCLUDE">
<!ENTITY % SMIL.AudioLayout.module        "INCLUDE">
<!ENTITY % SMIL.MultiWindowLayout.module  "INCLUDE">
<!ENTITY % SMIL.HierarchicalLayout.module "INCLUDE">

<!ENTITY % SMIL.layout.content "(%SMIL.region.qname;
                              | %SMIL.topLayout.qname;
                        | %SMIL.root-layout.qname;
                        | %SMIL.regPoint.qname;)*">
<!ENTITY % SMIL.region.content "(%SMIL.region.qname;)*">
<!ENTITY % SMIL.topLayout.content "(%SMIL.region.qname;)*">
<!ENTITY % SMIL.rootlayout.content "EMPTY">
<!ENTITY % SMIL.regPoint.content "EMPTY">

<!ENTITY % SMIL.layout.attrib          "%SMIL.Test.attrib;
%SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.rootlayout.attrib      "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.topLayout.attrib       "%SMIL.content-control-attrs;">
```

```
<!ENTITY % SMIL.region.attrib          "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.regPoint.attrib        "%SMIL.content-control-attrs;">

<!-- ================== Linking ============================================ -->
<!ENTITY % SMIL.LinkingAttributes.module "INCLUDE">
<!ENTITY % SMIL.BasicLinking.module      "INCLUDE">
<!ENTITY % SMIL.ObjectLinking.module   "INCLUDE">

<!ENTITY % SMIL.a.content      "(%SMIL.timecontainer.class;|%SMIL.media-object;|
                                 %SMIL.content-control;)*">
<!ENTITY % SMIL.area.content   "(%SMIL.animate.qname;| %SMIL.set.qname;)*">
<!ENTITY % SMIL.anchor.content "(%SMIL.animate.qname; | %SMIL.set.qname;)*">

<!ENTITY % SMIL.a.attrib      "%SMIL.smil-basictime.attrib; %SMIL.Test.attrib;
%SMIL.customTestAttr.attrib;">
<!ENTITY % SMIL.area.attrib   "%SMIL.smil-basictime.attrib; %SMIL.content-
control-attrs;">
<!ENTITY % SMIL.anchor.attrib "%SMIL.smil-basictime.attrib; %SMIL.content-
control-attrs;">

<!-- ================== Media  ============================================ -->
<!ENTITY % SMIL.BasicMedia.module                     "INCLUDE">
<!ENTITY % SMIL.MediaClipping.module                  "INCLUDE">
<!ENTITY % SMIL.MediaClipping.deprecated.module       "INCLUDE">
<!ENTITY % SMIL.MediaClipMarkers.module               "INCLUDE">
<!ENTITY % SMIL.MediaParam.module                     "INCLUDE">
<!ENTITY % SMIL.BrushMedia.module                     "INCLUDE">
<!ENTITY % SMIL.MediaAccessibility.module             "INCLUDE">

<!ENTITY % SMIL.media-object.content "(%SMIL.animation.elements;
                                     | %SMIL.switch.qname;
                                     | %SMIL.anchor.qname;
                                     | %SMIL.area.qname;
                                     | %SMIL.param.qname;)*">
<!ENTITY % SMIL.media-object.attrib "
  %SMIL.BasicInlineTiming.attrib;
  %SMIL.BasicInlineTiming-deprecated.attrib;
  %SMIL.MinMaxTiming.attrib;
  %SMIL.RestartTiming.attrib;
  %SMIL.RestartDefaultTiming.attrib;
  %SMIL.SyncBehavior.attrib;
  %SMIL.SyncBehaviorDefault.attrib;
  %SMIL.endsync.media.attrib;
  %SMIL.fill.attrib;
  %SMIL.fillDefault.attrib;
  %SMIL.Test.attrib;
  %SMIL.customTestAttr.attrib;
  %SMIL.regionAttr.attrib;
  %SMIL.Transition.attrib;
  %SMIL.backgroundColor.attrib;
  %SMIL.backgroundColor-deprecated.attrib;
  %SMIL.Sub-region.attrib;
  %SMIL.RegistrationPoint.attrib;
  %SMIL.fit.attrib;
  %SMIL.tabindex.attrib;
">
```

```
<!ENTITY % SMIL.brush.attrib        "%SMIL.skip-content.attrib;">
<!ENTITY % SMIL.param.attrib        "%SMIL.content-control-attrs;">

<!-- ================== Metadata ========================================== -->
<!ENTITY % SMIL.meta.content     "EMPTY">
<!ENTITY % SMIL.meta.attrib      "%SMIL.skip-content.attrib;">

<!ENTITY % SMIL.metadata.content "EMPTY">
<!ENTITY % SMIL.metadata.attrib  "%SMIL.skip-content.attrib;">

<!-- ================== Structure ========================================= -->
<!ENTITY % SMIL.Structure.module "INCLUDE">
<!ENTITY % SMIL.smil.content "(%SMIL.head.qname;?,%SMIL.body.qname;?)">
<!ENTITY % SMIL.head.content "(
       %SMIL.meta.qname;*,
       ((%SMIL.head-control.content;),   %SMIL.meta.qname;*)?,
       ((%SMIL.head-meta.content;),      %SMIL.meta.qname;*)?,
       ((%SMIL.head-layout.content;),    %SMIL.meta.qname;*)?,
       ((%SMIL.head-transition.content;),%SMIL.meta.qname;*)?
)">
<!ENTITY % SMIL.body.content "(%SMIL.timecontainer.class;|%SMIL.media-object;|
                          %SMIL.content-control;|a)*">

<!ENTITY % SMIL.smil.attrib "%SMIL.Test.attrib;">
<!ENTITY % SMIL.body.attrib "
      %SMIL.timecontainer.attrib;
      %SMIL.Description.attrib;
      %SMIL.fill.attrib;
">

<!-- ================== Transitions ======================================= -->
<!ENTITY % SMIL.BasicTransitions.module        "INCLUDE">
<!ENTITY % SMIL.TransitionModifiers.module     "INCLUDE">
<!ENTITY % SMIL.InlineTransitions.module       "IGNORE">

<!ENTITY % SMIL.transition.content "EMPTY">
<!ENTITY % SMIL.transition.attrib "%SMIL.content-control-attrs;">

<!-- ================== Timing ============================================ -->
<!ENTITY % SMIL.BasicInlineTiming.module      "INCLUDE">
<!ENTITY % SMIL.SyncbaseTiming.module         "INCLUDE">
<!ENTITY % SMIL.EventTiming.module            "INCLUDE">
<!ENTITY % SMIL.WallclockTiming.module        "INCLUDE">
<!ENTITY % SMIL.MultiSyncArcTiming.module     "INCLUDE">
<!ENTITY % SMIL.MediaMarkerTiming.module      "INCLUDE">
<!ENTITY % SMIL.MinMaxTiming.module           "INCLUDE">
<!ENTITY % SMIL.BasicTimeContainers.module    "INCLUDE">
<!ENTITY % SMIL.ExclTimeContainers.module     "INCLUDE">
<!ENTITY % SMIL.PrevTiming.module             "INCLUDE">
<!ENTITY % SMIL.RestartTiming.module          "INCLUDE">
<!ENTITY % SMIL.SyncBehavior.module           "INCLUDE">
<!ENTITY % SMIL.SyncBehaviorDefault.module    "INCLUDE">
<!ENTITY % SMIL.RestartDefault.module         "INCLUDE">
<!ENTITY % SMIL.fillDefault.module            "INCLUDE">

<!ENTITY % SMIL.par.attrib "
      %SMIL.endsync.attrib;
      %SMIL.fill.attrib;
      %SMIL.timecontainer.attrib;
      %SMIL.Test.attrib;
      %SMIL.customTestAttr.attrib;
      %SMIL.regionAttr.attrib;
">
<!ENTITY % SMIL.seq.attrib "
      %SMIL.fill.attrib;
      %SMIL.timecontainer.attrib;
      %SMIL.Test.attrib;
      %SMIL.customTestAttr.attrib;
      %SMIL.regionAttr.attrib;
">
<!ENTITY % SMIL.excl.attrib "
      %SMIL.endsync.attrib;
      %SMIL.fill.attrib;
      %SMIL.timecontainer.attrib;
      %SMIL.Test.attrib;
      %SMIL.customTestAttr.attrib;
      %SMIL.regionAttr.attrib;
        %SMIL.skip-content.attrib;
">
<!ENTITY % SMIL.par.content "(%SMIL.timecontainer.content;)*">
<!ENTITY % SMIL.seq.content "(%SMIL.timecontainer.content;)*">
<!ENTITY % SMIL.excl.content "((%SMIL.timecontainer.content;)*
                              | %SMIL.priorityClass.qname;+)">

<!ENTITY % SMIL.priorityClass.attrib  "%SMIL.content-control-attrs;">
<!ENTITY % SMIL.priorityClass.content "(%SMIL.timecontainer.content;)*">

<!-- end of smil-model-1.mod -->
```

C. SMIL 2.0 Structure Module

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

     This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author: Warner ten Kate, Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Structure//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-struct.mod"

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

<!-- ================== SMIL Document Root =============================== -->
<!ENTITY % SMIL.smil.attrib  "" >
<!ENTITY % SMIL.smil.content "EMPTY" >
<!ENTITY % SMIL.smil.qname   "smil" >

<!ELEMENT %SMIL.smil.qname; %SMIL.smil.content;>
<!ATTLIST %SMIL.smil.qname; %SMIL.smil.attrib;
        %Core.attrib;
        %I18n.attrib;
        xmlns %URI.datatype; #REQUIRED
>

<!-- ================== The Document Head ================================ -->
<!ENTITY % SMIL.head.content "EMPTY" >
<!ENTITY % SMIL.head.attrib  "" >
<!ENTITY % SMIL.head.qname   "head" >

<!ELEMENT %SMIL.head.qname; %SMIL.head.content;>
<!ATTLIST %SMIL.head.qname; %SMIL.head.attrib;
        %Core.attrib;
        %I18n.attrib;
>

<!--=================== The Document Body - Timing Root ================== -->
<!ENTITY % SMIL.body.content "EMPTY" >
<!ENTITY % SMIL.body.attrib  "" >
<!ENTITY % SMIL.body.qname   "body" >

<!ELEMENT %SMIL.body.qname; %SMIL.body.content;>
<!ATTLIST %SMIL.body.qname; %SMIL.body.attrib;
        %Core.attrib;
        %I18n.attrib;
>
<!-- end of SMIL-struct.mod -->
```

D. SMIL 2.0 Animation Module

```
<!-- ====================================================================== -->
<!-- SMIL Animation Module  ================================================ -->
<!-- file: SMIL-anim.mod

  This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Animation//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-anim.mod"

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


<!-- ============================= Dependencies ============================ -->
<!-- The integrating profile is expected to define the following entities,
     Unless the defaults provided are sufficient.
 -->

<!-- SMIL.SplineAnimation.module entity:  Define as "INCLUDE" if the integrating
     profile includes the SMIL 2.0 SplineAnimation Module, "IGNORE" if not.
     The default is "IGNORE", i.e. by default SplineAnimation is not included
     in the integrating language profile.
 -->
<!ENTITY % SMIL.SplineAnimation.module "IGNORE">

<!-- Animation depends on SMIL Timing, importing the attributes listed
     in the SMIL.AnimationTime.attrib entity.  If the integrating profile does
     include the SMIL.MinMaxTiming.module, its default value includes the
     attributes defined in SMIL.BasicInlineTiming.attrib and
       SMIL.MinMaxTiming.attrib.  Otherwise, it is defaulted to
       SMIL.BasicInlineTiming.attrib, which is the minimum requirement.

     Note that the profile can override these defaults by redefining
     SMIL.AnimationTime.attrib.  The profile is also expected to define
     SMIL.fill.attrib and SMIL.fillDefault.attrib.
 -->
<!ENTITY % SMIL.MinMaxTiming.module "IGNORE">
<![%SMIL.MinMaxTiming.module;[
  <!ENTITY % SMIL.AnimationTime.attrib "
      %SMIL.BasicInlineTiming.attrib;
      %SMIL.BasicInlineTiming-deprecated.attrib;
      %SMIL.MinMaxTiming.attrib;
  ">
]]>
<!ENTITY % SMIL.AnimationTime.attrib "%SMIL.BasicInlineTiming.attrib;">
<!ENTITY % SMIL.fill.attrib "">
```

```
<!ENTITY % SMIL.animTimingAttrs "
  %SMIL.AnimationTime.attrib;
  %SMIL.fill.attrib;
  %SMIL.fillDefault.attrib;
">

<!-- Language Designer chooses to integrate targetElement or xlink attributes.
     To integrate the targetElement attribute, define the entity
     animation-targetElement as "INCLUDE"; to integrate the XLink attributes,
     define animation-XLinkTarget as "INCLUDE".

     One or the other MUST be defined.  It is strongly recommended that only one
     of the two be defined.
-->

<!ENTITY % SMIL.animation-targetElement "IGNORE">
<![%SMIL.animation-targetElement;[
  <!ENTITY % SMIL.animTargetElementAttr
   "targetElement  IDREF  #IMPLIED"
  >
]]>
<!ENTITY % SMIL.animTargetElementAttr "">

<!ENTITY % SMIL.animation-XLinkTarget "IGNORE">
<![%SMIL.animation-XLinkTarget;[
  <!ENTITY % SMIL.animTargetElementXLink "
    actuate        (onRequest|onLoad)                  'onLoad'
    href           %URI.datatype;                      #IMPLIED
    show           (new | embed | replace)             #FIXED 'embed'
    type           (simple | extended | locator | arc) #FIXED 'simple'
">
]]>
<!ENTITY % SMIL.animTargetElementXLink "">


<!-- ========================== Attribute Groups =========================== -->

<!-- All animation elements include these attributes -->
<!ENTITY % SMIL.animAttrsCommon
 "%Core.attrib;
  %I18n.attrib;
  %SMIL.Test.attrib;
  %SMIL.animTimingAttrs;
  %SMIL.animTargetElementAttr;
  %SMIL.animTargetElementXLink;"
>

<!-- All except animateMotion need an identified target attribute -->
<!ENTITY % SMIL.animAttrsNamedTarget
 "%SMIL.animAttrsCommon;
  attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED"
>

<!-- All except set support the full animation-function specification,
     additive and cumulative animation.
     SplineAnimation adds the attributes keyTimes, keySplines and path,
       and the calcMode value "spline", to those of BasicAnimation.
 -->
<![%SMIL.SplineAnimation.module;[
  <!ENTITY % SMIL.splineAnimCalcModeValues "| spline">
  <!ENTITY % SMIL.splineAnimValueAttrs
   "keyTimes        CDATA             #IMPLIED
   keySplines      CDATA             #IMPLIED"
  >
  <!ENTITY % SMIL.splineAnimPathAttr
   "path            CDATA             #IMPLIED"
  >
]]>
<!ENTITY % SMIL.splineAnimCalcModeValues "">
<!ENTITY % SMIL.splineAnimValueAttrs "">
<!ENTITY % SMIL.splineAnimPathAttr "">

<!ENTITY % SMIL.animValueAttrs "
   %SMIL.BasicAnimation.attrib;
   calcMode    (discrete|linear|paced %SMIL.splineAnimCalcModeValues;) 'linear'
   %SMIL.splineAnimValueAttrs;
   additive        (replace | sum)    'replace'
   accumulate      (none | sum)       'none'"
>


<!-- ========================== Animation Elements ========================= -->

<!ENTITY % SMIL.animate.attrib  "">
<!ENTITY % SMIL.animate.content "EMPTY">
<!ENTITY % SMIL.animate.qname   "animate">
<!ELEMENT %SMIL.animate.qname; %SMIL.animate.content;>
<!ATTLIST %SMIL.animate.qname; %SMIL.animate.attrib;
  %SMIL.animAttrsNamedTarget;
  %SMIL.animValueAttrs;
>

<!ENTITY % SMIL.set.attrib  "">
<!ENTITY % SMIL.set.content "EMPTY">
<!ENTITY % SMIL.set.qname   "set">
<!ELEMENT %SMIL.set.qname; %SMIL.set.content;>
<!ATTLIST %SMIL.set.qname; %SMIL.set.attrib;
      %SMIL.animAttrsNamedTarget;
      to   CDATA  #IMPLIED
>

<!ENTITY % SMIL.animateMotion.attrib  "">
<!ENTITY % SMIL.animateMotion.content "EMPTY">
<!ENTITY % SMIL.animateMotion.qname   "animateMotion">
<!ELEMENT %SMIL.animateMotion.qname; %SMIL.animateMotion.content;>
<!ATTLIST %SMIL.animateMotion.qname; %SMIL.animateMotion.attrib;
      %SMIL.animAttrsCommon;
      %SMIL.animValueAttrs;
      %SMIL.splineAnimPathAttr;
      origin  (default)  "default"
>
```

```
<!ENTITY % SMIL.animateColor.attrib  "">
<!ENTITY % SMIL.animateColor.content "EMPTY">
<!ENTITY % SMIL.animateColor.qname   "animateColor">
<!ELEMENT %SMIL.animateColor.qname; %SMIL.animateColor.content;>
<!ATTLIST %SMIL.animateColor.qname; %SMIL.animateColor.attrib;
  %SMIL.animAttrsNamedTarget;
  %SMIL.animValueAttrs;
>

<!-- ========================== End Animation ============================= -->
<!-- end of SMIL-anim.mod -->
```

E. SMIL 2.0 Content Control Module

```
<!-- ================================================================= -->
<!-- SMIL Content Control Module  ==================================== -->
<!-- file: SMIL-control.mod

      This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen, Aaron Cohen
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Content Control//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-control.mod"

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

<!ENTITY % SMIL.BasicContentControl.module "INCLUDE">
<![%SMIL.BasicContentControl.module;[
  <!ENTITY % SMIL.switch.attrib "">
  <!ENTITY % SMIL.switch.content "EMPTY">
  <!ENTITY % SMIL.switch.qname "switch">

  <!ELEMENT %SMIL.switch.qname; %SMIL.switch.content;>
  <!ATTLIST %SMIL.switch.qname; %SMIL.switch.attrib;
        %Core.attrib;
        %I18n.attrib;
  >
]]>

<!-- ========================= CustomTest Elements ========================= -->
<!ENTITY % SMIL.CustomTestAttributes.module "IGNORE">
<![%SMIL.CustomTestAttributes.module;[

  <!ENTITY % SMIL.customTest.attrib "">
  <!ENTITY % SMIL.customTest.qname "customTest">
  <!ENTITY % SMIL.customTest.content "EMPTY">
  <!ELEMENT %SMIL.customTest.qname; %SMIL.customTest.content;>
  <!ATTLIST %SMIL.customTest.qname; %SMIL.customTest.attrib;
      defaultState      (true|false)            'false'
      override          (visible|hidden)        'hidden'
      uid               %URI.datatype;          #IMPLIED
      %Core.attrib;
      %I18n.attrib;
  >
  <!ENTITY % SMIL.customAttributes.attrib "">
  <!ENTITY % SMIL.customAttributes.qname "customAttributes">
  <!ENTITY % SMIL.customAttributes.content "(customTest+)">
  <!ELEMENT %SMIL.customAttributes.qname; %SMIL.customAttributes.content;>
  <!ATTLIST %SMIL.customAttributes.qname; %SMIL.customAttributes.attrib;
        %Core.attrib;
        %I18n.attrib;
  >

]]> <!-- end of CustomTestAttributes -->

<!-- ========================= PrefetchControl Elements ==================== -->
<!ENTITY % SMIL.PrefetchControl.module "IGNORE">
<![%SMIL.PrefetchControl.module;[
  <!ENTITY % SMIL.prefetch.attrib "">
  <!ENTITY % SMIL.prefetch.qname "prefetch">
  <!ENTITY % SMIL.prefetch.content "EMPTY">
  <!ELEMENT %SMIL.prefetch.qname; %SMIL.prefetch.content;>
  <!ATTLIST %SMIL.prefetch.qname; %SMIL.prefetch.attrib;
      src            %URI.datatype;        #IMPLIED
      mediaSize      CDATA                 #IMPLIED
      mediaTime      CDATA                 #IMPLIED
      bandwidth      CDATA                 #IMPLIED
      %Core.attrib;
      %I18n.attrib;
  >
]]>

<!-- end of SMIL-control.mod -->
```

F. SMIL 2.0 Layout Module

```
<!-- ====================================================================== -->
<!-- SMIL 2.0 Layout Modules ============================================== -->
<!-- file: SMIL-layout.mod

        This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen, Aaron Cohen
        Revision:   2001/07/31  Thierry Michel

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

        PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Layout//EN"
        SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-layout.mod"

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

<!-- ================== BasicLayout ======================================== -->
<!-- ================== BasicLayout Profiling Entities ===================== -->
<!ENTITY % SMIL.layout.attrib       "">
<!ENTITY % SMIL.region.attrib       "">
<!ENTITY % SMIL.rootlayout.attrib   "">
<!ENTITY % SMIL.layout.content     "EMPTY">
<!ENTITY % SMIL.region.content     "EMPTY">
<!ENTITY % SMIL.rootlayout.content "EMPTY">

<!-- ================== BasicLayout Entities =============================== -->
<!ENTITY % SMIL.common-layout-attrs "
      height              CDATA    'auto'
      width               CDATA    'auto'
      %SMIL.backgroundColor.attrib;
">

<!ENTITY % SMIL.region-attrs "
      bottom              CDATA    'auto'
      left                CDATA    'auto'
      right               CDATA    'auto'
      top                 CDATA    'auto'
      z-index             CDATA    #IMPLIED
      showBackground      (always|whenActive) 'always'
      %SMIL.fit.attrib;
">

<!-- ================== BasicLayout Elements =============================== -->
<!--
     Layout contains the region and root-layout elements defined by
     smil-basic-layout or other elements defined by an external layout
     mechanism.
-->

<!ENTITY % SMIL.layout.qname "layout">
<!ELEMENT %SMIL.layout.qname; %SMIL.layout.content;>
```

```
<!ATTLIST %SMIL.layout.qname; %SMIL.layout.attrib;
      %Core.attrib;
      %I18n.attrib;
        type    CDATA     'text/smil-basic-layout'
>

<!-- ================== Region Element ======================================-->
<!ENTITY % SMIL.region.qname "region">
<!ELEMENT %SMIL.region.qname; %SMIL.region.content;>
<!ATTLIST %SMIL.region.qname; %SMIL.region.attrib;
      %Core.attrib;
      %I18n.attrib;
        %SMIL.backgroundColor-deprecated.attrib;
        %SMIL.common-layout-attrs;
        %SMIL.region-attrs;
        regionName     CDATA     #IMPLIED
>

<!-- ================== Root-layout Element ===================================-->
<!ENTITY % SMIL.root-layout.qname "root-layout">
<!ELEMENT %SMIL.root-layout.qname; %SMIL.rootlayout.content; >
<!ATTLIST %SMIL.root-layout.qname; %SMIL.rootlayout.attrib;
      %Core.attrib;
      %I18n.attrib;
        %SMIL.backgroundColor-deprecated.attrib;
        %SMIL.common-layout-attrs;
>


<!-- ================== AudioLayout ======================================== -->
<!ENTITY % SMIL.AudioLayout.module "IGNORE">
<![%SMIL.AudioLayout.module;[
  <!-- ================== AudioLayout Entities ============================== -->
  <!ENTITY % SMIL.audio-attrs "
        soundLevel                        CDATA    '100&#37;'
  ">

  <!-- ================ AudioLayout Elements =============================== -->
  <!-- ================ Add soundLevel to region element =================== -->
  <!ATTLIST %SMIL.region.qname; %SMIL.audio-attrs;>
]]> <!-- end AudioLayout.module -->


<!-- ================ MultiWindowLayout ==================================== -->
<!ENTITY % SMIL.MultiWindowLayout.module "IGNORE">
<![%SMIL.MultiWindowLayout.module;[
  <!-- ============== MultiWindowLayout Profiling Entities ================= -->
  <!ENTITY % SMIL.topLayout.attrib    "">
  <!ENTITY % SMIL.topLayout.content   "EMPTY">

  <!-- ============== MultiWindowLayout Elements =========================== -->
  <!--================== topLayout element ================================== -->
  <!ENTITY % SMIL.topLayout.qname "topLayout">
  <!ELEMENT %SMIL.topLayout.qname; %SMIL.topLayout.content;>
  <!ATTLIST %SMIL.topLayout.qname; %SMIL.topLayout.attrib;
      %Core.attrib;
      %I18n.attrib;
```

```
        %SMIL.common-layout-attrs;
        close                (onRequest|whenNotActive) 'onRequest'
        open                 (onStart|whenActive)      'onStart'
  >
]]> <!-- end MultiWindowLayout.module -->


<!-- ====================== HierarchicalLayout ============================== -->
<!ENTITY % SMIL.HierarchicalLayout.module "IGNORE">
<![%SMIL.HierarchicalLayout.module;[
  <!-- ========== HierarchicalLayout Profiling Entities ==================== -->
  <!ENTITY % SMIL.regPoint.attrib     "">
  <!ENTITY % SMIL.regPoint.content   "EMPTY">

  <!-- ============ HierarchicalLayout Elements ============================ -->
  <!ENTITY % SMIL.regPoint.qname "regPoint">
  <!ELEMENT %SMIL.regPoint.qname; %SMIL.regPoint.content;>
  <!ATTLIST %SMIL.regPoint.qname; %SMIL.regPoint.attrib;
      %Core.attrib;
      %I18n.attrib;
        %SMIL.regAlign.attrib;
        bottom               CDATA    'auto'
        left                 CDATA    'auto'
        right                CDATA    'auto'
        top                  CDATA    'auto'
  >
]]> <!-- end HierarchicalLayout.module -->


<!-- end of SMIL-layout.mod -->
```

G. SMIL 2.0 Linking Module

```
<!-- ====================================================================== -->
<!-- SMIL Linking Module  ================================================== -->
<!-- file: SMIL-link.mod

     This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen, Lloyd Rutledge, Aaron Cohen
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Linking//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-link.mod"

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

<!-- ======================== LinkingAttributes Entities =================== -->
<!ENTITY % SMIL.linking-attrs "
      sourceLevel               CDATA               '100&#37;'
      destinationLevel          CDATA               '100&#37;'
      sourcePlaystate           (play|pause|stop)   #IMPLIED
      destinationPlaystate      (play|pause|stop)   'play'
      show                      (new|pause|replace) 'replace'
      accesskey                 %Character.datatype; #IMPLIED
      target                    CDATA                #IMPLIED
      external                  (true|false)        'false'
      actuate                   (onRequest|onLoad)  'onRequest'
      %SMIL.tabindex.attrib;
">



<!-- ========================= BasicLinking Elements ======================= -->
<!ENTITY % SMIL.BasicLinking.module "IGNORE">
<![%SMIL.BasicLinking.module;[

  <!-- ======================= BasicLinking Entities ======================= -->
  <!ENTITY % SMIL.Shape "(rect|circle|poly|default)">
  <!ENTITY % SMIL.Coords "CDATA">
    <!-- comma separated list of lengths -->

  <!ENTITY % SMIL.a.attrib  "">
  <!ENTITY % SMIL.a.content "EMPTY">
  <!ENTITY % SMIL.a.qname   "a">
  <!ELEMENT %SMIL.a.qname; %SMIL.a.content;>
  <!ATTLIST %SMIL.a.qname; %SMIL.a.attrib;
    %SMIL.linking-attrs;
    href                    %URI.datatype;      #IMPLIED
    %Core.attrib;
    %I18n.attrib;
  >
  <!ENTITY % SMIL.area.attrib  "">
  <!ENTITY % SMIL.area.content "EMPTY">
  <!ENTITY % SMIL.area.qname   "area">
  <!ELEMENT %SMIL.area.qname; %SMIL.area.content;>
  <!ATTLIST %SMIL.area.qname; %SMIL.area.attrib;
    %SMIL.linking-attrs;
    shape                      %SMIL.Shape;            'rect'
    coords                     %SMIL.Coords;           #IMPLIED
    href                       %URI.datatype;          #IMPLIED
    nohref                     (nohref)                #IMPLIED
    %Core.attrib;
    %I18n.attrib;
  >

  <!ENTITY % SMIL.anchor.attrib  "">
  <!ENTITY % SMIL.anchor.content "EMPTY">
  <!ENTITY % SMIL.anchor.qname   "anchor">
  <!ELEMENT %SMIL.anchor.qname; %SMIL.anchor.content;>
  <!ATTLIST %SMIL.anchor.qname; %SMIL.anchor.attrib;
    %SMIL.linking-attrs;
    shape                      %SMIL.Shape;            'rect'
    coords                     %SMIL.Coords;           #IMPLIED
    href                       %URI.datatype;          #IMPLIED
    nohref                     (nohref)                #IMPLIED
    %Core.attrib;
    %I18n.attrib;
  >
]]> <!-- end of BasicLinking -->

<!-- ======================== ObjectLinking ================================ -->
<!ENTITY % SMIL.ObjectLinking.module "IGNORE">
<![%SMIL.ObjectLinking.module;[

  <!ENTITY % SMIL.Fragment "
    fragment                   CDATA               #IMPLIED
  ">

  <!-- ====================== ObjectLinking Elements ======================= -->
  <!-- add fragment attribute to area, and anchor elements -->
  <!ATTLIST %SMIL.area.qname;
      %SMIL.Fragment;
  >

  <!ATTLIST %SMIL.anchor.qname;
      %SMIL.Fragment;
  >
]]>
<!-- ======================== End ObjectLinking ============================ -->

<!-- end of SMIL-link.mod -->
```

H. SMIL 2.0 Media Object Module

```
<!-- ====================================================================== -->
<!-- SMIL 2.0 Media Objects Modules ======================================== -->
<!-- file: SMIL-media.mod

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Rob Lanphier, Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:
     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Media Objects//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-media.mod"
     ====================================================================== -->

<!-- ================== Profiling Entities ================================== -->
<!ENTITY % SMIL.MediaClipping.module "IGNORE">
<![%SMIL.MediaClipping.module;[
  <!ENTITY % SMIL.mo-attributes-MediaClipping "
      %SMIL.MediaClip.attrib;
  ">
]]>
<!ENTITY % SMIL.mo-attributes-MediaClipping "">

<!ENTITY % SMIL.MediaClipping.deprecated.module "IGNORE">
<![%SMIL.MediaClipping.module;[
  <!ENTITY % SMIL.mo-attributes-MediaClipping-deprecated "
      %SMIL.MediaClip.attrib.deprecated;
  ">
  ]]>
<!ENTITY % SMIL.mo-attributes-MediaClipping-deprecated "">

<!ENTITY % SMIL.MediaParam.module "IGNORE">
<![%SMIL.MediaParam.module;[
  <!ENTITY % SMIL.mo-attributes-MediaParam "
        erase         (whenDone|never)      'whenDone'
        mediaRepeat   (preserve|strip)      'preserve'
  ">
  <!ENTITY % SMIL.param.qname "param">
  <!ELEMENT %SMIL.param.qname; EMPTY>

  <!ATTLIST %SMIL.param.qname; %SMIL.param.attrib;
    %Core.attrib;
    %I18n.attrib;
    name        CDATA           #IMPLIED
    value       CDATA           #IMPLIED
    valuetype   (data|ref|object) "data"
    type        %ContentType.datatype;  #IMPLIED
  >
]]>
<!ENTITY % SMIL.mo-attributes-MediaParam "">

<!ENTITY % SMIL.MediaAccessibility.module "IGNORE">
<![%SMIL.MediaAccessibility.module;[
  <!ENTITY % SMIL.mo-attributes-MediaAccessibility "
        readIndex    CDATA           #IMPLIED
  ">
]]>
<!ENTITY % SMIL.mo-attributes-MediaAccessibility "">

<!ENTITY % SMIL.BasicMedia.module "INCLUDE">
<![%SMIL.BasicMedia.module;[
  <!ENTITY % SMIL.media-object.content "EMPTY">
  <!ENTITY % SMIL.media-object.attrib "">

  <!-- ================ Media Objects Entities ============================= -->
  <!ENTITY % SMIL.mo-attributes-BasicMedia "
        src             CDATA   #IMPLIED
        type            CDATA   #IMPLIED
  ">

  <!ENTITY % SMIL.mo-attributes "
        %Core.attrib;
        %I18n.attrib;
        %SMIL.Description.attrib;
        %SMIL.mo-attributes-BasicMedia;
        %SMIL.mo-attributes-MediaParam;
        %SMIL.mo-attributes-MediaAccessibility;
        %SMIL.media-object.attrib;
  ">

  <!--
     Most info is in the attributes, media objects are empty or
     have children defined at the language integration level:
  -->

  <!ENTITY % SMIL.mo-content "%SMIL.media-object.content;">

  <!-- ================ Media Objects Elements ============================= -->
  <!ENTITY % SMIL.ref.qname        "ref">
  <!ENTITY % SMIL.audio.qname      "audio">
  <!ENTITY % SMIL.img.qname        "img">
  <!ENTITY % SMIL.video.qname      "video">
  <!ENTITY % SMIL.text.qname       "text">
  <!ENTITY % SMIL.textstream.qname "textstream">
  <!ENTITY % SMIL.animation.qname  "animation">

  <!ENTITY % SMIL.ref.content        "%SMIL.mo-content;">
  <!ENTITY % SMIL.audio.content      "%SMIL.mo-content;">
  <!ENTITY % SMIL.img.content        "%SMIL.mo-content;">
  <!ENTITY % SMIL.video.content      "%SMIL.mo-content;">
  <!ENTITY % SMIL.text.content       "%SMIL.mo-content;">
  <!ENTITY % SMIL.textstream.content "%SMIL.mo-content;">
  <!ENTITY % SMIL.animation.content  "%SMIL.mo-content;">

  <!ELEMENT %SMIL.ref.qname;           %SMIL.ref.content;>
  <!ELEMENT %SMIL.audio.qname;         %SMIL.audio.content;>
  <!ELEMENT %SMIL.img.qname;           %SMIL.img.content;>
  <!ELEMENT %SMIL.video.qname;         %SMIL.video.content;>
  <!ELEMENT %SMIL.text.qname;          %SMIL.text.content;>
  <!ELEMENT %SMIL.textstream.qname;    %SMIL.textstream.content;>
  <!ELEMENT %SMIL.animation.qname;     %SMIL.animation.content;>

  <!ATTLIST %SMIL.img.qname;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.text.qname;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.ref.qname;
          %SMIL.mo-attributes-MediaClipping;
          %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.audio.qname;
          %SMIL.mo-attributes-MediaClipping;
          %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.video.qname;
          %SMIL.mo-attributes-MediaClipping;
          %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.textstream.qname;
          %SMIL.mo-attributes-MediaClipping;
          %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
  <!ATTLIST %SMIL.animation.qname;
          %SMIL.mo-attributes-MediaClipping;
          %SMIL.mo-attributes-MediaClipping-deprecated;
        %SMIL.mo-attributes;
  >
]]>
<!ENTITY % SMIL.mo-attributes-BasicMedia "">

<!-- BrushMedia -->
<!ENTITY % SMIL.BrushMedia.module "IGNORE">
<![%SMIL.BrushMedia.module;[
  <!ENTITY % SMIL.brush.attrib "">
  <!ENTITY % SMIL.brush.content "%SMIL.mo-content;">
  <!ENTITY % SMIL.brush.qname "brush">
  <!ELEMENT %SMIL.brush.qname; %SMIL.brush.content;>
  <!ATTLIST %SMIL.brush.qname; %SMIL.brush.attrib;
        %Core.attrib;
        %I18n.attrib;
        %SMIL.Description.attrib;
        %SMIL.mo-attributes-MediaAccessibility;
        %SMIL.mo-attributes-MediaParam;
        %SMIL.media-object.attrib;
        color        CDATA            #IMPLIED
  >
]]>
<!-- end of SMIL-media.mod -->
```

I. SMIL 2.0 Metainformation Module

```
<!-- ================================================================== -->
<!-- SMIL Metainformation Module  =================================== -->
<!-- file: SMIL-metainformation.mod

     This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author: Thierry Michel, Jacco van Ossenbruggen
        Revision:   2001/07/31  Thierry Michel

     This module declares the meta and metadata elements types and
     its attributes, used to provide declarative document metainformation.

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Document Metadata//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-metainformation.mod"

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


<!-- ================== Profiling Entities ========================== -->

<!ENTITY % SMIL.meta.content     "EMPTY">
<!ENTITY % SMIL.meta.attrib      "">
<!ENTITY % SMIL.meta.qname       "meta">

<!ENTITY % SMIL.metadata.content "EMPTY">
<!ENTITY % SMIL.metadata.attrib  "">
<!ENTITY % SMIL.metadata.qname   "metadata">

<!-- ================== meta element =============================== -->

<!ELEMENT %SMIL.meta.qname; %SMIL.meta.content;>
<!ATTLIST %SMIL.meta.qname; %SMIL.meta.attrib;
  %Core.attrib;
  %I18n.attrib;
  content CDATA  #REQUIRED
  name    CDATA  #REQUIRED
  >

<!-- ================== metadata element =========================== -->

<!ELEMENT %SMIL.metadata.qname; %SMIL.metadata.content;>
<!ATTLIST %SMIL.metadata.qname; %SMIL.metadata.attrib;
  %Core.attrib;
  %I18n.attrib;
>

<!-- end of SMIL-metadata.mod -->
```

J. SMIL 2.0 Timing and Synchronization Module

```
<!-- ================================================================= -->
<!-- SMIL Timing and Synchronization Modules ========================== -->
<!-- file: SMIL-timing.mod

     This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen.
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Timing//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-timing.mod"

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


<!-- ================== Timing Elements =============================== -->

<!ENTITY % SMIL.BasicTimeContainers.module "IGNORE">
<![%SMIL.BasicTimeContainers.module;[
  <!ENTITY % SMIL.par.content "EMPTY">
  <!ENTITY % SMIL.seq.content "EMPTY">
  <!ENTITY % SMIL.par.attrib  "">
  <!ENTITY % SMIL.seq.attrib  "">
  <!ENTITY % SMIL.seq.qname   "seq">
  <!ENTITY % SMIL.par.qname   "par">

  <!ELEMENT %SMIL.seq.qname; %SMIL.seq.content;>
  <!ATTLIST %SMIL.seq.qname; %SMIL.seq.attrib;
   %Core.attrib;
   %I18n.attrib;
   %SMIL.Description.attrib;
  >

  <!ELEMENT %SMIL.par.qname; %SMIL.par.content;>
  <!ATTLIST %SMIL.par.qname; %SMIL.par.attrib;
   %Core.attrib;
   %I18n.attrib;
   %SMIL.Description.attrib;
  >
]]>  <!-- End of BasicTimeContainers.module -->


<!ENTITY % SMIL.ExclTimeContainers.module "IGNORE">
<![%SMIL.ExclTimeContainers.module;[
  <!ENTITY % SMIL.excl.content          "EMPTY">
  <!ENTITY % SMIL.priorityClass.content "EMPTY">
  <!ENTITY % SMIL.excl.attrib           "">
  <!ENTITY % SMIL.priorityClass.attrib  "">
  <!ENTITY % SMIL.excl.qname            "excl">
  <!ENTITY % SMIL.priorityClass.qname   "priorityClass">

  <!ELEMENT %SMIL.excl.qname; %SMIL.excl.content;>
  <!ATTLIST %SMIL.excl.qname; %SMIL.excl.attrib;
   %Core.attrib;
   %I18n.attrib;
   %SMIL.Description.attrib;
  >

  <!ELEMENT %SMIL.priorityClass.qname; %SMIL.priorityClass.content;>
  <!ATTLIST %SMIL.priorityClass.qname; %SMIL.priorityClass.attrib;
    peers   (stop|pause|defer|never) "stop"
    higher      (stop|pause)             "pause"
    lower       (defer|never)            "defer"
    pauseDisplay (disable|hide|show )    "show"
    %SMIL.Description.attrib;
    %Core.attrib;
    %I18n.attrib;
  >
]]>  <!-- End of ExclTimeContainers.module -->

<!-- end of SMIL-timing.mod -->
```

K. SMIL 2.0 Transition Effects Module

```
<!-- ====================================================================== -->
<!-- SMIL Transition Module  ============================================== -->
<!-- file: SMIL-transition.mod

     This is SMIL 2.0.

      Copyright: 1998-2001 W3C (MIT, INRIA, Keio), All Rights Reserved.
      See http://www.w3.org/Consortium/Legal/.

      Author:     Jacco van Ossenbruggen.
        Revision:   2001/07/31  Thierry Michel

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

     PUBLIC "-//W3C//ELEMENTS SMIL 2.0 Transition//EN"
     SYSTEM "http://www.w3.org/2001/SMIL20/SMIL-transition.mod"

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

<!ENTITY % SMIL.TransitionModifiers.module "IGNORE">
<![%SMIL.TransitionModifiers.module;[
 <!ENTITY % SMIL.transition-modifiers-attrs '
    horzRepeat   CDATA                       "0"
    vertRepeat   CDATA                       "0"
    borderWidth  CDATA                       "0"
    borderColor  CDATA                       "black"
 '>
]]> <!-- End of TransitionModifiers.module -->
<!ENTITY % SMIL.transition-modifiers-attrs "">

<!ENTITY % SMIL.BasicTransitions.module "INCLUDE">
<![%SMIL.BasicTransitions.module;[

 <!ENTITY % SMIL.transition-types "(barWipe|boxWipe|fourBoxWipe|barnDoorWipe|
  diagonalWipe|bowTieWipe|miscDiagonalWipe|veeWipe|barnVeeWipe|zigZagWipe|
  barnZigZagWipe|irisWipe|triangleWipe|arrowHeadWipe|pentagonWipe|
 hexagonWipe|ellipseWipe|eyeWipe|roundRectWipe|starWipe|miscShapeWipe|clockWipe|
  pinWheelWipe|singleSweepWipe|fanWipe|doubleFanWipe|doubleSweepWipe|
  saloonDoorWipe|windshieldWipe|snakeWipe|spiralWipe|parallelSnakesWipe|
  boxSnakesWipe|waterfallWipe|pushWipe|slideWipe|fade)"
 >

 <!ENTITY % SMIL.transition-subtypes "(bottom
  |bottomCenter|bottomLeft|bottomLeftClockwise|bottomLeftCounterClockwise|
  bottomLeftDiagonal|bottomRight|bottomRightClockwise|
  bottomRightCounterClockwise|bottomRightDiagonal|centerRight|centerTop|
  circle|clockwiseBottom|clockwiseBottomRight|clockwiseLeft|clockwiseNine|
  clockwiseRight|clockwiseSix|clockwiseThree|clockwiseTop|clockwiseTopLeft|
  clockwiseTwelve|cornersIn|cornersOut|counterClockwiseBottomLeft|
  counterClockwiseTopRight|crossfade|diagonalBottomLeft|
  diagonalBottomLeftOpposite|diagonalTopLeft|diagonalTopLeftOpposite|
  diamond|doubleBarnDoor|doubleDiamond|down|fadeFromColor|fadeToColor|
  fanInHorizontal|fanInVertical|fanOutHorizontal|fanOutVertical|fivePoint|
  fourBlade|fourBoxHorizontal|fourBoxVertical|fourPoint|fromBottom|fromLeft|
  fromRight|fromTop|heart|horizontal|horizontalLeft|horizontalLeftSame|
  horizontalRight|horizontalRightSame|horizontalTopLeftOpposite|
  horizontalTopRightOpposite|keyhole|left|leftCenter|leftToRight|
  oppositeHorizontal|oppositeVertical|parallelDiagonal|
  parallelDiagonalBottomLeft|parallelDiagonalTopLeft|
  parallelVertical|rectangle|right|rightCenter|sixPoint|top|topCenter|
  topLeft|topLeftClockwise|topLeftCounterClockwise|topLeftDiagonal|
  topLeftHorizontal|topLeftVertical|topRight|topRightClockwise|
  topRightCounterClockwise|topRightDiagonal|topToBottom|twoBladeHorizontal|
  twoBladeVertical|twoBoxBottom|twoBoxLeft|twoBoxRight|twoBoxTop|up|
  vertical|verticalBottomLeftOpposite|verticalBottomSame|verticalLeft|
  verticalRight|verticalTopLeftOpposite|verticalTopSame)"
 >

 <!ENTITY  % SMIL.transition-attrs '
    type          %SMIL.transition-types;     #IMPLIED
    subtype       %SMIL.transition-subtypes;  #IMPLIED
    fadeColor       CDATA                    "black"
    %SMIL.transition-modifiers-attrs;
 '>

 <!ENTITY % SMIL.transition.attrib  "">
 <!ENTITY % SMIL.transition.content "EMPTY">
 <!ENTITY % SMIL.transition.qname   "transition">
 <!ELEMENT %SMIL.transition.qname; %SMIL.transition.content;>
 <!ATTLIST %SMIL.transition.qname; %SMIL.transition.attrib;
    %Core.attrib;
    %I18n.attrib;
    %SMIL.transition-attrs;
    dur            %TimeValue.datatype; #IMPLIED
    startProgress CDATA                "0.0"
    endProgress   CDATA                "1.0"
    direction     (forward|reverse)   "forward"
 >
]]> <!-- End of BasicTransitions.module -->

<!ENTITY % SMIL.InlineTransitions.module "IGNORE">
<![%SMIL.InlineTransitions.module;[

 <!ENTITY % SMIL.transitionFilter.attrib  "">
 <!ENTITY % SMIL.transitionFilter.content "EMPTY">
 <!ENTITY % SMIL.transitionFilter.qname   "transitionFilter">
 <!ELEMENT %SMIL.transitionFilter.qname; %SMIL.transitionFilter.content;>
 <!ATTLIST %SMIL.transitionFilter.qname; %SMIL.transitionFilter.attrib;
    %Core.attrib;
    %I18n.attrib;
    %SMIL.transition-attrs;
    %SMIL.BasicInlineTiming.attrib;
    %SMIL.BasicAnimation.attrib;
    calcMode    (discrete|linear|paced) 'linear'
 >
]]> <!-- End of InlineTransitions.module -->

<!-- end of SMIL-transition.mod -->
```

SMIL Animation

W3C Recommendation 04-September-2001

This version:
http://www.w3.org/TR/2001/REC-smil-animation-20010904/
Latest version:
http://www.w3.org/TR/smil-animation
Previous version:
http://www.w3.org/TR/2001/PR-smil-animation-20010719/

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

Abstract

This is a W3C Recommendation of a specification of animation functionality for XML documents. It describes an animation framework as well as a set of base XML animation elements suitable for integration with XML documents. It is based upon the SMIL 1.0 timing model, with some extensions, and is a true subset of SMIL 2.0. This provides an intermediate stepping stone in terms of implementation complexity, for applications that wish to have SMIL-compatible animation but do not need or want time containers.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The SMIL Animation specification has been produced as part of the W3C Synchronized Multimedia Activity and was written by the SYMM Working Group (*members only*) of the W3C Interaction Domain, working with the SVG Working Group (*members only*) of the

W3C Document Formats Domain. The goals of the SYMM Working Group are discussed in the SYMM Working Group charter (*members only*), (revised July 2000 from original charter version).

The SYMM Working Group (*members only*) considers that all features in the SMIL 2.0 specification have been implemented at least twice in an interoperable way. The SYMM Working Group Charter (*members only*) defines this as the implementations having been developed independently by different organizations and each test in the SMIL 2.0 test suite has at least two passing implementations. The Implementation results are publicly released and are intended solely to be used as proof of SMIL 2.0 implementability. It is only a snap shot of the actual implementation behaviors at one moment of time, as these implementations may not be immediately available to the public. The interoperability data is not intended to be used for assessing or grading the performance of any individual implementation.

There are patent disclosures and license commitments associated with the SMIL 2.0 specification (and thus with the SMIL Animation specification also), these may be found on the SYMM Patent Statement page in conformance with W3C policy.

Please report errors in this document to www-smil@w3.org. The list of known errors in this specification is available at http://www.w3.org/2001/09/REC-smil-animation-20010904-errata.

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

Quick Table of Contents

Full Table of Contents

---

1. Introduction

This document describes a framework for incorporating animation onto a time line and a mechanism for composing the effects of multiple animations. A set of basic animation elements are also described that can be applied to any [XML]-based language. A language with which this module is integrated is referred to as a *host language*. A document containing animation elements is referred to as a *host document*.

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

The timing model is based upon SMIL 1.0 [SMIL1.0], with some changes and extensions to support additional timing features. SMIL Animation uses a simplified "flat" timing model, with no time containers (like `<par>` or `<seq>`). This version of SMIL Animation may not be used with documents that otherwise contain timing. See also Required definitions and constraints on element timing.

While this document defines a base set of animation capabilities, it is assumed that host languages may build upon the support to define additional or more specialized animation elements. In order to ensure a consistent model for document authors and runtime implementers, we introduce a framework for integrating animation with the SMIL timing model. Animation only manipulates attributes and properties of the target elements, and so does not require any specific knowledge of the target element semantics.

The examples in this document that include syntax for a host language use SMIL, SVG, XHTML and CSS. These are provided as an indication of possible integrations with various host languages.

2. Overview and terminology

2.1. Basics of animation

Animation is defined as a time-based manipulation of a *target element* (or more specifically of some *attribute* of the target element, the *target attribute*). The animation defines a mapping of time to values for the target attribute. This mapping accounts for all aspects of timing, as well as animation-specific semantics.

Animations specify a begin, and a *simple duration* that can be repeated. Each animation defines an *animation function* that produces a value for the target attribute, for any time within the simple duration. The author can specify how long or how many times an animation function should repeat. The simple duration combined with any repeating behavior defines the *active duration*.

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

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

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

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

When an animation is running, it should not actually change the attribute values in the DOM [DOM-Level-2]. The animation runtime should maintain a *presentation value* for each animated attribute, separate from the DOM or CSS Object Model (OM). If an implementation does not support an object model, it should maintain the original value as defined by the document as well as the presentation value. The presentation value is reflected in the display form of the document. Animations thus manipulate the presentation value, and should not affect the *base value* exposed by DOM or CSS OM. This is detailed in The animation sandwich model.

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

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

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

2.2. Animation function values

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

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

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

2.3. Symbols used in the semantic descriptions

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

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

3. Animation model

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

The time model depends upon several definitions for the host document: A host document is presented over a certain time interval. The start of the interval in which the document is presented is referred to as the *document begin*. The end of the interval in which the document is presented is referred to as the *document end*. The difference between the end and the begin is referred to as the *document duration*. The formal definitions of presentation and document begin and end are left to the host language designer (see also Required host language definitions).

3.1. Specifying the animation target

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

The target attribute

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

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

The `attributeType` attribute is optional. By default, the animation runtime will resolve the names according to the following rule: If there is a name conflict and `attributeType` is not specified, the list of CSS properties supported by the host language is matched first (if CSS is supported in the host language); if no CSS match is made (or CSS does not apply) the default namespace for the target element will be matched.

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

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

attributeName = <attributeName>

Specifies the name of the target attribute. An XMLNS prefix may be used to indicate the XML namespace for the attribute. The prefix will be interpreted in the scope of the animation element.

attributeType = "CSS | XML | auto"
Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following (values are case-sensitive):

"CSS"
This specifies that the value of "attributeName" is the name of a CSS property, as defined for the host document. This argument value is only meaningful in host language environments that support CSS.

"XML"
This specifies that the value of "attributeName" is the name of an XML attribute defined in the default XML namespace for the target element. Note that if the value for `attributeName` has an XMLNS prefix, the implementation must use the associated namespace as defined in the scope of the animation element.

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

The target element

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

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

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

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

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

`href` = *uri-reference*
This attribute specifies an XLink locator, referring to the target element to be animated.

When integrating animation elements into the host language, the language designer should avoid including both of these attributes. If however, the host language designer chooses to include both attributes in the host language, then when both are specified for a given animation element the XLink `href` attribute takes precedence over the `targetElement` attribute.

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

This example uses the simpler `targetElement` syntax:

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

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

```
<foo xmlns:xlink="http://www.w3.org/1999/xlink">
   ...
   <animate xlink:href="#foo" attributeName="bar" .../>
   ...
</foo>
```

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

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

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

`actuate` = 'onLoad'
: Indicates that the link to the target element is followed automatically (i.e., without user action).

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

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

3.2. Specifying the animation function f(t)

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

3.2.1. Animation function timing

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

This section is normative

begin

Defines when the element becomes active.
The attribute value is a semi-colon separated list of values.

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

Specifies the presentation time at which the animation begins. The begin is defined relative to the document begin.

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 animation 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 ")"

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.

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 animation will be determined by a "beginElement()" method call or a hyperlink targeted to the animation element.
The SMIL Animation DOM methods are described in the Supported methods section.
Hyperlink-based timing is described in the Hyperlinks and timing section.

Begin value semantics

This section is normative

- If no begin is specified, the default timing is equivalent to an offset value of "0".
- 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 not be processed normally. If no legal value is specified for a begin or end attribute, the element assumes an "indefinite" begin or end time (respectively).

This section is informative

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

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.

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

- The computed offset relative to the document 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 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 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, 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:

```
<animate begin="foo.click-8s" dur="3s" repeatCount="10"
.../>
```

The animation begins when the user 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 click. 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.

dur

Specifies the simple duration.
The attribute value can be one of the following types of values:

Clock-value

Specifies the length of the simple duration in presentation time.
Value must be greater than 0.

"indefinite"

Specifies the simple duration as indefinite.

If no `begin` is specified, the default value is "0" - the animation begins when the document begins. If there is any error in the argument value syntax for `begin`, the default value for `begin` will be used.

If the animation does not have a `dur` attribute, the simple duration is indefinite. Note that interpolation will not work if the simple duration is indefinite (although this may still be useful for <set> elements). See also Interpolation and indefinite simple durations.

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

If the begin is specified to be "indefinite" or specifies an event-base, the time of the begin is not actually known until the element is activated (e.g., with a hyperlink, DOM method call or the referenced event). The time is referred to as *unresolved* when it is not known. At the point at which the element begin is activated, the time becomes *resolved*. This is described in detail in Unifying event-based and scheduled timing.

Examples

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

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

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

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

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

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

3.2.2. Animation function values

In addition to the target attribute and the timing, an animation must specify how to change the value over time. An animation can be described either as a list of *values*, or in a simplified form using *from*, *to* and *by* values.

from = "<value>"
Specifies the starting value of the animation.
to = "<value>"
Specifies the ending value of the animation.
by = "<value>"
Specifies a relative offset value for the animation.

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

If a list of values is used, the animation will apply the values in order over the course of the animation (pacing and interpolation between these values is described in the next section). If a list of *values* is specified, any *from*, *to* and *by* attribute values are ignored.

The simpler *from/to/by* syntax provides for several variants. To use one of these variants, one of `by` or `to` must be specified; a `from` value is optional. It is not legal to specify both `by` and `to` attributes - if both are specified, only the `to` attribute will be used (the `by` will be ignored). The combinations of attributes yield the following classes of animation:

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

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

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

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

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

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

Leading and trailing white space, and white space before and after semicolon separators, will be ignored.

If any values (i.e., the argument-values for `from`, `to`, `by` or `values` attributes) are not legal, the animation will have no effect (see also Handling Syntax Errors). Similarly, if none of the `from`, `to`, `by` or `values` attributes are specified, the animation will have no effect.

Interpolation and indefinite simple durations

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

Examples

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

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

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

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

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

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

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

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

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

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

3.2.3. Animation function calculation modes

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

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

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

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

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

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

`keyTimes` = "<list>"
: A semicolon-separated list of time values used to control the pacing of the animation. Each time in the list corresponds to a value in the `values` attribute list, and defines when the value should be used in the animation function. Each time value in the `keyTimes` list is specified as a floating point value between 0 and 1 (inclusive), representing a proportional offset into the simple duration of the animation element.

If a list of `keyTimes` is specified, there must be exactly as many values in the `keyTimes` list as in the `values` list.

Each successive time value must be greater than or equal to the preceding time value.

The `keyTimes` list semantics depends upon the interpolation mode:

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

If the interpolation mode is "paced", the `keyTimes` attribute is ignored.

If there are any errors in the `keyTimes` specification (bad values, too many or too few values), the animation will have no effect.

If the simple duration is indefinite, any `keyTimes` specification will be ignored.

`keySplines` = "<list>"

A set of Bezier control points associated with the `keyTimes` list, defining a cubic Bezier function that controls interval pacing. The attribute value is a semicolon separated list of control point descriptions. Each control point description is a set of four floating point values: `x1 y1 x2 y2`, describing the Bezier control points for one time segment. The `keyTimes` values that define the associated segment are the Bezier "anchor points", and the `keySplines` values are the control points. Thus, there must be one fewer sets of control points than there are `keyTimes`.

The values must all be in the range 0 to 1.

This attribute is ignored unless the `calcMode` is set to "spline".

If there are any errors in the `keySplines` specification (bad values, too many or too few values), the animation will have no effect.

If `calcMode` is set to "discrete", "linear" or "spline", but the `keyTimes` attribute is not specified, the values in the `values` attribute are assumed to be equally spaced through the animation duration, according to the `calcMode`:

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

This semantic applies as well when the `keySplines` attribute is specified, but `keyTimes` is not. The times associated to the `keySplines` values are determined as described above.

The syntax for the control point sets in `keySplines` lists is:

```
control-pt-set ::= ( fpval comma-wsp fpval comma-wsp
fpval comma-wsp fpval )
fpval          ::= Floating point number
comma-wsp      ::= S (spacechar|",") S
```

Control point values are separated by at least one white space character or a comma. Additional white space around the separator is allowed. The allowed syntax for floating point numbers must be defined in the host language.

For the shorthand forms *from-to animation* and *from-by animation*, there are only 2 values. A discrete *from-to animation* will set the "from" value for the first half of the simple duration and the "to" value for the second half of the simple duration. Similarly, a discrete *from-by animation* will set the "from" value for the first half of the simple duration and for the second half of the simple duration will set the computed result of applying the "by" value. For the shorthand form *to animation*, there is only 1 value; a discrete *to animation* will simply set the "to" value for the simple duration.

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

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

Interpolation modes illustrated

The three illustrations 1a, 1b and 1c below show how the same basic animation will change a value over time, given different interpolation modes. All examples use the default timing (no `keyTimes` or `keySplines` specified). All examples are based upon the following example, but with different values for `calcMode`:

```
<animate dur="30s" values="0; 1; 2; 4; 8; 15"
calcMode="[as specified]" />
```

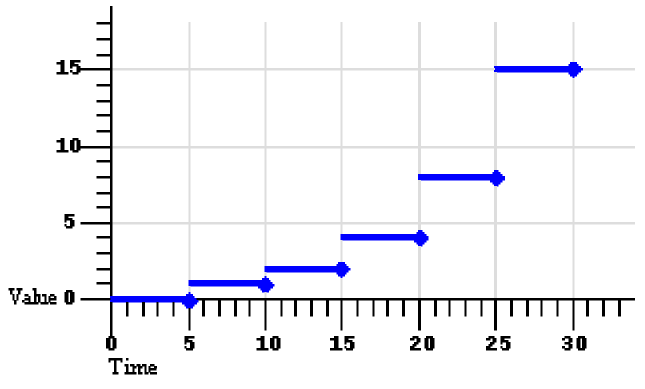


Figure 1a: Default discrete animation.

`calcMode="discrete"`

There are 6 segments of equal duration: 1 segment per value.

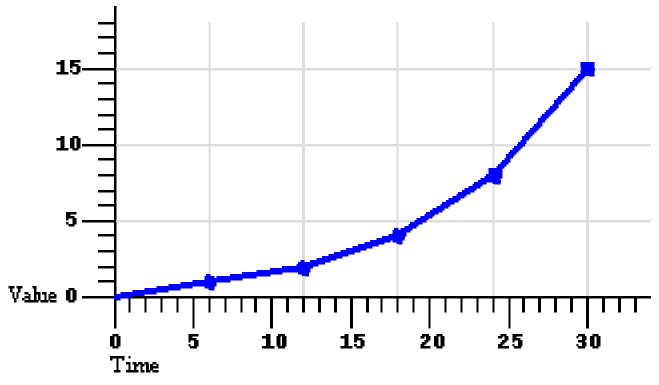


Figure 1b: Default linear animation.

`calcMode="linear"`

There are 5 segments of equal duration: n-1 segments for n values. Spline interpolation is a refinement of linear, and is further illustrated in Figure 2, below.

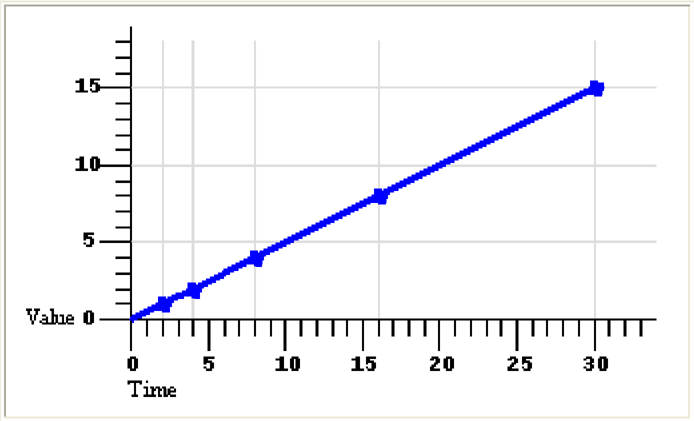


Figure 1c: Default paced animation.

`calcMode="paced"`

There are 5 segments of varying duration: n-1 segments for n values, computed to yield a constant rate of change in the value.

Examples

The following example describes a simple discrete animation:

```
<animate attributeName="foo" dur="8s"
     values="bar; fun; far; boo" />
```

The value of the attribute "foo" will be set to each of the four strings for 2 seconds each. Because the string values cannot be interpolated, only *discrete* animation is possible; any `calcMode` attribute would be ignored.

Discrete animation can also be used with `keyTimes`, as in the following example:

```
<animateColor attributeName="color" dur="10s"
calcMode="discrete"
     values="green; yellow; red" keyTimes="0.0; 0.5;" />
```

This example also shows how `keyTimes` values can interact with an indefinite duration. The value of the "color" attribute will be set to green for 5 seconds, and then to yellow for 5 seconds, and then will remain red for the remainder of the document, since the (unspecified) duration defaults to "indefinite".

The following example describes a simple linear animation:

```
<animate attributeName="x" dur="10s" values="0; 10; 100"
     calcMode="linear"/>
```

The value of "x" will change from 0 to 10 in the first 5 seconds, and then from 10 to 100 in the second 5 seconds. Note that the values in the `values` attribute are spaced evenly in time with no `keyTimes` specified; in this case the result is a much larger actual change in the value during the second half of the animation. Contrast this with the same example changed to use "paced" interpolation:

```
<animate attributeName="x" dur="10s" values="0; 10; 100"
     calcMode="paced"/>
```

To produce an even pace of change to the attribute "x", the second segment defined by the values list gets most of the simple duration: The value of "x" will change from 0 to 10 in the first second, and then from 10 to 100 in the next 9 seconds. While this example could be easily authored as a *from-to* animation without paced interpolation, many examples (such as motion paths) are much harder to author without the "paced" value for `calcMode`.

The following example illustrates the use of `keyTimes`:

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

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

Extending this example to use `keySplines`:

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

The `keyTimes` still cause the "x" attribute to have a value of "0" at the start of the animation, "50" after 8 seconds and "100" at the end of the animation. However, the `keySplines` values define a curve for pacing the interpolation between values. In the example above, the spline causes an ease-in and ease-out effect between time 0 and 8 seconds (i.e., between `keyTimes` 0 and .8, and `values` "0" and "50"), but a strict linear interpolation between 8 seconds and the end (i.e., between `keyTimes` .8 and 1, and `values` "50" and "100"). See Figure 2 below for an illustration of the curves that these `keySplines` values define.

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

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

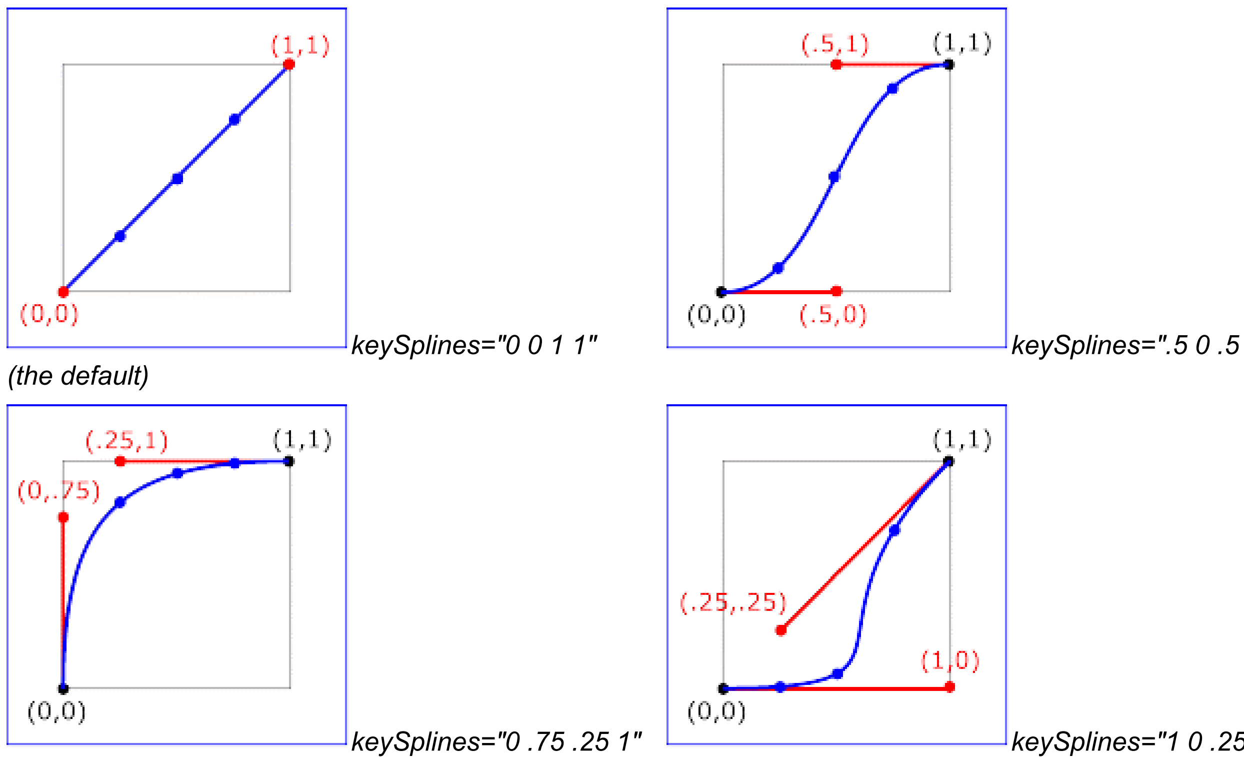


Figure 2: Illustration of keySplines effect

To illustrate the calculations, consider the simple example:

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

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

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

For a formal definition of Bezier spline calculation, see [COMP-GRAPHICS].

The `keyTimes` and `keySplines` attributes can also be used with the *from/to/by* shorthand forms for specifying values, as in the following example:

```
<animate attributeName="foo" from="10" to="20"
     dur="10s" keyTimes="0.0; 0.7"
     calcMode="spline" keySplines=".5 0 .5 1" />
```

The value will change from 10 to 20, using an "*ease-in/ease-out*" curve specified by the keySplines values. The `keyTimes` values cause the value of 20 to be reached at 7 seconds, and to hold there for the remainder of the 10 second simple duration.

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

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

3.3. Specifying the animation effect F(t)

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

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

The period of time during which the animation is actively playing, including any repeat behavior, is described as the active duration. The active duration may be computed from the simple duration and the repeat specification, and it may be constrained with the **`end`** attribute. The complete rules for computing the active duration are presented in the section Computing the active duration.

3.3.1. Repeating animations

Repeating an animation causes the animation function **`f(t)`** to be "played" several times in sequence. The author can specify either *how many times* to repeat, using `repeatCount`, or *how long* to repeat, using `repeatDur`. Each repeat *iteration* is one instance of "playing" the animation function **`f(t)`**.

If the simple duration is indefinite, the animation cannot repeat. See also the section Computing the active duration.

repeatCount
: Specifies the number of iterations of the animation function. 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 animation is defined to repeat indefinitely (i.e., until the document ends).

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

  Clock-value
  : Specifies the duration in presentation time to repeat the animation function **`f(t)`**.

  "indefinite"
  : The animation is defined to repeat indefinitely (i.e., until the document ends).

At most one of `repeatCount` or `repeatDur` should be specified. If both are specified (and the simple duration is not indefinite), the active duration is defined as the minimum of the specified repeatDur and the simple duration multiplied by repeatCount. For the purposes of this comparison, a defined value is considered to be "less than" a value of "indefinite". If the simple duration is indefinite, and both `repeatCount` and `repeatDur` are specified, the `repeatCount` will be ignored, and the `repeatDur` will be used (refer to the examples below describing `repeatDur` and an indefinite simple duration). These rules are included in the section Computing the active duration.

Examples

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

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

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

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

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

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

Note that if the simple duration is not defined (e.g. it is indefinite), repeat behavior is not defined (but `repeatDur` still defines the active duration). In the following example the simple duration is indefinite, and so the `repeatCount` is effectively ignored. Nevertheless, this is not considered an error: the active duration is also indefinite. The effect of the animation is to to just use the value for **`f(0)`**, setting the fill color to red for the remainder of the document duration.

```
<animate attributeName="fill" from="red" to="blue"
repeatCount="2" />
```

In the following example, the simple duration is indefinite, but the `repeatDur` still determines the active duration. The effect of the animation is to set the fill color to red for 10 seconds.

```
<animate attributeName="fill" from="red" to="blue"
repeatDur="10s" />
```

In the following example, the simple duration is longer than the duration specified by `repeatDur`, and so the active duration will effectively cut short the simple duration. However, the animation function still interpolates using the specified simple duration. The effect of the animation is to interpolate the value of "top" from 10 to 17, over the course of 7 seconds.

```
<animate attributeName="top" from="10" to="20"
         dur="10s" repeatDur="7s" />
```

The `min` attribute and restart:

The `min` attribute does not prevent an element from restarting before the minimum active duration is reached.

Controlling behavior of repeating animation - Cumulative animation

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

Using the path notation for a simple arc (detailed in The animateMotion element), we describe this example as:

```
<img ...>
   <animateMotion path="m 0 0 c 30 50 70 50 100 0 z"
      dur="5s"  accumulate="sum" repeatCount="4" />
</img>
```

The image moves from the original position along the arc over the course of 5 seconds. As the animation repeats, it builds upon the previous value and begins the second arc where the first one ended, as illustrated in Figure 3, below. In this way, the image "bounces" across the screen. The same animation could be described as a complete path of 4 arcs, but in the general case the path description can get quite large and cumbersome to edit.

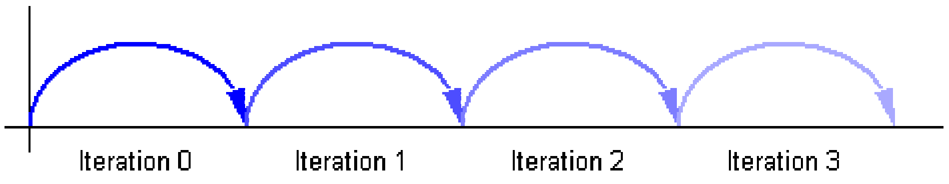


Figure 3: Illustration of repeating animation with `accumulate="sum"`. Each repeat iteration builds upon the previous.

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

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

accumulate = "none | sum"

Controls whether or not the animation is cumulative.
If `"sum"`, each repeat iteration after the first builds upon the last value of the previous iteration.
If `"none"`, repeat iterations are not cumulative, and simply repeat the animation function **`f(t)`**. This is the default.

This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.

Cumulative animation is not defined for "*to animation*". This attribute will be ignored if the animation function is specified with only the `to` attribute. See also Specifying function values.

Any numeric attribute that supports addition can support cumulative animation. For example, we can define a "pulsing" animation that will grow the "width" of an SVG `<rect>` element by 100 pixels in 50 seconds.

```
<rect width="20px"...>
   <animate attributeName="width" dur="5s"
      values="0; 15; 10" additive="sum"
      accumulate="sum" repeatCount="10" />
</rect>
```

Each simple duration causes the rectangle width to bulge by 15 pixels and end up 10 pixels larger. The shape is 20 pixels wide at the beginning, and after 5 seconds is 30 pixels wide. The animation repeats, and builds upon the previous values. The shape will bulge to 45 pixels and then end up 40 pixels wide after 10 seconds, and will eventually end up 120 (20 + 100) pixels wide after all 10 repeats.

From-to and *from-by* animations also support cumulative animation, as in the following example:

```
<rect width="20px"...>
   <animate attributeName="width" dur="5s" from="10px"
    to="20px" accumulate="sum" repeatCount="10" />
</rect>
```

The rectangle will grow from 10 to 20 pixels in the first 5 seconds, and then from 20 to 30 in the next 5 seconds, and so on up to 110 pixels after 10 repeats. Note that since the default value for `additive` is "replace", the original value is ignored. The following example makes the animation explicitly additive:

```
<rect width="20px"...>
   <animate attributeName="width" dur="5s" from="10px"
     to="20px"  accumulate="sum" additive="sum"
     repeatCount="10" />
</rect>
```

The results are the same as before, except that all the values are shifted up by the original value of 20. The rectangle is 30 pixels wide after 5 seconds, and 130 pixels wide after 10 repeats.

Computing cumulative animation values

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

Let $\mathbf{f}_i\mathbf{(t)}$ represent the cumulative animation function for a given iteration `i`.

The first iteration $\mathbf{f}_0\mathbf{(t)}$ is unaffected by `accumulate`, and so is the same as the original animation function definition.

$f_0(t) = f(t)$

Let **ve** be the last value specified for the animation function (e.g., the "to" value, the last value in a "values" list, or the end of a "path"). Each iteration after the first adds in the computed offset:

$f_i(t) = (ve * i) + f(t)$; $i >= 1$

3.3.2. Controlling the active duration

SMIL Animation provides an additional control over the active duration. The end attribute allows the author to constrain the active duration of the animation by specifying an end value, using a simple offset, a time base, an event-base or DOM method calls. The end attribute can *constrain* but not *extend* the active duration that is otherwise defined by dur and any repeat behavior. The rules for combining the attributes to compute the active duration are presented in the section, Computing the active duration.

end

Defines an end value for the animation that can constrain the active duration. The attribute value is a semi-colon separated list of values.

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.

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

Specifies the presentation time of the end. The end value is thus defined relative to the document begin.

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 element begin. The animation 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 ")"

Describes an accessKey that determines the end value. The end value is defined relative to the time that the accessKey character is input by the user.

wallclock-sync-value : "wallclock(" wallclock-value ")"

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

"indefinite"

The end value of the animation will be determined by a "endElement()" (or equivalent) method call.
The SMIL Animation DOM methods are described in the Supported methods section.
Hyperlink-based timing is described in the Hyperlinks and timing section.

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

Examples:

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.

```
<animate dur="2s" repeatDur="10s" end="foo.end" ... />
```

In the following example, the animation begins when the user clicks on the target element. The active duration will end 30 seconds after the document begins. Note that if the user has not clicked on the target element before 30 seconds elapse, the animation will never begin.

```
<animate begin="click" dur="2s" repeatDur="indefinite"
         end="30s" ... />
```

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

In this example, a presentation describes some factory processes. It uses animation to move an image around (e.g. against a background), demonstrating how an object moves from one part of a factory to another. Each step is a motion path, and set to repeat 3 times to make the point clear. Each animation can also be ended by clicking on some element "next" that allows the user to advance the presentation to the next step.

```
<img id="objectToMove" ... >
  <animateMotion id="step1" begin="0" dur="5s"
     repeatCount="3" end="next.click" path.../>
  <animateMotion id="step2" begin="step1.end" dur="5s"
     repeatCount="3" end="next.click" path.../>
  <animateMotion id="step3" begin="step2.end" dur="5s"
     repeatCount="3" end="next.click" path.../>
  <animateMotion id="step4" begin="step3.end" dur="5s"
     repeatCount="3" end="next.click" path.../>
  <animateMotion id="step5" begin="step4.end" dur="5s"
     repeatCount="3" end="next.click" path.../>
</img>
```

In this case, the active end of each animation is defined to be the earlier of 15 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.

3.3.3. 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, the respective attribute will be ignored (as though it were not specified).

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).

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.

See also the section The min attribute and restart.

3.3.4. Computing the active duration

The table in Figure 4 shows the semantics of all possible combinations of simple duration, `repeatCount` and `repeatDur`, and `end`. The following conventions are used in the table:

- If a cell is empty, it indicates that the associated attribute is omitted in the syntax.
- Where the table entry is "defined", this corresponds to an explicit specified value other than "indefinite". Note that if the simple duration is not specified, it is defined to be indefinite.
- Where the entry is a star ("*"), the value does not matter and can be any of the possibilities.

Additionally, the following rules must be followed in computing values:

- Where the active duration is specified as the minimum of several values (MIN), it may not always be possible to calculate this when the document begins. If the `end` is event-based or DOM-based, then an event or method call that activates `end` *before* the duration specified by `dur` and/or `repeatCount` or `repeatDur` will cut short the active duration at the `end` activation time.
- If the value of `end` cannot be resolved (e.g. when it is event-based), the value is considered to be "indefinite" for the purposes of evaluating the active duration. If and when the end value becomes resolved, the active duration is reevaluated.

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

- If `end` is specified but neither of `repeatCount` or `repeatDur` are specified, then the active duration is defined as the minimum of the simple duration and the duration defined by `end`.
- If both `end` and either (or both) of `repeatCount` or `repeatDur` are specified, the active duration is defined by the minimum duration defined by the respective attributes.
- It is possible to have an indefinite simple duration and a defined, finite active duration. The active duration can *constrain* (cut short) the simple duration, but the active duration does not define the simple duration, or change its value (i.e., the simple duration is still indefinite as used in the simple animation function).
- For any active duration and simple duration that are both not indefinite, the number of repeat iterations is defined by the active duration divided by the

simple duration (this may yield partial repeat iterations, just as `repeatCount` can specify).

The following symbols are used in the table:

B
The begin of an animation.
d
The simple duration of an animation.

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

Figure 4: Computing the active duration for different combinations of simple duration, `repeatCount` and `repeatDur`, and `end`.

3.3.5. Freezing animations

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

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

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

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

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

The animation ends 15 seconds after the document begin, but the image remains at the top value of 103. The attribute *freezes* the last value of the animation for the remainder of the document duration.

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

`fill` = "freeze | remove"
This attribute can have the following values:
freeze
The animation effect F(t) is defined to freeze the effect value at the last value of the active duration. The animation effect is "frozen" for the remainder of the document duration (or until the animation is restarted - see Restarting animations).
remove
The animation effect is removed (no longer applied) when the active duration of the animation is over. After the active end of the animation, the animation no longer affects the target (unless the animation is restarted - see Restarting animations).
This is the default value.

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

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

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

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

Comparison to SMIL timing

SMIL Animation specifies that `fill="freeze"` remains in effect for the remainder of the document, or until the element is restarted. In the more general SMIL timing model that allows time containers, the duration of the freeze effect is controlled by the time container, and never extends past the end of the time container simple duration. While this may appear to conflict, the SMIL Animation definition of `fill="freeze"` is consistent with the SMIL timing model. It is simply the case that in SMIL Animation, the document is the only "time container", and so the effect is as described above.

3.3.6. Additive animation

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

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

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

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

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

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

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

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

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

Additive animation is defined for numeric attributes and other data types for which some addition function is defined. This includes numeric attributes for concepts such as position, widths and heights, sizes, etc. This also includes color (refer to The animateColor element), and may include other data types as specified by the host language.

It is often useful to combine additive animations and `fill` behavior, for example when a series of motions are defined that should build upon one another:

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

The image moves along the first path, and then starts the second path from the end of the first, then follows the third path from the end of the second, and stays at the final point.

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

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

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

In addition, some numeric attributes (e.g., a telephone number attribute) may not sensibly support addition - it is left to the host language to specify which attributes support additive animation. Attribute types such as strings and Booleans for which addition is not defined, cannot support additive animation.

How from, to and by attributes affect additive behavior.

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

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

animation, however the ending value specified by the **to** attribute overrides the underlying value as though the animation was non-additive.

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

d
: is the simple duration

t
: is a time within the simple duration (0 <= t <= d)

v_{cur}
: is the current base value (at time t)

v_{to}
: is the defined "to" value

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

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

Multiple *to-animations* will also combine according to these semantics. The higher-priority animation will "win", and the end result will be to set the attribute to the final value of the higher-priority *to-animation*.

Multiple *by-animations* combine according to the general rules for additive animation and the animation sandwich model.

The use of `from` values does not imply either additive no non-additive animation, and both are possible. The `from` value for an additive animation is simply added to the underlying value, just as for the initial value is in animations specified with a `values` list. Additive behavior for *from-to* and *from-by* animations is controlled by the `additive` attribute, as in the general case.

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

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

The presentation value for "x" in the example above, over the course of the 10 seconds is presented in Figure 5 below. These values are simply computed using the formula described above. Note that the value for **`F(t)`** for A2 is the presentation value for "x".

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

Figure 5: Effect of Additive to-animation example

Additive and Cumulative animation

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

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

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

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

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

3.3.7. Restarting animation

When an animation is defined to begin at a simple offset (e.g. `begin="5s"`), there is an unequivocal time when the element begins. However, if an animation is defined to begin relative to an event (e.g. `begin="foo.click"`), the event can happen at any time, and moreover can happen *more than once* (e.g., if the user clicks on "foo" several times). In some cases, it is desirable to *restart* an animation if a second begin event is received. In other cases, an author may want to preclude this behavior. The `restart` attribute controls the circumstances under which an animation is restarted:

`restart` = "always | whenNotActive | never"

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

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

never
: The animation cannot be restarted for the remainder of the document duration.

Note that there are several ways that an animation 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 animation with `begin` specified as an event-value can be restarted when the named event fires multiple times.
- An animation with `begin` specified as a syncbase value, where the syncbase element can restart. When an animation restarts, other animations defined to begin relative to the begin or active end of the restarting animation may also restart (subject to the value of `restart` on these elements).
- An animation with `begin` specified as "indefinite" can be restarted when the DOM methods `beginElement()` or `beginElementAt()` are called repeatedly.

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

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

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).

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

Note that using restart can also allow the author to define a single UI event to both begin and end an element, as follows:

```
<img ...>
   <animate id="foo" begin="click" dur="2s"
       repeatDur="indefinite" end="click"
       restart="whenNotActive" ... />
</img>
```

If "foo" were defined with the default restart behavior "always", a second click on the image would simply restart the animation. However, since the second click cannot restart the animation when restart is set to "whenNotActive", the click will just end the active duration and stop the animation. This is sometimes described as "toggle" activation. See also Event sensitivity and Unifying event-based and scheduled timing.

Resetting element state

This section is normative

When an element restarts, certain state is "reset":

- 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.

Comparison to SMIL timing

SMIL Animation specifies that `restart="never"` precludes restart for the remainder of the document duration. In the more general SMIL 2.0 [SMIL20] timing model that allows time containers, the duration of the `restart="never"` semantic is defined by the time container, and only extends to the end of the time container simple duration. While this may appear to conflict, the SMIL Animation definition of `restart="never"` is consistent with the SMIL timing model. It is simply the case that in SMIL Animation, the document is the only "time container", and so the effect is as described above.

3.4. Handling syntax errors

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

In particular, this means that if other animation elements are defined to begin or end relative to an animation that "has no effect", the other animation elements will begin and end as though there were no syntax errors. The presentation runtime may indicate an error, but need not halt presentation or animation of the document.

Some host languages and/or runtimes may choose to impose stricter error handling (see also Error handling semantics for a discussion of host language issues with error handling). Authoring environments may also choose to be more intrusive when errors are detected.

3.5. The animation sandwich model

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

The remainder of this discussion uses the generic term OM for both the XML DOM [DOM-Level-2] as well as the CSS-OM. If an implementation does not support an object model, it should ideally maintain the original value as defined by the document as well as the presentation value; for the purposes of this section, we will consider this original value to be equivalent to the value in the OM.

In some implementations of DOM, it may be difficult or impractical to main a presentation value as described. CSS values should always be supported as described, as the CSS-OM provides a mechanism to do so. In implementations that do not support separate presentation values for general XML DOM properties, the implementation must at least restore the original value when animations no longer have an effect.

The rest of this discussion assumes the recommended approach using a separate presentation value.

The model accounting for the OM and concurrently active or frozen animations for a given attribute is described as a "sandwich", a loose analogy to the layers of meat and cheeses in a "submarine sandwich" (a long sandwich made with many pieces of meats and cheese layered along the length of the bread). In the analogy, time is associated with the length of the sandwich, and each animation has its duration represented by the length of bread that the layer covers. On the bottom of the sandwich is the base value taken from the OM. Each active (or frozen) animation is a layer above this. The layers (i.e. the animations) are placed on the sandwich both in time along the length of the bread, as well as in order according to *priority*, with higher priority animations placed above (i.e. on top of) lower priority animations. At any given point in time, you can take a slice of the sandwich and see how the animation layers stack up.

Note that animations manipulate the presentation value coming out of the OM in which the attribute is defined, and pass the resulting value on to the next layer of document processing. This does not replace or override any of the normal document OM processing cascade.

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

In CSS2 and the DOM 2 CSS-OM, the terms "specified", "computed" and "actual" are used to describe the results of evaluating the syntax, the cascade and the presentation rendering. When animation is applied to CSS properties of a particular element, the base value to be animated is read using the (readonly) `getComputedStyle()` method on that element. The values produced by the animation are written into an override stylesheet for that element, which may be obtained using the `getOverrideStyle()` method. These new values then affect the cascade and are reflected in a new computed value (and thus, modified presentation). This means that the effect of animation overrides all style sheet rules, except for user rules with the `!important` property. This enables `!important` user style settings to have priority over animations, an important requirement for accessibility. Note that the animation may have side effects upon the document layout. See also the [CSS2] specification (the terms are defined in section 6.1), and the [DOM2CSS] specification (section 5.2.1).

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

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

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

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

Some attributes that support additive animation have a defined legal range for values (e.g., an opacity attribute may allow values between 0 and 1). In some cases, an animation function may yield out of range values. It is recommended that implementations clamp the results to the legal range as late as possible, before applying them to the presentation value. Ideally, the effect of all the animations active or frozen at a given point should be combined, before any clamping is performed. Although individual animation functions may yield out of range values, the combination of additive animations may still be legal. Clamping only the final result and not the effect of the individual animation functions provides support for these cases. Intermediate results may be clamped when necessary although this is not optimal. The host language must define the clamping semantics for each attribute that can be animated. As an example, this is defined for The animateColor element.

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

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

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

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

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

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

3.6. Timing model details

3.6.1. Timing and real-world clock times

Throughout this specification, animation is described as a function of "time". In particular, the animation function is described as producing a value for any "time" in the range of the simple duration. However, the simple duration can be repeated, and the animation can begin and restart in many ways. As such, there is no direct relationship between the "time" that an animation function uses, and the real world concept of time as reflected on a clock.

When a `keySplines` attribute is used to adjust the pacing between values in an animation, the semantics can be thought of as changing the pace of time in the given interval. An equivalent model is that `keySplines` simply changes the pace at which interpolation *progresses* through the given interval. The two interpretations are equivalent mathematically, and the significant point is that the notion of "time" as defined for the animation function **`f(t)`** should not be construed as real world clock time. For the purposes of animation, "time" can behave quite differently from real world clock time.

3.6.2. Interval timing

SMIL Animation assumes the most common model for *interval timing*. This describes intervals of time (i.e. durations) in which the begin time of the interval is included in the interval, but the end time is excluded from the interval. This is also referred to as *end-point exclusive* timing. This model makes arithmetic for intervals work correctly, and provides sensible models for sequences of intervals.

Background rationale

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

In the world of media and timelines, the same applies: Let A be a video, a clip of audio, or an animation. Assume "A" begins at 10 and runs until 15 (in any units - it does not matter). If "B" is defined to follow "A", then it begins at 15 (and not at 15 plus some minimum interval). When an animation runtime engine 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 animation

For animation, several results of this are important: the definition of repeat, and the value sampled during the "frozen" state.

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

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

At time 0, the simple duration is sampled at 0, and the first value is applied. This is the *inclusive* begin of the interval. The simple duration is sampled normally up to 4 seconds. However, the appropriate way to map time on the active duration to time on the simple duration is to use the remainder of division by the simple duration:

```
simpleTime = REMAINDER( activeTime, d )
```

or

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

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

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

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

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

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

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

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

3.6.3. Unifying interactive and scheduled timing

SMIL Animation describes extensions to SMIL 1.0 to support interactive timing of animation elements. These extensions allow the author to specify that an animation should begin or end in response to an event (such as a user-input event like "click"), or to a hyperlink activation, or to a DOM method call.

The syntax to describe this uses event-value specifications and the special argument value "indefinite" for the `begin` and `end` attribute values. Event values describe user interface and other events. DOM method calls to begin or end an animation require that the associated attribute use the special value "indefinite". A hyperlink can also be targeted at an animation element that specifies `begin="indefinite"`. The animation will begin when the hyperlink is activated (usually by the user clicking on the anchor). It is not possible to directly control the active end of an animation using hyperlinks.

Background

The current model represents an evolution from earlier multimedia runtimes. These were typically either pure, static schedulers or pure event-based systems. Scheduler models present a linear timeline that integrates both discrete and continuous media. Scheduler models tend to be good for storytelling, but have limited support for user-interaction. Event-based systems, on the other hand, model multimedia as a graph of event bindings. Event-based systems provide flexible support for user-interaction, but generally have poor scheduling facilities; they are best applied to highly interactive and experiential multimedia.

The SMIL 1.0 model is primarily a scheduling model, but with some flexibility to support continuous media with unknown duration. User interaction is supported in the form of timed hyperlinking semantics, but there was no support for activating individual elements via interaction.

Modeling interactive, event-based content in SMIL

To integrate interactive content into SMIL timing, the SMIL 1.0 scheduler model is extended to support several new concepts: *indeterminate timing,* and *activation* of the element.

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 (e.g., by a call to the `beginElement()` method). From a scheduling perspective, the time is described as *unresolved* before the activation. Once the element begin or end has been activated, the time is *resolved*.

The event-activation support provides a means of associating an event with the begin or active end time for an element. When the event is raised (e.g., when the user clicks on something), the associated time is resolved to a *determinate* time. For event-based begin times, the element becomes active (begins to play) at the time that the event is raised (plus any specified offset). The element plays from the beginning of the animation function. For event-based active end times, the element becomes inactive (stops playing) when the associated event is raised.

Note that an event based `end` will not be activated until the element has already begun. Any specified `end` event is ignored before the element begins. See also Event sensitivity.

Note that when an element restarts, any event-based end time that was resolved in the previous instance of play, will be reset to the unresolved state.

Related to event-activation is *link-activation*. Hyperlinking has defined semantics in SMIL 1.0 to *seek* a document to a point in time. When combined with indeterminate timing, hyperlinking yields a variant on interactive content. A hyperlink can be targeted at an element that does not have a scheduled begin time. When the link is traversed, the element begins. The details of when hyperlinks activate an element, and when they seek the document timeline are presented in the next section.

Note that hyperlink activation only applies to an element begin time, and not to the element end. Event and DOM based activation can apply to both begin and end times.

Note that elements can define the `begin` or `end` relative to another element, using a syncbase-value (the begin or end of another element). If the syncbase element is in turn defined with, for example, event-based times, the syncbase value is not resolved, and so the `begin` or `end` of the current element is also unresolved. For a `begin` or `end` time to be resolved, any referenced syncbase value must also be resolved.

3.6.4. Event sensitivity

This section is informative

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

This section is normative

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

1. If an element is not active, 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.
2. 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.

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.

3.6.5. Hyperlinks and timing

Hyperlinking semantics must be specifically defined for animation in order to ensure predictable behavior. Earlier hyperlinking semantics, such as those defined by SMIL 1.0 are insufficient because they do not handle indeterminate and interactive timing. Here we extend SMIL 1.0 semantics for use in presentations that include animations with indeterminate and interactive timing.

Hyperlinking behavior is described as *seeking* the document. To *seek* in this sense means to advance the document timeline to the specified time.

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

1. If the target element is active, seek the document time back to the (current) begin time of the element. If there are multiple begin times, use the begin time that corresponds to the current "begin instance".
2. Else if the target element begin time is resolved (i.e., there is any resolved time in the list of begin times, or if the begin time was forced by an earlier hyperlink or a `beginElement()` method call), seek the document time (forward or back, as needed) to the earliest resolved begin time of the target element. Note that the begin time may be resolved as a result of an earlier hyperlink, DOM or event activation. Once the begin time is resolved, hyperlink traversal always seeks.
3. Else (animation begin time is unresolved) just resolve the target animation begin time at current document time. Disregard the sync-base or event base of the animation, and do not "back-propagate" any timing logic to resolve the child, but rather treat it as though it were defined with `begin="indefinite"` and just resolve begin time to the current document time.

Note that hyperlink activation does not introduce any restart behavior, and is not subject to the `restart` attribute semantics.

If a seek of the document 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 current time at the moment of hyperlink traversal.

After seeking a document forward, the document should be in the same state as if the user had allowed the presentation to run normally from the current time until reaching the animation element begin time (but had otherwise not interacted with the document). In particular, seeking the presentation time forward should also cause any other animation elements that have resolved begin times between the current time and the seeked-to time to begin. These elements may have ended, or may still be active or frozen at the seeked-to time, depending upon their begin times and active durations. Also any animation elements currently active at the time of hyperlinking should "fast-forward" over the seek interval. These may end or may be still active or frozen at the seeked-to time, depending upon their active durations. The net effect is that seeking forward to a presentation time puts the document into a state identical to that as if the document presentation time advanced undisturbed to reach the seek time.

If the resolved begin time for an animation element that is the target of a hyperlink is before the current presentation time, the presentation must seek backwards. Seeking backwards will rewind any animations active during the seek interval and will turn off any animations that are resolved to begin at a time after the seeked-to time. Note that resolved begin times (e.g. a begin associated with an event) are not cleared or lost by seeking to an earlier time. Subject to the rules above for hyperlinks that target animation elements, hyperlinking to elements with resolved begin times will function normally, advancing the presentation time forward to the previously resolved time.

These hyperlinking semantics assume that a record is kept of the resolved begin time for all animation elements, and this record is available to be used for determining the correct presentation time to seek to. Once resolved, begin times are not cleared. However, they can be overwritten by subsequent resolutions driven by multiple occurrences of an event (i.e. by restarting). For example:

```
<animate id="A" begin="10s" .../>
<animate id="B" begin="A.begin+5s" .../>
<animate id="C" begin="click" .../>
<animate id="D" begin="C.begin+5s" .../>
...
<a href="#D">Start the last animation</a>
```

The begin time of elements "A" and "B" can be immediately resolved to be at 10 and 15 seconds respectively. The begin of elements "C" and "D" are unresolved when the document starts. Therefore activating the hyperlink will have no effect upon the presentation time or upon elements "C" and "D". Now, assume that "C" is clicked at 25 seconds into the presentation. The click on "C" in turn resolves "D" to begin at 30 seconds. From this point on, traversing the hyperlink will cause the presentation time to be seeked to 30 seconds.

If at 60 seconds into the presentation, the user again clicks on "C", "D" will become re-resolved to a presentation time of 65 seconds. Subsequent activation of the hyperlink will result in the seeking the presentation to 65 seconds.

3.6.6. Propagating changes to times

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

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

```
<rect ...>
   <animate id="bar" end="click" .../>
   <animate id="foo" begin="bar.end" .../>
</rect>
```

we say that "foo" is a *time-dependent* of "bar" - that is, the "foo" begin time depends upon the active end of "bar".

An element ***A*** is a time dependent of another element ***B*** if ***A*** specifies ***B*** as a syncbase element. In addition, if element ***A*** is a time dependent of element ***B***, and if element ***B*** is a time dependent of element ***C*** (i.e., element ***B*** defines element ***C*** as a syncbase element), then element ***A*** is an *indirect* time dependent of element ***C***.

When an element begins or ends, the time dependents of the element are effectively notified of the action, and the schedule for the time dependents may be affected. Note than an element must actually begin before any of the time dependents (dependent on the begin) are affected, and that an element must actually end before any of the time dependents (dependent on the end) are affected. This impacts the definition of the priority ordering of animation elements, as discussed in The animation sandwich model.

In the example above, any changes to the active end time of "bar" must be propagated to the begin of "foo". The effect of the changes depends upon the state of "foo" when the change happens, as detailed below.

If the *begin* time of an element is dependent upon another element (as for "foo" in the example), the resulting behavior when the syncbase element ("bar") propagates changes is determined as follows:

- If the time dependent ("foo") has not yet begun, then the begin time is simply updated in the schedule.
- If the time dependent ("foo") is currently active, then the `restart` attribute determines the behavior: if it is "always", then the time dependent will restart; otherwise the propagated change is ignored.
- If the time dependent ("foo") has already begun (at least once) but is not currently active, then the `restart` attribute determines the behavior: if it is "always" or "whenNotActive", then the time dependent will restart; otherwise the propagated change is ignored.

Note that the semantic is directly analogous to event-base timing and the `restart` attribute.

If the *end* time of an element is dependent upon another element, the semantic is much simpler:

- If the time dependent has not yet begun or is currently active, then the end time is simply updated in the schedule, and the active duration is recalculated (according to the table in Computing the active duration).
- If the time dependent has already ended the active duration, then the change is ignored. Even if the recomputed active duration would extend past the current time, the element does not "restart" and "re-end".

Another way to think of this is that the end time is always recalculated, but it will not affect the presentation unless the element is currently active, or unless the element begins (or restarts) after the change happens.

3.6.7. Timing attribute value grammars

This section is normative

The syntax specifications are defined using EBNF notation as defined in [XML10]

In the syntax specifications that follow, allowed white space is indicated as "S", defined as follows (taken from the [XML10] definition for "S"):

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

Begin values

This section is normative

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

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

Parsing timing specifiers

Several of the timing specification values have a similar syntax. In addition, XML ID attributes are allowed to contain the dot '.' separator character. The backslash character '\' can be used to escape the dot separator within identifier and event-name references. To parse an individual item in a value-list, the following approach defines the correct interpretation.

1. 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 token "wallclock", it should be parsed as a wallclock-sync-value.

4. Else if the value is the 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). In the following, ignore any `'.'` separator characters preceded by a backslash `'\'` escape character. In addition, strip any leading backslash `'\'` escape character.
    1. If the token contains no `'.'` separator character, then the value should be parsed as an event-value with an unspecified (i.e. default) eventbase-element.
    2. Else if the token ends with the string "`.begin`" or "`.end`", then the value should be parsed as a syncbase-value.
    3. Else, the value should be parsed as an event-value (with a specified eventbase-element). Before parsing the event value, any backslash `'\'` escape character after the `'.'` separator character should be removed.

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 document time.

The implicit syncbase for an offset value is the document begin.

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   ::= IDREF
```

- The IDREF is a legal XML identifier.

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.
- 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.
- If the clock value is omitted, it defaults to "0".
- No embedded white space is allowed between a syncbase element and a time-symbol.
- White space will be ignored before and after a "+" or "-" for a clock value.
- Leading and trailing white space characters (i.e. before and after the entire syncbase value) will be ignored.

Examples:

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

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 [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 is defined to be the target element of the animation,

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

- 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.

Examples:

```
begin=" x.load "          : Begin when "load" is observed on "x"
begin="x.focus+3s"        : Begin 3 seconds after an "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"
```

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.

The following example describes a qualified repeat eventbase value:

```
<animate id="foo" repeatCount="10" end="endAnim.click"
... />
<img id="endAnim" begin="foo.repeat(2)" .../>
```

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

AccessKey values

AccessKey values allow an author to tie a begin or end time to a particular keypress, 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.

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-val ::= "wallclock(" S (DateTime | WallTime)  S ")"
DateTime      ::= Date "T" WallTime
Date          ::= Years "-" Months "-" Days
WallTime      ::= (HHMM-Time | HHMMSS-Time)(TZD)?
```

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

- 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.

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.

3.6.8. Evaluation of begin and end time lists

This section is informative

Animation elements 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. In this version of the time model without time containers, instance times remain in the lists forever, once they have been added. For example, times associated with event-values are only added when the associated event happens, but remain in the lists thereafter. Similarly, Instance times for syncbase-values are added to the list each time a new interval is created for the syncbase element, and remain in the list.

When the timegraph is initialized, each element creates 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 document time (relative to the document begin). All sync-arcs, event arcs, wallclock values, etc. must be converted to this time space for easy comparison.

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. 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

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.
- **wallclock-sync-values** are similar to offset values. Each wallclock-sync-value condition yields a single instance time.
- **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, 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. 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.

If no attribute is present, the default begin value (an offset-value of 0) must be evaluated.

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 document 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 document 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.

Element life-cycle

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.

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 document. The cycle begins by computing the first current interval. This requires some special consideration of the lists of times, but is relatively straight-forward. 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 two edge cases:

1. The element can begin before the document begins, and so appears to begin part way into the local timeline. The model must handle begin times before the document begin (i.e. before 0).
2. The element has one or more intervals defined that begin *and end* before the document begins (before 0). These are filtered out of the model.

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

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

// Calculates the first acceptable interval for an element
// Returns:
//    Interval if there is such an interval
//    FAILURE if there is no such interval
Interval getFirstInterval()
{
Time beginAfter=-INFINITY;
while( TRUE ) // loop till return
{
   Set tempBegin = the first value in the begin list that is >=
beginAfter.
   If there is no such value  // No interval
      return FAILURE;
   If there was no end attribute specified
      // this calculates the active end with no end constraint
      tempEnd = calcActiveEnd( tempBegin );
   else
   {
      // We have a begin value - get an end
      Set tempEnd = the first value in the end list that is >=
tempBegin.
      // Allow for non-0-duration interval that begins immediately
      // after a 0-duration interval.
      If tempEnd == tempBegin && tempEnd has already been used in
        an interval calculated in this method call
      {
         set tempEnd to the next value in the end list that is >
tempEnd
      }
      If there is no such value
      {
         // Events leave the end open-ended. If there are other
         // conditions that have not yet generated instances,
         // they must be unresolved.
         if endHasEventConditions()
            OR if the instance list is empty
            tempEnd = UNRESOLVED;
         // if all ends are before the begin, bad interval
         else
            return FAILURE;
      }
      // this calculates the active dur with an end constraint
      tempEnd = calcActiveEnd( tempBegin, tempEnd );
   }
   // We have an end - is it after the parent simple begin?
   if( tempEnd > 0 )
      return( Interval( tempBegin, tempEnd ) );
   // interval is too early
   else if( restart == never )
      // if can't restart, no good interval
      return FAILURE;
   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 (i.e., the interval end is before the interval begin). In this case, the interval must be deleted 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.

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

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

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

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

Detecting Cycles

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

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

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

Examples

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

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

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

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

Example 3: In the following example, the 2 animations define begin times that are mutually dependent, but the first has multiple begin conditions that allow the cycle to propagate forwards. The animation "foo" will first be active from 0 to 3 seconds, with the second animation "bar" active 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.

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

```
<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>
```

3.7. Animation function value details

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

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

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

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

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

1. The first interface converts a string (the animation function value) to a unitless, canonical number (either an integer or a floating point value). This allows animation elements to interpolate between values without requiring specific knowledge of data types like CSS-length. The interface will likely require a reference to the target attribute, to determine the legal abstract values. If the passed string cannot be converted to a unitless scalar, the animation element will treat the animation function values as strings, and the `calcMode` will default to "discrete".
2. The second interface converts a unitless canonical number to a legal string value for the target attribute. This may, for example, simply convert the

number to a string and append a suffix for the canonical units. The animation element uses the result of this to actually set the presentation value.

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

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

The above recommendations notwithstanding, it is sometimes useful to interpolate values in a specific unit-space, and to apply the result using the specified units rather than canonical units. This is especially true for certain relative units such as those defined by CSS (e.g. em units). If an animation specifies all the values in the same units, an implementation may use knowledge of the associated syntax to interpolate in the unit space, and apply the result within the animation sandwich, in terms of the specified units rather than canonical units. As noted above, this solution does not scale well to the general case. Nevertheless, in certain applications (such as CSS properties), it may be desirable to take this approach.

3.8. Common syntax DTD definitions

Timing attributes

```
<!ENTITY % timingAttrs
  begin          CDATA   #IMPLIED
  dur            CDATA   #IMPLIED
  end            CDATA   #IMPLIED
  restart        (always | never | whenNotActive)   "always"
  repeatCount    CDATA   #IMPLIED
  repeatDur      CDATA   #IMPLIED
  fill           (remove | freeze) "remove"
>
```

Animation attributes

```
<!ENTITY % animAttrs
  attributeName  CDATA   #REQUIRED
  attributeType  CDATA   #IMPLIED
  additive       (replace | sum) "replace"
  accumulate     (none | sum) "none"
>
<!ENTITY % animTargetAttr
  targetElement  IDREF   #IMPLIED
>
<!ENTITY % animLinkAttrs
  type      (simple | extended | locator | arc) #FIXED
"simple"
  show      (new | embed | replace) #FIXED 'embed'
  actuate   (user | auto) #FIXED 'auto'
  href      CDATA   #IMPLIED
>
```

4. Animation elements

4.1. The animate element

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

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

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

Numerous examples are provided above.

4.2. The set element

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

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

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

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

```
<!ELEMENT set EMPTY>
```

```
<!ATTLIST set
  %timingAttrs
  attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED
  to             CDATA  #IMPLIED
>
```

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

Examples

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

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

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

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

4.3. The animateMotion element

The **`<animateMotion>`** element will move an element along a path. The element abstracts the notion of motion and position across a variety of layout mechanisms - the host language defines the layout model and must specify the precise semantics of position and motion. The path can be described in several ways:

- Specifying x,y pairs for the `from/to/by` attributes. These will define a straight line motion path.
- Specifying x,y pairs for the `values` attribute. This will define a motion path of straight line segments, or points (if `calcMode` is set to discrete). This will override any `from/to/by` attribute values.
- Specifying a path in the path attribute. This will define a motion path using a subset of the SVG path syntax, and provides smooth path motion. This will override any `from/to/by` or `values` attribute values.

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

The syntax for the x, y value pairs is:

```
coordinate-pair ::= ( coordinate comma-wsp coordinate )
coordinate      ::= Number
```

Coordinate values are separated by at least one white space character or a comma. Additional white space around the separator is allowed. The values of `coordinate` must be defined as some sort of number in the host language.

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

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

As noted in Animation function values, if any values (i.e., the argument-values for `from`, `to`, `by` or `values` attributes, *or* for the `path` attribute) are not legal, the animation will have no effect (see also Handling Syntax Errors). The same is true if none of the `from`, `to`, `by`, `values` or `path` attributes are specified.

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

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

`path` = "<path-description>"

Specifies the curve that describes the attribute value as a function of time. The supported syntax is a subset of the SVG path syntax. Support includes commands to describes lines ("MmLlHhVvZz") and Bezier curves ("Cc"). For details refer to the path specification in SVG [SVG].

Note that SVG provides two forms of path commands - "absolute" and "relative". These terms may appear to be related to the definition of additive animation and/or to the "origin" attribute, but they are orthogonal. The terms "absolute" and "relative" apply only to the definition of the path itself, and not to the operation of the animation. The "relative" commands define a path point relative to the previously specified point. The terms "absolute" and "relative" are unrelated to the definitions of both "additive" animation and any specification of "origin".

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

A path data segment must begin with either one of the "moveto" commands.

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

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

Horizontal Line To commands - "H <x>" or "h <dx>"
Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). Multiple x values can be provided.

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

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

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

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

`calcMode`
Defined as above in Animation function calculation modes, but note that the default `calcMode` for `animateMotion` is "paced". This will produce constant velocity motion across the path.

The use of "discrete" for the `calcMode` together with a "`path`" specification is allowed, but will simply jump the target element from point to point. If the `keyTimes` attribute is not specified, the times are derived from the points in the `path` specification (as described in Animation function calculation modes).

The use of "linear" for the `calcMode` with more than 2 points described in "`values`", "`path`" or "`keyTimes`" may result in motion with varying velocity. The "linear" `calcMode` specifies that time is evenly divided among the segments defined by the "`values`" or "`path`" (note: any "`keyTimes`" list defines the same number of segments). The use of "linear" does not specify that time is divided evenly according to the *distance* described by each segment.

For motion with constant velocity, `calcMode` should be set to "paced".

For complete velocity control, calcMode can be set to "spline" and the author can specify a velocity control spline with "`keyTimes`" and "`keySplines`".

`origin` = "default"
Specifies the origin of motion for the animation. The values and semantics of this attribute are dependent upon the layout and positioning model of the host language. In some languages, there may be only one option (i.e. "default"). However, in CSS positioning for example, it is possible to specify a motion path relative to the container block, or to the layout position of the element. It is often useful to describe motion relative to the position of the element as it is laid out (e.g., from off screen left to the layout position, specified as from="(-100, 0)" and to="(0, 0)". Authors must be able to describe motion both in this manner, as well as relative to the container block. The `origin` attribute supports this distinction. Nevertheless, because the host language defines the layout model, the host language must also specify the "default" behavior, as well as any additional attribute values that are supported.

Note that the definition of the layout model in the host language specifies whether containers have bounds, and the behavior when an element is moved outside the bounds of the layout container. In CSS2 [CSS2], for example, this can be controlled with the "clip" property.

Note that for additive animation, the "origin" distinction is not meaningful. This attribute only applies when `additive` is set to "replace".

4.4. The animateColor element

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

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

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

```
<!ELEMENT animateColor EMPTY>
<!ATTLIST animateColor
  %animAttrs
  %timingAttrs
```

```
    calcMode       (discrete | linear
                    | paced | spline ) "linear"
    values         CDATA  #IMPLIED
    from           CDATA  #IMPLIED
    to             CDATA  #IMPLIED
    by             CDATA  #IMPLIED
    keyTimes       CDATA  #IMPLIED
    keySplines     CDATA  #IMPLIED
  >
```

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

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

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

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

5. Integrating SMIL Animation into a host language

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

5.1. Required host language definitions

The host language designer must define some basic concepts in the context of the particular host language. These provide the basis for timing and presentation semantics.

The host language designer must define what "presenting a document" means. A typical example is that the document is displayed on a screen.

The host language designer must define the *document begin*. Possible definitions are that the document begins when the complete document has been received by a client over a network, or that the document begins when certain document parts have been received.

The host language designer must define the *document end*. This is typically when the associated application exits or switches context to another document.

A host language should provide a means of uniquely identifying each animation element within a document. The facility provided should be the same as for the other elements in the language. For example, since SMIL 1.0 identifies each element with an "id" attribute that contains an XML ID value for that element, animation elements added to SMIL 1.0 should also have an "id" attribute.

5.2. Required definitions and constraints on animation targets

Specifying the target element

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

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

Target attribute issues

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

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

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

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

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

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

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

Integrating animateMotion functionality

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

Language integration example: SVG

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

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

5.3. Constraints on manipulating animation elements

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

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

Note in particular that if the `attributeName` attribute can be changed (either by animation or script), problems may arise if the target attribute has a namespace qualified name. Current DOM specifications do not include a mechanism to handle this binding.

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

5.4. Required definitions and constraints on element timing

This specification assumes that animation elements are the only elements in the host language that have timing semantics (this restriction may be removed in a future version of SMIL Animation). This specification cannot be used for host languages that contain elements with timing semantics. For example, the following integration of animation with SMIL 1.0 is illegal with this version of SMIL animation:

```
<par id="illegalExample">
  <img begin="2s" dur="1m" src="foo.png" alt="Sad face
for bad example" />
    <anchor id="anc" href="#bar" coords="0%,0%,50%,50%"
dur="30s" />
    <set targetElement="anc" attributeName="coords"
         begin="10s" dur="20s" fill="freeze"
         to="50%,50%,100%,100%" />
  </img>
</par>
```

The set of "animation elements" that may have timing includes both the elements defined in this specification, as well as extension animation elements defined in host languages. Extension animation elements must conform to the animation framework described in this document. In particular, extension animation elements may not be defined to contain other animation elements in a way that would introduce hierarchic timing as supported by the `par` and `seq` elements in SMIL 1.0 [SMIL].

Supported events for event-base timing

The host language must specify which event names are legal in event base values. If the host language defines no allowed event names, event-based timing is effectively precluded for the host language.

Host languages may specify that dynamically created events (as per the [DOM2Events] specification) are legal as event names, and not explicitly list the allowed names.

5.5. Error handling semantics

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

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

5.6. SMIL Animation namespace

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

http://www.w3.org/2001/smil-animation

6. Document Object Model support

Any XML-based language that integrates SMIL Animation will inherit the basic interfaces defined in DOM [DOM-Level-2] (although not all languages may require a DOM implementation). SMIL Animation specifies the interaction of animation and DOM. SMIL Animation also defines constraints upon the basic DOM interfaces, and specific DOM interfaces to support SMIL Animation.

Note that the language designer integrating SMIL Animation must specify any constraints upon SMIL Animation with respect to the DOM. This includes the specification of language attributes that can or cannot be animated, as well as the definition of addition for any attributes that support additive animation.

6.1. Events and event model

This section is informative

SMIL event-timing assumes that the host language supports events, and that the events can be bound in a declarative manner. DOM Level 2 Events [DOM2Events] describes functionality to support this.

This section is normative

The specific events supported are defined by the host language. If no events are defined by a host language, event-timing is effectively omitted.

This module defines a set of events that may be included by a host language. These include:

beginEvent
: This event is raised when the element local timeline begins to play. It will be raised each time the element begins the active duration (i.e., when it restarts, but not when it repeats). It may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was begun with a DOM method.

endEvent
: This event is raised at the active end of the element. Note that this event is not raised at the simple end of each repeat. This event may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was ended with a DOM method.

repeat
: This event is raised when the element local timeline repeats. It will be raised each time the element repeats, after the first iteration. Associated with the repeat event is an integer that indicates which repeat iteration is beginning. The value is a 0-based integer, but the repeat event is not raised for the first iteration and so the observed values will be >= 1.

If an element is restarted while it is currently playing, the element will raise an `endEvent` and then a `beginEvent`, as the element restarts.

The `beginEvent` may not be raised at the time that is calculated as the begin for an element. For example the element can specify a begin time before the beginning of the document (either with a negative offset value, or with a syncbase time that resolves to a time before the document begin). In this case, a time dependent of the begin syncbase time will be defined relative to the calculated begin time. The `beginEvent` will be raise when the element actually begins - in the example case when the document 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 as a negative offset from a user event). See also the discussion Propagating changes to times.

6.2. Supported interfaces

SMIL Animation supports several methods for controlling the behavior of animation: `beginElement()`, `beginElementAt()`, `endElement()`, and `endElementAt()`. These methods are used to begin and end the active duration of an element. Authors can (but are not required to) declare the timing to respond to the DOM using the following syntax:

```
<animate begin="indefinite" end="indefinite" .../>
```

If a DOM method call is made to begin or end the element (using `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.

- 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.
- Note that `beginElement()` is subject to the restart attribute in the same manner that event-based begin timing is. Refer also to the section Restarting animations.

The expectation of the following interface is that an instance of the ElementTimeControl interface can be obtained by using binding-specific casting methods on an instance of an animate element. A DOM application can use the `hasFeature` method of the DOMImplementation interface to determine whether the `ElementTimeControl` interface is supported or not. The feature string for this interface is "TimeControl".

Interface *ElementTimeControl*

IDL Definition

```
interface ElementTimeControl {
  boolean            beginElement();
  boolean            beginElementAt(in float offset));
  boolean            endElement();
  boolean            endElementAt(in float offset);
};
```

Methods

`beginElement`

Creates a begin instance time for the current time.

Return Value

`void`

No Parameters

`beginElementAt`

Creates a begin instance time for the current time plus or minus the passed offset.

Parameters

`float`	`offset`	The offset in seconds at which to begin the element.

Return Value

`void`

`endElement`

Creates an end instance time for the current time.

Return Value

`void`

No Parameters

`endElementAt`

Creates an end instance time for the current time plus or minus the passed offset.

Parameters

`float`	`offset`	The offset in seconds at which to end the element. Must be >= 0.

Return Value

`void`

Interface *TimeEvent*

The `TimeEvent` interface provides specific contextual information associated with Time events.

IDL Definition

```
interface TimeEvent : events::Event {
  readonly attribute views::AbstractView  view;
  readonly attribute long             detail;
  void               initTimeEvent(in DOMString typeArg,
                                   in views::AbstractView
viewArg,
                                   in long detailArg);
};
```

Attributes

`view` of type `views::AbstractView`, readonly

The `view` attribute identifies the `AbstractView` from which the event was generated.

`detail` of type `long`, readonly

Specifies some detail information about the `Event`, depending on the type of event.

Methods

`initTimeEvent`

The `initTimeEvent` method is used to initialize the value of a `TimeEvent` created through the `DocumentEvent` interface. This method may only be called before the `TimeEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

Parameters

`DOMString`	`typeArg`	Specifies the event type.
`views::AbstractView`	`viewArg`	Specifies the `Event`'s `AbstractView`.
`long`	`detailArg`	Specifies the `Event`'s detail.

No Return Value
No Exceptions

The different types of events that can occur are:

begin

Raised when the element begins. See also Events and event model.

- Bubbles: No
- Cancelable: No
- Context Info: None

end

Raised when the element ends its active duration. See also Events and event model.

- Bubbles: No
- Cancelable: No
- Context Info: None

repeat

Raised when the element repeats. See also Events and event model.

- Bubbles: No
- Cancelable: No
- Context Info: detail (current iteration)

6.3. IDL definition

smil.idl:

```
// File: smil.idl
#ifndef _SMIL_IDL_
#define _SMIL_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module smil
{
  typedef dom::DOMString DOMString;

  interface ElementTimeControl {
    void            beginElement();
    void            beginElementAt(in float offset);
    void            endElement();
    void            endElementAt(in float offset);
  };

  interface TimeEvent : events::Event {
    readonly attribute views::AbstractView  view;
    readonly attribute long             detail;
    void               initTimeEvent(in DOMString
typeArg,
                                        in
views::AbstractView viewArg,
                                        in long detailArg);
  };
};

#endif // _SMIL_IDL_
```

6.4. Java language binding

org/w3c/dom/smil/ElementTimeControl.java:

```
package org.w3c.dom.smil;

import org.w3c.dom.DOMException;

public interface ElementTimeControl {
    public void  beginElement();

    public void  beginElementAt(float offset);

    public void endElement();

    public void endElementAt(float offset);

}
```

org/w3c/dom/smil/TimeEvent.java:

```
package org.w3c.dom.smil;

import org.w3c.dom.events.Event;
import org.w3c.dom.views.AbstractView;

public interface TimeEvent extends Event {
    public AbstractView getView();

    public int getDetail();

    public void initTimeEvent(String typeArg,
                              AbstractView viewArg,
                              int detailArg);
}
```

6.5. ECMAScript language binding

Object ElementTimeControl
- **The ElementTimeControl object has the following methods:**
  - **beginElement()**
    This method returns a **void**.
  - **beginElementAt(offset)**
    This method returns a **void**. The **offset** parameter is of type **float**.
  - **endElement()**
    This method returns a **void**.
  - **endElementAt(offset)**
    This method returns a **void**. The **offset** parameter is of type **float**.

Object TimeEvent
- **TimeEvent has all the properties and methods of Event as well as the properties and methods defined below.**
- **The TimeEvent object has the following properties:**
  - **view**
    This property is of type **AbstractView**.
  - **detail**
    This property is of type **long**.
- **The TimeEvent object has the following methods:**
  - **initTimeEvent(typeArg, viewArg, detailArg)**

This method returns a **void**. The **typeArg** parameter is of type **DOMString**. The **viewArg** parameter is of type **views::AbstractView**. The **detailArg** parameter is of type **long**.

7. Appendix: Differences from SMIL 1.0 timing model

- No time containers supported - does not support `<seq>` and `<par>`
- Renamed "element-event" concept to "syncbase value", changed syntax.
- Added event-based timing support for `begin` and `end` attributes.
- Added hyperlink activation support to `begin` attribute.
- Support DOM methods for activation of `begin` and `end` attributes, and for `begin`, `end` and `repeat` events.
- Modified `end` attribute semantics to align with SMIL 2.0.
- Added `repeatCount` and `repeatDur` and omitted `repeat`. This aligns with SMIL 2.0.
- Syncbase-value (offset) can exceed duration of syncbase element
- Tweaked Clock value definition to support >24 hours.

8. References

[CSS2]
"Cascading Style Sheets, level 2", B. Bos, H. W. Lie, C. Lilley, I. Jacobs, 12 May 1998. Available at http://www.w3.org/TR/REC-CSS2.

[COMP-GRAPHICS]
"Computer Graphics : Principles and Practice, Second Edition", James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips, Addison-Wesley, pp. 488-491.

[DATETIME]
"Date and Time Formats", M. Wolf, C. Wicksteed. W3C Note 27 August 1998, Available at: http://www.w3.org/TR/NOTE-datetime

[DOM-Level-2]
"Document Object Model (DOM) Level 2 Core Specification"
Available at http://www.w3.org/TR/DOM-Level-2-Core/.

[DOM2CSS]
"Document Object Model CSS"
Available at http://www.w3.org/TR/DOM-Level-2-Style/css.html.

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

[HTML]
"HTML 4.01 Specification", D. Raggett, A. Le Hors, I. Jacobs, 24 December 1999. Available at http://www.w3.org/TR/REC-html40.

[ISO8601]
"Data elements and interchange formats - Information interchange - Representation of dates and times", International Organization for Standardization, 1998.

[ISO10646]
""Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993. This reference refers to a set of codepoints that may evolve as new characters are assigned to them. This reference therefore includes future amendments as long as

they do not change character assignments up to and including the first five amendments to ISO/IEC 10646-1:1993. Also, this reference assumes that the character sets defined by ISO 10646 and Unicode remain character-by-character equivalent. This reference also includes future publications of other parts of 10646 (i.e., other than Part 1) that define characters in planes 1-16. "

[SMIL1.0]
"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification W3C Recommendation 15-June-1998 ".
Available at: http://www.w3.org/TR/REC-smil.

[SMIL20]
"Synchronized Multimedia Integration Language (SMIL 2.0) Specification",
Available at http://www.w3.org/TR/smil20/

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

[sRGB]
IEC 61966-2-1 (1999-10) - "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB", ISBN: 2-8318-4989-6 ICS codes: 33.160.60, 37.080 TC 100 51 pp.
Available at: http://www.iec.ch/nr1899.htm.

[SVG]
"Scalable Vector Graphics (SVG) 1.0 Specification", W3C Proposed Recommendation, 19 July 2001.
Available at http://www.w3.org/TR/SVG/.

[XLink]
"XML Linking Language (XLink)", S. DeRose, E. Maler, D. Orchard, editors, 27 June 2001. Available at http://www.w3.org/TR/xlink/

[XML]
"Extensible Markup Language (XML) 1.0", T. Bray, J. Paoli, C.M. Sperberg-McQueen, Eve Maler, editors, 6 October 2000.
Available at http://www.w3.org/TR/REC-xml

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

Intro to SVG
Scalable Vector Graphics

§1. Overview

1.1 What is SVG?

Scalable Vector Graphics (SVG) is an XML-based language for representing interactive, 2D vector-graphics documents. SVG is one of many open standards put forth by the World Wide Web Consortium (w3c.org). One fundamental benefit to using SVG is that the graphics don't lose quality when zoomed or resized ("scalable"). Another benefit is the comparatively small size of an SVG document. An author can specify how the SVG content changes over time; all elements, attributes, and style-sheet settings can be animated. The attendee will learn to author standalone and browser-embedded presentations in SVG.

1.2 History of SVG

World Wide Web Consortium: www.w3.org

SVG	First Working Draft	11-Feb-1999
SVG 1.0	W3C Recommendation	04-Sept-2001

At the time of writing,

SVG 1.1	Working Draft in Last Call	15-Feb-2002
Mobile SVG Profiles	Working Draft in Last Call	15-Feb-2002

Order of W3C Validation:

Working Draft in Development
Working Draft in Last Call
Candidate Recommendation
Proposed Recommendation
Recommendation

1.3 Advantages of an XML-based Language

- searchable, machine-understandable
- Document syntax can be validated
- Integrates with other XML languages, such as SMIL
- Non-proprietary format backed by industry

§2. Vector Graphics vs Raster/Bitmapped Graphics

2.1 Raster/Bitmapped Graphics

Image is pre-rendered. Resolution of image is fixed. Enlarging or shrinking image causes image to lose quality. Image is just a set of pixels, no represenation of what the pixels are.

2.2 Vector Graphics

Image is rendered at the appropriate resolution "on demand". Basically the same quality regardless of size of output device (web page, mobile device, etc). Because SVG is XML-based, can attach meaning to each object in the image; could extract info from several documents to create a new document.

2.3 Compare to Flash, which is also vector graphics

SVG Advantages
- non-proprietary format
- text-based
- integrates with other XML-based languages
- searchable.

SVG Disadvantage
- Flash has mature authoring tool.

§3. Players

Internet Explorer 6	http://www.microsoft.com/
Adobe SVG Plug-in	http://www.adobe.com/svg/
RealOnePlayer	http://www.real.com
Amaya	http://www.w3.org/Amaya/

3.1 Embedded document

Example: Display svg document from inside an html page. Adobe plug-in will do the rendering.

```
<html>
  <body>
     <embed src="circ1.svg" name="circ1" type="image/svg-xml"
         width="400" height="400"
         pluginspage="http://www.adobe.com/svg/viewer/install" />
  </body>
</html>
```

3.2 Standalone document

Example: Document is a ".svg" file viewed directly.

```
<svg>
    <!-- content -->
</svg>
```

§4. Basic SVG document structure

1. Content enclosed by **`<svg>`** tag

```
<svg>
    <!-- content -->
</svg>
```

`<svg>` is the top-level element that contains the entire document.
Comments are delimited by **`<!--`** and **`-->`** as in HTML and XML

2. Identify as XML document

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg>
    <!-- content -->
</svg>
```

3. Add the document type declaration:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg>
    <!-- content -->
</svg>
```

4. Add the namespace:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg  xmlns="http://www.w3.org/2000/svg">
    <!-- content -->
</svg>
```

5. (optional, but recommended) Specify the viewport:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg  xmlns=http://www.w3.org/2000/svg
      width="200px" height="150px">
    <!-- content -->
</svg>
```

6. (optional, but recommended) Specify the title of the document:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
     "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg  xmlns="http://www.w3.org/2000/svg"
      width="200px" height="150px">
    <title> My Title Goes Here </title>
    <!-- content -->
</svg>
```

§5. Coordinate systems and units

SVG Canvas
infinite. where drawing occurs.

SVG Viewport
visible area where rendering occurs.
attributes: `width`, `height`
origin: top/left corner

length unit identifiers
`em`, `ex`, `px`, `pt`, `pc`, `cm`, `mm`, `in`, and percentages

§6. Basic shapes and text

Note: For brevity, the XML prolog information has been removed from these examples.

6.1 Attributes common to all shapes

`stroke`	Outline color of shape
`stroke-width`	Thickness of outline
`fill`	Inside color, for closed shapes

6.2 Basic Shapes

6.2.1 **`rect`**

Specify the top-left corner (`x,y`) and the extents (`width`, `height`) of the rectangle.

```
<svg width="400" height="200">
    <title> SVG Basic Shape: Rectangle </title>
    <rect stroke="black" fill="none"
          x="0px" y="0px"
          width="100px"  height="100px" />
</svg>
```

Example: SVG_Ex/basicShapes/rect1.svg

6.2.2 **`circle`**

Specify the center (`cx,cy`) and the radius `r` of the circle.

```
<svg width="400px" height="400px">
    <title> SVG Basic Shape: Circle </title>
    <circle stroke="black" fill="blue"
            cx="200px" cy="200px" r="100px" />
</svg>
```

Example: SVG_Ex/basicShapes/circ1.svg

6.2.3 **`ellipse`**

Specify the center (`cx,cy`) and the radii (`rx, ry`) of the ellipse.

```
<svg width="300" height="300">
    <title> SVG Basic Shape: Ellipse </title>
    <ellipse  stroke="blue" stroke-width="5" fill="none"
              cx="200px" cy="200px"
              rx="50px"  ry="30px" />
</svg>
```

Example: SVG_Ex/basicShapes/ellipse1.svg

6.2.4 `line`

Specify the start point (`x1,y1`) and end point (`x2, y2`).

```
<svg width="400" height="300">
    <title> SVG Basic Shape: Line </title>
    <line stroke="black" x1="10px"  y1="0px"
                         x2="340px" y2="250px" />
</svg>
```

Example: SVG_Ex/basicShapes/line1.svg

6.2.5 `polyline`

Specify the end points along the polyline as `x,y` pairs.

```
<svg width="600" height="400">
    <title> SVG Basic Shape: Polyline </title>
    <polyline fill="none" stroke="darkred" stroke-width="5"
              points= "0,400  100,250  225,300  300,150
                     400,200  500,275  525,210" />
</svg>
```

Example: SVG_Ex/basicShapes/line1.svg

6.2.6 `polygon`

Specify the end points along the polygon as `x,y` pairs. You don't need to specify the starting point twice; the polygon will be a closed shape automatically.

```
<svg width="400" height="400">
    <title> SVG Basic Shape: Polygon </title>
    <polygon fill="blueviolet" stroke="gold" stroke-width="7"
             points="0,0 100,50 200,0 200,200 100,150 0,200" />
</svg>
```

Example: SVG_Ex/basicShapes/polygon1.svg

6.3 Text

Example: SVG_Ex/basicShapes/text1.svg

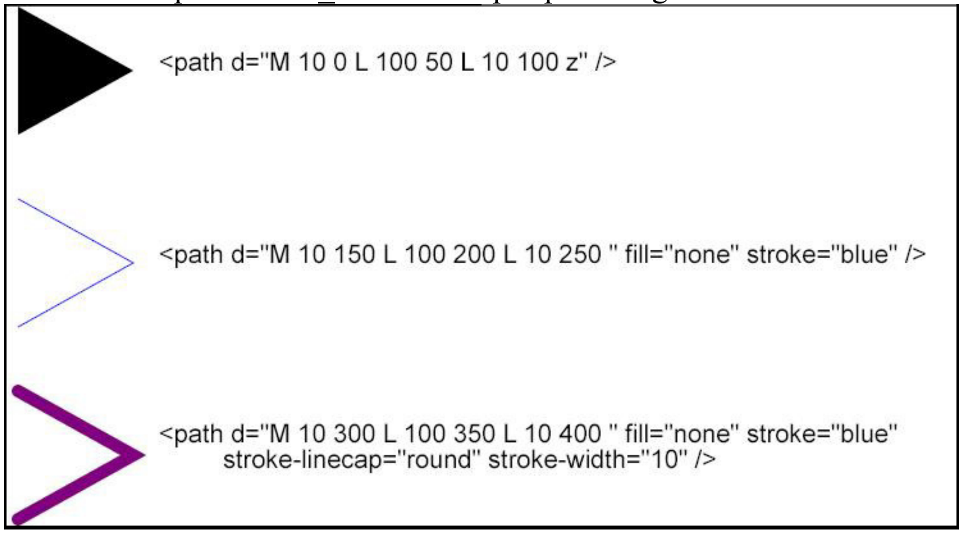
```
<svg width="300" height="300" >
     <text x="20" y="100"
           font-size="26" fill="darkcyan"> Hello, World </text>
</svg>
```

6.4 Paths

A path may be closed or open.

M	moveto
L	lineto
z	closepath

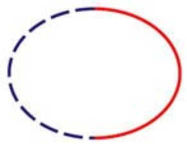
Example 1: SVG_Ex/basicShapes/path1.svg



```
<path d="M 10 0 L 100 50 L 10 100 z" />

<path d="M 10 150 L 100 200 L 10 250 " fill="none" stroke="blue" />

<path d="M 10 300 L 100 350 L 10 400 " fill="none" stroke="blue"
      stroke-linecap="round" stroke-width="10" />
```

Example 2: SVG_Ex/basicShapes/path2.svg

```
<svg width="600" height="600" >
     <path d="M 300 200 a100,75 0 0,1 0,200" stroke="red"
           stroke-width="5" fill="none" />
     <path d="M 300 200 a100,75 0 0,0 0,200"
           stroke="midnightblue"
           stroke-dasharray="30,10"
           stroke-width="5" fill="none" />
</svg>
```



§7. Painting

Excerpt From SVG W3C Recommendation:

7.1 Recognized color keyword names

The following is the list of recognized color keywords that can be used as a keyword value for data type <color>:

aliceblue	rgb(240, 248, 255)	lightpink	rgb(255, 182, 193)
antiquewhite	rgb(250, 235, 215)	lightsalmon	rgb(255, 160, 122)
aqua	rgb(0, 255, 255)	lightseagreen	rgb(32, 178, 170)
aquamarine	rgb(127, 255, 212)	lightskyblue	rgb(135, 206, 250)
azure	rgb(240, 255, 255)	lightslategray	rgb(119, 136, 153)
beige	rgb(245, 245, 220)	lightslategrey	rgb(119, 136, 153)
bisque	rgb(255, 228, 196)	lightsteelblue	rgb(176, 196, 222)
black	rgb(0, 0, 0)	lightyellow	rgb(255, 255, 224)
blanchedalmond	rgb(255, 235, 205)	lime	rgb(0, 255, 0)
blue	rgb(0, 0, 255)	limegreen	rgb(50, 205, 50)
blueviolet	rgb(138, 43, 226)	linen	rgb(250, 240, 230)
brown	rgb(165, 42, 42)	magenta	rgb(255, 0, 255)
burlywood	rgb(222, 184, 135)	maroon	rgb(128, 0, 0)
cadetblue	rgb(95, 158, 160)	mediumaquamarine	rgb(102, 205, 170)
chartreuse	rgb(127, 255, 0)	mediumblue	rgb(0, 0, 205)
chocolate	rgb(210, 105, 30)	mediumorchid	rgb(186, 85, 211)
coral	rgb(255, 127, 80)	mediumpurple	rgb(147, 112, 219)
cornflowerblue	rgb(100, 149, 237)	mediumseagreen	rgb(60, 179, 113)
cornsilk	rgb(255, 248, 220)	mediumslateblue	rgb(123, 104, 238)
crimson	rgb(220, 20, 60)	mediumspringgreen	rgb(0, 250, 154)
cyan	rgb(0, 255, 255)	mediumturquoise	rgb(72, 209, 204)
darkblue	rgb(0, 0, 139)	mediumvioletred	rgb(199, 21, 133)
darkcyan	rgb(0, 139, 139)	midnightblue	rgb(25, 25, 112)
darkgoldenrod	rgb(184, 134, 11)	mintcream	rgb(245, 255, 250)
darkgray	rgb(169, 169, 169)	mistyrose	rgb(255, 228, 225)
darkgreen	rgb(0, 100, 0)	moccasin	rgb(255, 228, 181)
darkgrey	rgb(169, 169, 169)	navajowhite	rgb(255, 222, 173)
darkkhaki	rgb(189, 183, 107)	navy	rgb(0, 0, 128)

darkmagenta	rgb(139, 0, 139)	oldlace	rgb(253, 245, 230)
darkolivegreen	rgb(85, 107, 47)	olive	rgb(128, 128, 0)
darkorange	rgb(255, 140, 0)	olivedrab	rgb(107, 142, 35)
darkorchid	rgb(153, 50, 204)	orange	rgb(255, 165, 0)
darkred	rgb(139, 0, 0)	orangered	rgb(255, 69, 0)
darksalmon	rgb(233, 150, 122)	orchid	rgb(218, 112, 214)
darkseagreen	rgb(143, 188, 143)	palegoldenrod	rgb(238, 232, 170)
darkslateblue	rgb(72, 61, 139)	palegreen	rgb(152, 251, 152)
darkslategray	rgb(47, 79, 79)	paleturquoise	rgb(175, 238, 238)
darkslategrey	rgb(47, 79, 79)	palevioletred	rgb(219, 112, 147)
darkturquoise	rgb(0, 206, 209)	papayawhip	rgb(255, 239, 213)
darkviolet	rgb(148, 0, 211)	peachpuff	rgb(255, 218, 185)
deeppink	rgb(255, 20, 147)	peru	rgb(205, 133, 63)
deepskyblue	rgb(0, 191, 255)	pink	rgb(255, 192, 203)
dimgray	rgb(105, 105, 105)	plum	rgb(221, 160, 221)
dimgrey	rgb(105, 105, 105)	powderblue	rgb(176, 224, 230)
dodgerblue	rgb(30, 144, 255)	purple	rgb(128, 0, 128)
firebrick	rgb(178, 34, 34)	red	rgb(255, 0, 0)
floralwhite	rgb(255, 250, 240)	rosybrown	rgb(188, 143, 143)
forestgreen	rgb(34, 139, 34)	royalblue	rgb(65, 105, 225)
fuchsia	rgb(255, 0, 255)	saddlebrown	rgb(139, 69, 19)
gainsboro	rgb(220, 220, 220)	salmon	rgb(250, 128, 114)
ghostwhite	rgb(248, 248, 255)	sandybrown	rgb(244, 164, 96)
gold	rgb(255, 215, 0)	seagreen	rgb(46, 139, 87)
goldenrod	rgb(218, 165, 32)	seashell	rgb(255, 245, 238)
gray	rgb(128, 128, 128)	sienna	rgb(160, 82, 45)
grey	rgb(128, 128, 128)	silver	rgb(192, 192, 192)
green	rgb(0, 128, 0)	skyblue	rgb(135, 206, 235)
greenyellow	rgb(173, 255, 47)	slateblue	rgb(106, 90, 205)
honeydew	rgb(240, 255, 240)	slategray	rgb(112, 128, 144)
hotpink	rgb(255, 105, 180)	slategrey	rgb(112, 128, 144)
indianred	rgb(205, 92, 92)	snow	rgb(255, 250, 250)
indigo	rgb(75, 0, 130)	springgreen	rgb(0, 255, 127)
ivory	rgb(255, 255, 240)	steelblue	rgb(70, 130, 180)
khaki	rgb(240, 230, 140)	tan	rgb(210, 180, 140)

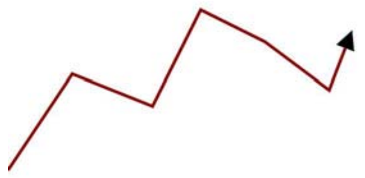
lavender	rgb(230, 230, 250)	teal	rgb(0, 128, 128)
lavenderblush	rgb(255, 240, 245)	thistle	rgb(216, 191, 216)
lawngreen	rgb(124, 252, 0)	tomato	rgb(255, 99, 71)
lemonchiffon	rgb(255, 250, 205)	turquoise	rgb(64, 224, 208)
lightblue	rgb(173, 216, 230)	violet	rgb(238, 130, 238)
lightcoral	rgb(240, 128, 128)	wheat	rgb(245, 222, 179)
lightcyan	rgb(224, 255, 255)	white	rgb(255, 255, 255)
lightgoldenrodyellow	rgb(250, 250, 210)	whitesmoke	rgb(245, 245, 245)
lightgray	rgb(211, 211, 211)	yellow	rgb(255, 255, 0)
lightgreen	rgb(144, 238, 144)	yellowgreen	rgb(154, 205, 50)
lightgrey	rgb(211, 211, 211)		

7.2 Fill/Stroke

single color
gradient
pattern

7.3 Markers

```
<svg xmlns="http://www.w3.org/2000/svg">
   <defs>
       <marker id="Tri" viewBox="0 0 10 10" refX="0" refY="5"
               markerUnits="strokeWidth" markerWidth="6"
               markerHeight="6" orient="auto">
              <path d="M 0 0 L 10 5 L 0 10 z" />
       </marker>
   </defs>

   <polyline fill="none" stroke="darkred" stroke-width="5"
             marker-end="url(#Tri)"
             points="0,400 100,250 225,300 300,150 400,200
                   500,275 525,210" />
</svg>
```



Example: SVG_Ex/polyline2.svg

§8. Interactivity, scripting, and event handling

8.1 Events

focusin
focusout
activate
click
mousedown
mouseup
mouseover
mousemove
mouseout

8.2 ECMAScript

Uses Document object model (DOM)
ECMAScript Binding for SVG
ECMAScript Language Binding

Example: SVG_Ex/events/mouseover.svg

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
     "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg">
  <script type="text/ecmascript"> <![CDATA[
      function text_on(evt,which) {
         title = evt.target.ownerDocument.getElementById(which);
         title.setAttribute("visibility", "visible");
      }
   ]]> </script>
  <script type="text/ecmascript"> <![CDATA[
      function text_off(evt,which) {
         title = evt.target.ownerDocument.getElementById(which);
         title.setAttribute("visibility", "hidden");
      }
   ]]> </script>

  <text onmouseover="text_on(evt,'hints')"
        onmouseout="text_off(evt,'hints')"
        x="75px" y="50px">MouseOverMe</text>
  <g id="hints" visibility="hidden">
    <text x="175px" y="20px" >On mouse over, </text>
    <text x="175px" y="40px">this text becomes visible </text>
  </g>

</svg>
```

Example: SVG_Ex/events/mouseover2.svg

On mouse over the text, replace the text displayed. This involves modifying the content in the Document Object Model.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
     "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg">
  <script type="text/ecmascript"> <![CDATA[
      function text_on(evt,which) {
         doc = evt.target.ownerDocument;
         title = doc.getElementById(which);
         node = title.getChildNodes().item(0);
         node.setData(7);
      }
   ]]> </script>

  <text id="me" onmouseover="text_on(evt,'me')"
        x="75px" y="50px">MouseOverMe
  </text>
</svg>
```

§9. Hyperlinking

Uniform Resource Identifier (uri)
Similar to HTML <a>

§10. Animation

From SMIL
- animate
- set
- animateMotion
- animateColor

SVG extensions to SMIL animation: animateTransform

```
<svg>
    <title> Animate Text Across the Screen </title>
    <text font-size="20"> &lt;animateMotion path="M 400 50 L 10 50"
          begin="0s" dur="10s" fill="freeze" />
          <animateMotion path="M 400 50 L 10 50" begin="0s"
                         dur="10s" fill="freeze" />
  </text>
</svg>
```

§11. Defs

Definitions that will be used elsewhere in the document

`symbol`: graphical template that be instantiated by **`'use'`**

§12. Grouping: container element <g>

- attributes are inherited by all contained elements

Example: SVG_Ex/basicShapes/path2_group.svg

```
<svg width="600" height="600">
    <g stroke-width="5" fill="none">
         <path d="M 300 200 a100,75 0 0,1 0,200" stroke="red"  />
         <path d="M 300 200 a100,75 0 0,0 0,200"
               stroke="midnightblue" stroke-dasharray="30,10" />
    </g>
</svg>
```

Example:

```
<g font-size="20">
     <text x="20px" y="150px"> Item 1 </text>
     <text x="70px" y="170px"> Item 2 </text>
</g>
```

§13. Metadata (data about data)

Generalized version of HTML's meta

```
<meta name="DC.Creator" content="Kathy Barshatzky" />
<meta name="DC.Rights"  content="SAIC" />
<meta name="DC.Date"    content="2001-12-10" />
```

RDF (Resource Description Framework), W3C Recommendation, Feb 1999

- framework: doesn't define any properties.
- Statement-based

▪ Resource	(the thing being described. URI),
▪ Property	(describes Resource. has a name.)
▪ Statement	(Resource, Property, and value)

  Statement: The Creator [Property] of svgIntro.html [Resource] is Kathy Barshatzky [value].

- Need a properties package, such as the 15 core elements from the Dublin Core Metadata Initiative. See dublincore.org (`xmlns:dc`).

```
<metadata>
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/">

      <rdf:Description rdf:about="2002CourseNotes_3_SVG.doc">
          <dc:title> Intro to SVG </dc:title>
          <dc:creator> Kathy Barshatzky </dc:creator>
          <dc:rights> javakathy.com </dc:rights>
          <dc:date> 2002-04-05 </dc:date>
          <dc:language>en</dc:language>
          <dc:publisher>SIGGRAPH</dc:publisher></rdf:Description>
   </rdf:RDF>
</metadata>
```

§14. Requirements for Future Versions of SVG

SVG 1.1/2.0 Requirements: Coordinates and Transformations

"SVG should allow elements to be defined in the coordinate system used by the view port. SVG 1.0 only allows elements to be defined in the user coordinate system, ensuring they are always affected by the current user space to view port transformation. Many applications, such as user interfaces, require objects that are not affected by the user space transformation, i.e. their position and size remain constant. Examples of such applications are the legend on a chart, symbols on a map and buttons in a user interface. [SVG 1.1] [SVG 2.0]"

SVG 1.1/2.0: Grouping

"SVG may provide a mechanism to control rendering order, such as a "z-index" attribute. [SVG 2.0]
SVG may provide the concept of layers. [SVG 2.0]"

§15. Accessibility Issues

See http://www.w3.org/TR/SVG-access/ for a discussion and examples on how to make your presentation accessible to people with special needs.

Appendix

§Appendix-1. Reference Sites

I. Recommendations/Specifications

www.w3.org/MarkUp	HTML
www.w3.org/AudioVideo	SMIL
www.w3.org/XML	XML
www.w3.org/Style/CSS	Cascading Style Sheets
www.w3.org/Style/XSL	XML Style Sheets

The following W3C recommendations are available for reference on the CD in the folder. Please check the sites for the latest revisions. Credit and copyrights belong to the original authors.

Synchronized Multimedia Integration Language

```
http://www.w3.org/TR/REC-smil/         SMIL 1.0
http://www.w3.org/TR/smil20/           SMIL 2.0
   References/SMIL/smil20_DTD
   References/SMIL/smil20_spec
```

XHTML: The Extensible HyperText Markup Language

```
http://www.w3.org/TR/xhtml1/
   References/XHTML/xhtml10.pdf
   References/XHTML/xhtml11.pdf
   References/XHTML/xhtml-basic.pdf
```

XML: Extensible Markup Language

```
http://www.w3.org/XML
   References/XML/REC-xml-20001006.pdf
```

Namespaces in XML

```
http://www.w3.org/TR/1999/REC-xml-names-19990114/
   References/XML/NamespacesInXML.htm
```

MIME(Multipurpose Internet Mail Extensions) types

```
http://www.oac.uci.edu/indiv/ehood/MIME/
```

Internet Engineering Task Force

```
http://www.ietf.org/
```

II. Tutorials and More References

CWI (Centrum voor Wiskunde en Informatica)
Excellent SMIL presentation by Lloyd Rutledge and Lynda Hardman, members of the W3C working group that developed SMIL:
http://www.cwi.nl/~media/SMIL/Tutorial/
http://www.cwi.nl/~media/publications/SMILTutorial.pdf

RealSystem G2 Syntax Style
http://service.real.com/help/library/blueprints/stylehtml/syntax.htm

RealText Authoring Guide
This guide tells how to create and stream RealText. Last update: December 15, 2000
http://docs.real.com/docs/smil/realtextauthoringguide8.pdf

http://service.real.com/help/library/encoders.html
http://service.real.com/help/library/guides/realtext/realtext.htm

Cascading Style Sheets
http://www.w3.org/Style/CSS/
http://www.w3.org/MarkUp/Guide/Style.html

Multipurpose Internet Mail Extensions (as in MIME content-type)
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1521.html

Helio. Creators of Java-based SMIL Player, SOJA.
www.helio.org/products/smil/tutorial/toc.html

QuickTime-specific authoring
www.apple.com/quicktime/authoring/qtsmil.html

III. W3C Maillists

There are many mailing lists provided by the W3C for discussion and development on the World Wide Web. A full list of them is available at:

http://www.w3.org/Mail/Lists

Administrative requests should be sent to the request address:

www-smil-request@w3.org
www-svg-request@w3.org

The -request mail address accepts the following commands (in the Subject of an e-mail message):

`subscribe`	Subscribe to the list. If you want to subscribe under a different address, use a Reply-To: address header in the message.
`unsubscribe`	Unsubscribe from the list.
`help`	Get information about the mailing list.
`archive help`	Get information about the list archive(s).

Online Archives

Every submission sent to this list is archived and made available online. Archives of public lists are available at:

http://lists.w3.org/Archives/Public/

The application/xhtml+xml Media Type
RFC 3236

Network Working Group   M. Baker
Request for Comments: 3236   Planetfred, Inc.
Category: Informational   P. Stark
                                         Ericsson Mobile Communications
                                                           January 2002

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document defines the 'application/xhtml+xml' MIME media type for XHTML based markup languages; it is not intended to obsolete any previous IETF documents, in particular RFC 2854 which registers 'text/html'.

1. Introduction

In 1998, the W3C HTML working group began work on reformulating HTML in terms of XML 1.0 [XML] and XML Namespaces [XMLNS]. The first part of that work concluded in January 2000 with the publication of the XHTML 1.0 Recommendation [XHTML1], the reformulation for HTML 4.01 [HTML401].

Work continues in the Modularization of XHTML Recommendation [XHTMLM12N], the decomposition of XHTML 1.0 into modules that can be used to compose new XHTML based languages, plus a framework for supporting this composition.

This document only registers a new MIME media type,'application/xhtml+xml'. It does not define anything more than is required to perform this registration.

This document follows the convention set out in [XMLMIME] for the MIME subtype name; attaching the suffix "+xml" to denote that the entity being described conforms to the XML syntax as defined in XML 1.0 [XML].

This document was prepared by members of the W3C HTML working group based on the structure, and some of the content, of RFC 2854, the registration of 'text/html'. Please send comments to www-html@w3.org, a public mailing list (requiring subscription) with archives at <http://lists.w3.org/Archives/Public/www-html/>.

2. Registration of MIME media type application/xhtml+xml

MIME media type name:	application
MIME subtype name:	xhtml+xml
Required parameters:	none

Optional parameters:

charset

This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in [XMLMIME].

profile

See Section 8 of this document.

Encoding considerations:

See Section 4 of this document.

Security considerations:

See Section 7 of this document.

Interoperability considerations:

XHTML 1.0 [XHTML10] specifies user agent conformance rules that dictate behaviour that must be followed when dealing with, among other things, unrecognized elements.

With respect to XHTML Modularization [XHTMLMOD] and the existence of XHTML based languages (referred to as XHTML family members) that are not XHTML 1.0 conformant languages, it is possible that 'application/xhtml+xml' may be used to describe some of these documents. However, it should suffice for now for the purposes of interoperability that user agents accepting 'application/xhtml+xml' content use the user agent conformance rules in [XHTML1].

Although conformant 'application/xhtml+xml' interpreters can expect that content received is well-formed XML (as defined in [XML]), it cannot be guaranteed that the content is valid XHTML (as defined in [XHTML1]). This is in large part due to the reasons in the preceding paragraph.

Published specification:

XHTML 1.0 is now defined by W3C Recommendation; the latest published version is [XHTML1]. It provides for the description of some types of conformant content as "text/html", but also doesn't disallow the use with other content types (effectively allowing for the possibility of this new type).

Applications which use this media type:

Some content authors have already begun hand and tool authoring on the Web with XHTML 1.0. However that content is currently described as "text/html", allowing existing Web browsers to process it without reconfiguration for a new media type.

There is no experimental, vendor specific, or personal tree predecessor to 'application/xhtml+xml'. This new type is being registered in order to allow for the expected deployment of XHTML on the World Wide Web, as a first class XML application where authors can expect that user agents are conformant XML 1.0 [XML] processors.

Additional information:

Magic number:
There is no single initial byte sequence that is always present for XHTML files. However, Section 5 below gives some guidelines for recognizing XHTML files. See also section 3.1 in [XMLMIME].

File extension:
There are three known file extensions that are currently in use for XHTML 1.0; ".xht", ".xhtml", and ".html".

It is not recommended that the ".xml" extension (defined in [XMLMIME]) be used, as web servers may be configured to distribute such content as type "text/xml" or "application/xml". [XMLMIME] discusses the unreliability of this approach in section 3. Of course, should the author desire this behaviour, then the ".xml" extension can be used.

Macintosh File Type code: TEXT

Person & email address to contact for further information:
Mark Baker <mark.baker@canada.sun.com>

Intended usage: COMMON

Author/Change controller:
The XHTML specifications are a work product of the World Wide Web Consortium's HTML Working Group. The W3C has change control over these specifications.

3. Fragment identifiers

URI references (Uniform Resource Identifiers, see [RFC2396] as updated by [RFC2732]) may contain additional reference information, identifying a certain portion of the resource. These URI references end with a number sign ("#") followed by an identifier for this portion (called the "fragment identifier"). Interpretation of fragment identifiers is dependent on the media type of the retrieval result.

For documents labeled as 'text/html', [RFC2854] specified that the fragment identifier designates the correspondingly named element, these were identified by either a unique id attribute or a name attribute for some elements. For documents described with the application/xhtml+xml media type, fragment identifiers share the same syntax and semantics with other XML documents, see [XMLMIME], section 5.

At the time of writing, [XMLMIME] does not define syntax and semantics of fragment identifiers, but refers to "XML Pointer Language (XPointer)" for a future XML fragment identification mechanism. The current specification for XPointer is available at http://www.w3.org/TR/xptr. Until [XMLMIME] gets updated, fragment identifiers for XHTML documents designate the element with the corresponding ID attribute value (see [XML] section 3.3.1); any XHTML element with the "id" attribute.

4. Encoding considerations

By virtue of XHTML content being XML, it has the same considerations when sent as 'application/xhtml+xml' as does XML. See [XMLMIME], section 3.2.

5. Recognizing XHTML files

All XHTML documents will have the string "**<html**" near the beginning of the document. Some will also begin with an XML declaration which begins with "<?xml", though that alone does not indicate an XHTML document. All conforming XHTML 1.0 documents will include an XML document type declaration with the root element type 'html'.

XHTML Modularization provides a naming convention by which a public identifier for an external subset in the document type declaration of a conforming document will contain the string "//DTD XHTML". And while some XHTML based languages require the doctype declaration to occur within documents of that type, such as XHTML 1.0, or XHTML Basic (http://www.w3.org/TR/xhtml-basic), it is not the case that all XHTML based languages will include it.

All XHTML files should also include a declaration of the XHTML namespace. This should appear shortly after the string "<html", and should read **'xmlns="http://www.w3.org/1999/xhtml"'**.

6. Charset default rules

By virtue of all XHTML content being XML, it has the same considerations when sent as 'application/xhtml+xml' as does XML. See [XMLMIME], section 3.2.

7. Security Considerations

The considerations for "text/html" as specified in [TEXTHTML] and for 'application/xml' as specified in [XMLMIME], also hold for 'application/xhtml+xml'.

In addition, because of the extensibility features for XHTML as provided by XHTML Modularization, it is possible that 'application/xhtml+xml' may describe content that has security implications beyond those described here. However, if the user agent follows the user agent conformance rules in [XHTML1], this content will be ignored. Only in the case where the user agent recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

8. The "profile" optional parameter

This parameter is meant to solve the short-term problem of using MIME media type based content negotiation (such as that done with the HTTP "Accept" header) to negotiate for a variety of XHTML based languages. It is intended to be used only during content negotiation. It is not expected that it be used to deliver content, or that origin web servers have any knowledge of it (though they are welcome to). It is primarily targeted for use on the network by proxies in the HTTP chain that manipulate data formats (such as transcoders).

The parameter is intended to closely match the semantics of the "profile" attribute of the HEAD element as defined in [HTML401] (section 7.4.4.3), except it is applied to the document as a whole rather than just the META elements. More specifically, the value of the profile attribute is a URI that can be used as a name to identify a language. Though the URI need not be resolved in order to be useful as a name, it could be a namespace, schema, or a language specification.

As an example, user agents supporting only XHTML Basic (see http://www.w3.org/TR/xhtml-basic) currently have no standard means to convey their inability to support the additional functionality in XHTML 1.0 [XHTML1] that is not found in XHTML Basic. While XHTML Basic user agent conformance rules (which are identical to XHTML 1.0) provide some guidance to its user agent implementators for handling some additional content, the additional content in XHTML 1.0 that is not part of XHTML Basic is substantial, making those conformance rules insufficient for practical processing and rendering to the end user. There is also the matter of the potentially substantial burden on the user agent in receiving and parsing this additional content.

The functionality afforded by this parameter can also be achieved with at least two other more general content description frameworks; the "Content-features" MIME header described in RFC 2912, and UAPROF from the WAPforum (see http://www.wapforum.org/what/technical.htm). At this time, choosing one of these solutions would require excluding the other, as interoperability between the two has not been defined. For this reason, it is suggested that this parameter be used until such time as that issue has been addressed.

An example use of this parameter as part of a HTTP GET transaction would be;

```
Accept: application/xhtml+xml;
  profile="http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd"
```

9. Author's Address

Mark A. Baker
Planetfred, Inc.
44 Byward Market, Suite 240
Ottawa, Ontario, CANADA. K1N 7A2
Phone: +1-613-789-1818
EMail: mbaker@planetfred.com
EMail: distobj@acm.org

Peter Stark
Ericsson Mobile Communications
Phone: +464-619-3000
EMail: Peter.Stark@ecs.ericsson.com

10. References

[HTML401] Raggett, D., et al., "HTML 4.01 Specification", W3C Recommendation. Available at <http://www.w3.org/TR/html401>

[MIME] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[URI] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[XHTML1] "XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4 in XML 1.0", W3C Recommendation. Available at <http://www.w3.org/TR/xhtml1>.

[XML] "Extensible Markup Language (XML) 1.0", W3C Recommendation. Available at <http://www.w3.org/TR/REC-xml>

[TEXTHTML] Connolly, D. and L. Masinter, "The 'text/html' Media Type", RFC 2854, June 2000.

[XMLMIME] Murata, M., St.Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[XHTMLM12N] "Modularization of XHTML", W3C Recommendation. Available at: <http://www.w3.org/TR/xhtml-modularization>

11. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

The `application/smil` and `application/smil+xml` Media Types

Network Working Group P. Hoschka
INTERNET DRAFT W3C
draft-hoschka-smil-media-type-10.txt March 2002

Status of this Memo

Abstract

This document specifies the Media Type for version 1 and version 2 of the Synchronized Multimedia Integration Language (SMIL 1.0 and SMIL 2.0). SMIL allows integrating a set of independent multimedia objects into a synchronized multimedia presentation.

1. Introduction

The World Wide Web Consortium has issued specifications which define version 1 [1] and version 2 [2] of the Synchronized Multimedia Integration Language (SMIL). This memo provides information about the application/smil and application/smil+xml Media Types.

The definition is based on RFC3023 defining the use of the "application/xml" media type [3]. Before using the "application/smil" or "application/smil+xml" media type, implementors must thus be familiar with [3].

2. Synchronized Multimedia Integration Language

SMIL allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can

1.describe the temporal behavior of the presentation
2.describe the layout of the presentation on a screen
3.associate hyperlinks with media objects

4.define conditional content inclusion/exclusion based on system/network properties

3. Registration Information

```
To: ietf-types@iana.org
Subject: Registration of MIME media type application/smil
```

MIME media type name:	application
MIME subtype name:	smil
Required parameters:	none

Optional parameters:

charset

All of the considerations described in RFC3023 also apply to the SMIL media type.

profile

See Section 5 of this document.

Encoding considerations:

All of the considerations described in RFC3023 also apply to the SMIL media type.

Security considerations:

SMIL documents contain a construct that allows "infinite loops". This is indispensable for a multimedia format. However, SMIL clients should foresee provisions such as a "stop" button that lets users interrupt such an "infinite loop".

As with HTML, SMIL documents contain links to other media (images,sounds, videos, text, ...) and those links are typically followed automatically by software, resulting in the transfer of files without the explicit request of the user for each one. The security considerations of each linked file are those of the individual registered types.

The SMIL language contains "switch" elements. SMIL provides no mechanism that assures the media objects contained in a "switch" element provide equivalent information. An author, knowing that one SMIL player will display one alternative of a "switch" and another will display a different part, can put different information in the two parts. While there are legitimate use cases for this, this also gives rise to a security consideration: The author can fool viewers into thinking that the same information was displayed when in fact it was not.

In addition, all of the security considerations of RFC3023 also apply to SMIL.

Interoperability considerations:

SMIL documents contain links to other media objects. The SMIL player must be able to decode the media types of these media in order to display the whole document. To increase interoperability, SMIL has provisions for including alternate versions of a media object in a document.

Published specification: see [1] and [2]

Applications which use this media type:

SMIL players and editors

Additional information:

Semantics of fragment identifiers in URIs: The SMIL media type allows to append a fragment identifier to a URI pointing to a SMIL resource (e.g. http://www.example.com/test.smil#foo). The semantics of fragment identifiers for SMIL resources are defined in [2].

Magic number(s):
There is no single initial byte sequence that is always present for SMIL files. However, Section 4 below gives some guidelines for recognizing SMIL files.

File extension(s):
.smil, .smi, .sml

NOTE: On the Windows operating system and the Macintosh platform, the ".smi" extension is used by other formats. To avoid conflicts, it is thus recommended to use the extension ".smil" for storing SMIL files on these platforms.

Macintosh File Type Code(s): "TEXT", ".SMI", "SMIL"
Object Identifier(s) or OID(s): none

Person & email address to contact for further information:
The author of this memo.

Intended usage: COMMON

Author/Change controller:
The SMIL 1.0 and SMIL 2.0 specifications are a work product of the World Wide Web Consortium's SYMM Working Group.

The W3C has change control over both specifications.

```
To: ietf-types@iana.org Subject: Registration of MIME media type
application/smil+xml
```

MIME media type name: application
MIME subtype name: smil+xml

Required parameters:	see registration of application/smil
Optional parameters:	see registration of application/smil
Encoding considerations:	see registration of application/smil
Security considerations:	see registration of application/smil
Interoperability considerations:	see registration of application/smil
Published specification:	see registration of application/smil
Applications which use this media type:	see registration ofapplication/smil
Additional information:	see registration of application/smil
Magic number(s):	see registration of application/smil
File extension(s):	see registration of application/smil
Macintosh File Type Code(s):	see registration of application/smil

Object Identifier(s) or OID(s):	see registration of `application/smil`
Person & email address to contact for further information:	see registration of `application/smil`
Intended usage:	see registration of `application/smil`
Author/Change controller:	see registration of `application/smil`

4. Recognizing SMIL files

All SMIL files will have the string "**<smil**" near the beginning of the file. Some will also begin with an XML declaration which begins with "<?xml", though that alone does not indicate a SMIL document.

All SMIL 2.0 files must also include a declaration of the SMIL 2.0 namespace. This should appear shortly after the string "<smil", and should read **'xmlns="http://www.w3.org/2001/SMIL20/PR/Language"'**.

5. The "profile" optional parameter

This parameter is meant to be used in MIME media type based content negotiation (such as that done with the HTTP "Accept" header) to negotiate for a variety of SMIL based languages. It is modelled after the "profile" parameter in the application/xhtml+smil MIME type registration [4], and is motivated by very similar considerations.

The parameter is intended to be used only during content negotiation. It is not expected that it be used to deliver content, or that origin web servers have any knowledge of it (though they are welcome to). It is primarily targetted for use on the network by proxies in the HTTP chain that manipulate data formats (such as transcoders).

The value of the profile attribute is a URI that can be used as a name to identify a language. Though the URI need not be resolved in order to be useful as a name, it could be a namespace, schema, or a language specification.

As an example, user agents supporting only SMIL Basic (see http://www.w3.org/TR/smil20/smil-basic.html) currently have no standard means to convey their inability to fully support SMIL 2.0. While SMIL 2.0 Basic user agents are required to parse the full SMIL 2.0 language, there is potentially a substantial burden in receiving and parsing document content that will not be presented to the user, since its functionality is not included in SMIL Basic.

In future, the functionality afforded by this parameter will also be achievable by the emerging CC/PP framework [5]. It is suggested that the "profile" parameter be used until the CC/PP framework has been finalized.

An example use of this parameter as part of a HTTP GET transaction would be:

```
Accept: application/smil+xml;
   profile="http://www.w3.org/2001/SMIL20/REC/HostLanguage"
```

6. References

[1] "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", W3C Recommendation REC-smil-19980615, http://www.w3.org/TR/1998/REC-smil/, July 1998.

[2] "Synchronized Multimedia Integration Language (SMIL) 2.0 Specification", W3C Recommendation, http://www.w3.org/TR/smil20/, August 2001.

[3] M. Murata, S. St.Laurent, D. Kohn E. "XML Media Types", RFC 3023, January 2001.

[4] M. Baker. "The 'application/xhtml+xml' Media Type", Internet Draft draft-baker-xhtml-media-reg-01.txt, February 2001.

[5] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto. "Composite Capability/ Preferences Profiles (CC/PP): Structure and Vocabularies", W3C Working Draft http://www.w3.org/TR/CCPP-struct-vocab/, March 2001.

6. Author's Address

Philipp Hoschka
W3C/INRIA
2004, route des Lucioles - B.P. 93
06902 Sophia Antipolis Cedex
FRANCE

Phone: +33 (0)492387984
Fax:+33 (0)493657765
EMail: ph@w3.org

Registered Media Types

[RFC2045,RFC2046] specifies that Content Types, Content Subtypes, CharacterSets, Access Types, and conversion values for MIME mail will be assigned and listed by the IANA.

Most recent update/additions are listed in boldface type.

Content Types and Subtypes

Type	Subtype	Description	Reference
text	plain		[RFC2646,RFC2046]
	richtext		[RFC2045,RFC2046]
	enriched		[RFC1896]
	tab-separated-values		[Paul Lindner]
	html		[RFC2854]
	sgml		[RFC1874]
	vnd.latex-z		[Lubos]
	vnd.fmi.flexstor		[Hurtta]
	uri-list		[RFC2483]
	vnd.abc		[Allen]
	rfc822-headers		[RFC1892]
	vnd.in3d.3dml		[Powers]
	prs.lines.tag		[Lines]
	vnd.in3d.spot		[Powers]
	css		[RFC2318]
	xml		**[RFC3023]**
	xml-external-parsed-entity		**[RFC3023]**
	rtf		[Lindner]
	directory		[RFC2425]
	calendar		[RFC2445]
	vnd.wap.wml		[Stark]
	vnd.wap.wmlscript		[Stark]
	vnd.motorola.reflex		[Patton]
	vnd.fly		[Gurney]
	vnd.wap.sl		[WAP-Forum]
	vnd.wap.si		[WAP-Forum]
	t140		[RFC2793]
	vnd.ms-mediapackage		[Nelson]
	vnd.IPTC.NewsML		[IPTC]
	vnd.IPTC.NITF		[IPTC]
	vnd.curl		[Hodge]
	vnd.DMClientScript		[Bradley]
	parityfec		[RFC3009]

Type	Subtype	Description	Reference
multipart	mixed		[RFC2045,RFC2046]
	alternative		[RFC2045,RFC2046]
	digest		[RFC2045,RFC2046]
	parallel		[RFC2045,RFC2046]
	appledouble		[MacMime,Patrik Faltstrom]
	header-set		[Dave Crocker]
	form-data		[RFC2388]
	related		[RFC2387]
	report		[RFC1892]
	voice-message		[RFC2421,RFC2423]
	signed		[RFC1847]
	encrypted		[RFC1847]
	byteranges		[RFC2068]
message	rfc822		[RFC2045,RFC2046]
	partial		[RFC2045,RFC2046]
	external-body		[RFC2045,RFC2046]
	news		[RFC 1036, Henry Spencer]
	http		[RFC2616]
	delivery-status		[RFC1894]
	disposition-notification		[RFC2298]
	s-http		[RFC2660]
application	octet-stream		[RFC2045,RFC2046]
	postscript		[RFC2045,RFC2046]
	oda		[RFC2045,RFC2046]
	atomicmail		[atomicmail,Borenstein]
	andrew-inset		[andrew-inset,Borenstein]
	slate		[slate,terry crowley]
	wita		[Wang Info Transfer,Larry Campbell]
	dec-dx		[Digital Doc Trans, Larry Campbell]
	dca-rft		[IBM Doc Content Arch, Larry Campbell]
	activemessage		[Ehud Shapiro]
	rtf		[Paul Lindner]
	applefile		[MacMime,Patrik Faltstrom]
	mac-binhex40		[MacMime,Patrik Faltstrom]
	news-message-id		[RFC1036, Henry Spencer]
	news-transmission		[RFC1036, Henry Spencer]
	wordperfect5.1		[Paul Lindner]
	pdf		[Paul Lindner]
	zip		[Paul Lindner]
	macwriteii		[Paul Lindner]
	msword		[Paul Lindner]
	remote-printing		[RFC1486,Rose]
	mathematica		[Van Nostern]
	cybercash		[Eastlake]
	commonground		[Glazer]
	iges		[Parks]
	riscos		[Smith]
	eshop		[Katz]
	x400-bp		[RFC1494]
	sgml		[RFC1874]
	cals-1840		[RFC1895]
	pgp-encrypted		[RFC2015]

Type	Subtype	Description	Reference
application	pgp-signature		[RFC2015]
	pgp-keys		[RFC2015]
	vnd.framemaker		[Wexler]
	vnd.mif		[Wexler]
	vnd.ms-excel		[Gill]
	vnd.ms-powerpoint		[Gill]
	vnd.ms-project		[Gill]
	vnd.ms-works		[Gill]
	vnd.ms-tnef		[Gill]
	vnd.svd		[Becker]
	vnd.music-niff		[Butler]
	vnd.ms-artgalry		[Slawson]
	vnd.truedoc		[Chase]
	vnd.koan		[Cole]
	vnd.street-stream		[Levitt]
	vnd.fdf		[Zilles]
	set-payment-initiation		[Korver]
	set-payment		[Korver]
	set-registration-initiation		[Korver]
	set-registration		[Korver]
	vnd.seemail		[Webb]
	vnd.businessobjects		[Imoucha]
	vnd.meridian-slingshot		[Wedel]
	vnd.xara		[Matthewman]
	sgml-open-catalog		[Grosso]
	vnd.rapid		[Szekely]
	vnd.enliven		[Santinelli]
	vnd.japannet-registration-wakeup		[Fujii]
	vnd.japannet-verification-wakeup		[Fujii]
	vnd.japannet-payment-wakeup		[Fujii]
	vnd.japannet-directory-service		[Fujii]
	vnd.intertrust.digibox		[Tomasello]
	vnd.intertrust.nncp		[Tomasello]
	prs.alvestrand.titrax-sheet		[Alvestrand]
	vnd.noblenet-web		[Solomon]
	vnd.noblenet-sealer		[Solomon]
	vnd.noblenet-directory		[Solomon]
	prs.nprend		[Doggett]
	vnd.webturbo		[Rehem]
	hyperstudio		[Domino]
	vnd.shana.informed.formtemplate		[Selzler]
	vnd.shana.informed.formdata		[Selzler]
	vnd.shana.informed.package		[Selzler]
	vnd.shana.informed.interchange		[Selzler]
	vnd.$commerce_battelle		[Applebaum]
	vnd.osa.netdeploy		[Klos]
	vnd.ibm.MiniPay		[Herzberg]
	vnd.japannet-jpnstore-wakeup		[Yoshitake]
	vnd.japannet-setstore-wakeup		[Yoshitake]
	vnd.japannet-verification		[Yoshitake]
	vnd.japannet-registration		[Yoshitake]
	vnd.hp-HPGL		[Pentecost]
	vnd.hp-PCL		[Pentecost]
	vnd.hp-PCLXL		[Pentecost]

Type	Subtype	Description	Reference
application	vnd.musician		[Adams]
	vnd.FloGraphIt		[Floersch]
	vnd.intercon.formnet		[Gurak]
	vemmi		[RFC2122]
	vnd.ms-asf		[Fleischman]
	vnd.ecdis-update		[Buettgenbach]
	vnd.powerbuilder6		[Guy]
	vnd.powerbuilder6-s		[Guy]
	vnd.lotus-wordpro		[Wattenberger]
	vnd.lotus-approach		[Wattenberger]
	vnd.lotus-1-2-3		[Wattenberger]
	vnd.lotus-organizer		[Wattenberger]
	vnd.lotus-screencam		[Wattenberger]
	vnd.lotus-freelance		[Wattenberger]
	vnd.fujitsu.oasys		[Togashi]
	vnd.fujitsu.oasys2		[Togashi]
	vnd.swiftview-ics		[Widener]
	vnd.dna		[Searcy]
	prs.cww		[Rungchavalnont]
	vnd.wt.stf		[Wohler]
	vnd.dxr		[Duffy]
	vnd.mitsubishi.misty-guard.trustweb		[Tanaka]
	vnd.ibm.modcap		[Hohensee]
	vnd.acucobol		[Lubin]
	vnd.fujitsu.oasys3		[Okudaira]
	marc		[RFC2220]
	vnd.fujitsu.oasysprs		[Ogita]
	vnd.fujitsu.oasysgp		[Sugimoto]
	vnd.visio		[Sandal]
	vnd.netfpx		[Mutz]
	vnd.audiograph		[Slusanschi]
	vnd.epson.salt		[Nagatomo]
	vnd.3M.Post-it-Notes		[O'Brien]
	vnd.novadigm.EDX		[Swenson]
	vnd.novadigm.EXT		[Swenson]
	vnd.novadigm.EDM		[Swenson]
	vnd.claymore		[Simpson]
	vnd.comsocaller		[Dellutri]
	pkcs7-mime		[RFC2311]
	pkcs7-signature		[RFC2311]
	pkcs10		[RFC2311]
	vnd.yellowriver-custom-menu		[Yellow]
	vnd.ecowin.chart		[Olsson]
	vnd.ecowin.series		[Olsson]
	vnd.ecowin.filerequest		[Olsson]
	vnd.ecowin.fileupdate		[Olsson]
	vnd.ecowin.seriesrequest		[Olsson]
	vnd.ecowin.seriesupdate		[Olsson]
	EDIFACT		[RFC1767]
	EDI-X12		[RFC1767]
	EDI-Consent		[RFC1767]
	vnd.wrq-hp3000-labelled		[Bartram]
	vnd.minisoft-hp3000-save		[Bartram]
	vnd.ffsns		[Holstage]
	vnd.hp-hps		[Aubrey]

Type	Subtype Description	Reference
application	vnd.fujixerox.docuworks	[Taguchi]
	xml	**[RFC3023]**
	xml-external-parsed-entity	**[RFC3023]**
	xml-dtd	**[RFC3023]**
	vnd.anser-web-funds-transfer-initiation	[Mori]
	vnd.anser-web-certificate-issue-initiation	[Mori]
	vnd.is-xpr	[Natarajan]
	vnd.intu.qbo	[Scratchley]
	vnd.publishare-delta-tree	[Ben-Kiki]
	vnd.cybank	[Helmee]
	batch-SMTP	[RFC2442]
	vnd.uplanet.alert	[Martin]
	vnd.uplanet.cacheop	[Martin]
	vnd.uplanet.list	[Martin]
	vnd.uplanet.listcmd	[Martin]
	vnd.uplanet.channel	[Martin]
	vnd.uplanet.bearer-choice	[Martin]
	vnd.uplanet.signal	[Martin]
	vnd.uplanet.alert-wbxml	[Martin]
	vnd.uplanet.cacheop-wbxml	[Martin]
	vnd.uplanet.list-wbxml	[Martin]
	vnd.uplanet.listcmd-wbxml	[Martin]
	vnd.uplanet.channel-wbxml	[Martin]
	vnd.uplanet.bearer-choice-wbxml	[Martin]
	vnd.epson.quickanime	[Gu]
	vnd.commonspace	[Chandhok]
	vnd.fut-misnet	[Pruulmann]
	vnd.xfdl	[Manning]
	vnd.intu.qfx	[Scratchley]
	vnd.epson.ssf	[Hoshina]
	vnd.epson.msf	[Hoshina]
	vnd.powerbuilder7	[Shilts]
	vnd.powerbuilder7-s	[Shilts]
	vnd.lotus-notes	[Laramie]
	pkixcmp	[RFC2510]
	vnd.wap.wmlc	[Stark]
	vnd.wap.wmlscriptc	[Stark]
	vnd.motorola.flexsuite	[Patton]
	vnd.wap.wbxml	[Stark]
	vnd.motorola.flexsuite.wem	[Patton]
	vnd.motorola.flexsuite.kmr	[Patton]
	vnd.motorola.flexsuite.adsi	[Patton]
	vnd.motorola.flexsuite.fis	[Patton]
	vnd.motorola.flexsuite.gotap	[Patton]
	vnd.motorola.flexsuite.ttc	[Patton]
	vnd.ufdl	[Manning]
	vnd.accpac.simply.imp	[Leow]
	vnd.accpac.simply.aso	[Leow]
	vnd.vcx	[T.Sugimoto]
	ipp	[RFC2910]
	ocsp-request	[RFC2560]
	ocsp-response	[RFC2560]
	vnd.previewsystems.box	[Smolgovsky]
	vnd.mediastation.cdkey	[Flurry]
	vnd.pg.format	[Gandert]

Type	Subtype	Description	Reference
application	vnd.pg.osasli		[Gandert]
	vnd.hp-hpid		[Gupta]
	pkix-cert		[RFC2585]
	pkix-crl		[RFC2585]
	vnd.Mobius.TXF		[Kabayama]
	vnd.Mobius.PLC		[Kabayama]
	vnd.Mobius.DIS		[Kabayama]
	vnd.Mobius.DAF		[Kabayama]
	vnd.Mobius.MSL		[Kabayama]
	vnd.cups-raster		[Sweet]
	vnd.cups-postscript		[Sweet]
	vnd.cups-raw		[Sweet]
	index		[RFC2652]
	index.cmd		[RFC2652]
	index.response		[RFC2652]
	index.obj		[RFC2652]
	index.vnd		[RFC2652]
	vnd.triscape.mxs		[Simonoff]
	vnd.powerbuilder75		[Shilts]
	vnd.powerbuilder75-s		[Shilts]
	vnd.dpgraph		[Parker]
	http		[RFC2616]
	sdp		[RFC2327]
	vnd.eudora.data		[Resnick]
	vnd.fujixerox.docuworks.binder		[Matsumoto]
	vnd.vectorworks		[Pharr]
	vnd.grafeq		[Tupper]
	vnd.bmi		[Gotoh]
	vnd.ericsson.quickcall		[Tidwell]
	vnd.hzn-3d-crossword		[Minnis]
	vnd.wap.slc		[WAP-Forum]
	vnd.wap.sic		[WAP-Forum]
	vnd.groove-injector		[Joseph]
	vnd.fujixerox.ddd		[Onda]
	vnd.groove-account		[Joseph]
	vnd.groove-identity-message		[Joseph]
	vnd.groove-tool-message		[Joseph]
	vnd.groove-tool-template		[Joseph]
	vnd.groove-vcard		[Joseph]
	vnd.ctc-posml		[Kohlhepp]
	vnd.canon-lips		[Muto]
	vnd.canon-cpdl		[Muto]
	vnd.trueapp		[Hepler]
	vnd.s3sms		[Tarkkala]
	iotp		[RFC2935]
	vnd.mcd		[Gotoh]
	vnd.httphone		[Lefevre]
	vnd.informix-visionary		[Gales]
	vnd.msign		[Borcherding]
	vnd.ms-lrm		[Ledoux]
	vnd.contact.cmsg		[Patz]
	vnd.epson.esf		[Hoshina]
	whoispp-query		[RFC2957]
	whoispp-response		[RFC2958]
	vnd.mozilla.xul+xml		[McDaniel]

Type	Subtype	Description	Reference
application	parityfec		[RFC3009]
	vnd.palm		[Peacock]
	vnd.fsc.weblaunch		[D.Smith]
	vnd.tve-trigger		[Welsh]
	dvcs		[RFC3029]
	sieve		[RFC3028]
	vnd.vividence.scriptfile		[Risher]
	vnd.hhe.lesson-player		[Jones]
	beep+xml		[RFC3080]
	font-tdpfr		[RFC3073]
	vnd.mseq		[Le Bodic]
	vnd.aether.imp		[Moskowitz]
	vnd.Mobius.MQY		[Devasia]
	vnd.Mobius.MBK		[Devasia]
	vnd.vidsoft.vidconference		[Hess]
	vnd.ibm.afplinedata		[Buis]
	vnd.irepository.package+xml		**[Knowles]**
	vnd.sss-ntf		**[Bruno]**
	vnd.sss-dtf		**[Bruno]**
	vnd.sss-cod		**[Dani]**
	vnd.pvi.ptid1		**[Lamb]**
	isup		**[RFCISUP]**
	qsig		**[RFCISUP]**
	timestamp-query		**[RFC3161]**
	timestamp-reply		**[RFC3161]**
	vnd.pwg-xhtml-print+xml		**[Wright]**
image	jpeg		[RFC2045,RFC2046]
	gif		[RFC2045,RFC2046]
	ief	Image Exchange Format	[RFC1314]
	g3fax		[RFC1494]
	tiff	Tag Image File Format	[RFC2302]
	cgm	Computer Graphics Metafile	[Francis]
	naplps		[Ferber]
	vnd.dwg		[Moline]
	vnd.svf		[Moline]
	vnd.dxf		[Moline]
	png		[Randers-Pehrson]
	vnd.fpx		[Spencer]
	vnd.net-fpx		[Spencer]
	vnd.xiff		[SMartin]
	prs.btif		[Simon]
	vnd.fastbidsheet		[Becker]
	vnd.wap.wbmp		[Stark]
	prs.pti		[Laun]
	vnd.cns.inf2		[McLaughlin]
	vnd.mix		[Reddy]
	vnd.fujixerox.edmics-rlc		[Onda]
	vnd.fujixerox.edmics-mmr		[Onda]
	vnd.fst		[Fuldseth]

Type	Subtype	Description	Reference
audio	basic		[RFC2045,RFC2046]
	32kadpcm		[RFC2421,RFC2422]
	vnd.qcelp		[Lundblade]
	vnd.digital-winds		[Strazds]
	vnd.lucent.voice		[Vaudreuil]
	vnd.octel.sbc		[Vaudreuil]
	vnd.rhetorex.32kadpcm		[Vaudreuil]
	vnd.vmx.cvsd		[Vaudreuil]
	vnd.nortel.vbk		[Parsons]
	vnd.cns.anp1		[McLaughlin]
	vnd.cns.inf1		[McLaughlin]
	L16		[RFC2586]
	vnd.everad.plj		[Cicelsky]
	telephone-event		[RFC2833]
	tone		[RFC2833]
	prs.sid		[Walleij]
	vnd.nuera.ecelp4800		[Fox]
	vnd.nuera.ecelp7470		[Fox]
	mpeg		[RFC3003]
	parityfec		[RFC3009]
	MP4A-LATM		[RFC3016]
	vnd.nuera.ecelp9600		[Fox]
	G.722.1		[RFC3047]
	mpa-robust		**[RFC3119]**
	vnd.cisco.nse		**[Kumar]**
	DAT12		**[RFCAVTD]**
	L20		**[RFCAVTD]**
	L24		**[RFCAVTD]**
video	mpeg		[RFC2045,RFC2046]
	quicktime		[Paul Lindner]
	vnd.vivo		[Wolfe]
	vnd.motorola.video		[McGinty]
	vnd.motorola.videop		[McGinty]
	vnd.fvt		[Fuldseth]
	pointer		[RFC2862]
	parityfec		[RFC3009]
	vnd.mpegurl		[Recktenwald]
	MP4V-ES		[RFC3016]
	vnd.nokia.interleaved-multimedia		[Kangaslampi]

Type	Subtype	Description	Reference
model			[RFC2077]
	iges		[Parks]
	vrml		[RFC2077]
	mesh		[RFC2077]
	vnd.dwf		[Pratt]
	vnd.gtw		[Ozaki]
	vnd.flatland.3dml		[Powers]
	vnd.vtu		[Rabinovitch]
	vnd.mts		[Rabinovitch]
	vnd.gdl		[Babits]
	vnd.gs-gdl		[Babits]
	vnd.parasolid.transmit.text		[Dearnaley,Juckes]
	vnd.parasolid.transmit.binary		[Dearnaley,Juckes]

The "media-types" directory contains a subdirectory for each content type and each of those directories contains a file for each content subtype.

```
                             |-application-
                             |-audio-------
                             |-image-------
              |-media-types-|-message-----
                             |-model-------
                             |-multipart---
                             |-text--------
                             |-video-------
```

```
   URL = ftp://ftp.isi.edu/in-notes/iana/assignments/media-types
```

Character Sets

All of the character sets listed the section on Character Sets are registered for use with MIME as MIME Character Sets. The correspondance between the few character sets listed in the MIME specifications [RFC2045,RFC2046] and the list in that section are:

Type	Description	Reference
US-ASCII	see ANSI_X3.4-1968 below	[RFC2045,RFC2046]
ISO-8859-1	see ISO_8859-1:1987 below	[RFC2045,RFC2046]
ISO-8859-2	see ISO_8859-2:1987 below	[RFC2045,RFC2046]
ISO-8859-3	see ISO_8859-3:1988 below	[RFC2045,RFC2046]
ISO-8859-4	see ISO_8859-4:1988 below	[RFC2045,RFC2046]
ISO-8859-5	see ISO_8859-5:1988 below	[RFC2045,RFC2046]
ISO-8859-6	see ISO_8859-6:1987 below	[RFC2045,RFC2046]
ISO-8859-7	see ISO_8859-7:1987 below	[RFC2045,RFC2046]
ISO-8859-8	see ISO_8859-8:1988 below	[RFC2045,RFC2046]
ISO-8859-9	see ISO_8859-9:1989 below	[RFC2045,RFC2046]

Access Types

Type	Description	Reference
FTP		[RFC2045,RFC2046]
ANON-FTP		[RFC2045,RFC2046]
TFTP		[RFC2045,RFC2046]
AFS		[RFC2045,RFC2046]
LOCAL-FILE		[RFC2045,RFC2046]
MAIL-SERVER		[RFC2045,RFC2046]
content-id		[RFC1873]

Conversion Values

Conversion values or Content Transfer Encodings.

Type	Description	Reference
7BIT		[RFC2045,RFC2046]
8BIT		[RFC2045,RFC2046]
BASE64		[RFC2045,RFC2046]
BINARY		[RFC2045,RFC2046]
QUOTED-PRINTABLE		[RFC2045,RFC2046]

MIME / X.400 MAPPING TABLES

MIME to X.400 Table

MIME content-type	X.400 Body Part	Reference
text/plain		
charset=us-ascii	ia5-text	[RFC1494]
charset=iso-8859-x	EBP - GeneralText	[RFC1494]
text/richtext	no mapping defined	[RFC1494]
application/oda	EBP - ODA	[RFC1494]
application/octet-stream	bilaterally-defined	[RFC1494]
application/postscript	EBP - mime-postscript-body	[RFC1494]
image/g3fax	g3-facsimile	[RFC1494]
image/jpeg	EBP - mime-jpeg-body	[RFC1494]
image/gif	EBP - mime-gif-body	[RFC1494]
audio/basic	no mapping defined	[RFC1494]
video/mpeg	no mapping defined	[RFC1494]

Abbreviation: EBP - Extended Body Part

X.400 to MIME Table
Basic Body Parts

X.400 Basic Body Part	MIME content-type	Reference
ia5-text	text/plain;charset=us-ascii	[RFC1494]
voice	No Mapping Defined	[RFC1494]
g3-facsimile	image/g3fax	[RFC1494]
g4-class1	no mapping defined	[RFC1494]
teletex	no mapping defined	[RFC1494]
videotex	no mapping defined	[RFC1494]
encrypted	no mapping defined	[RFC1494]
bilaterally-defined	application/octet-stream	[RFC1494]
nationally-defined	no mapping defined	[RFC1494]
externally-defined	See Extended Body Parts	[RFC1494]

X.400 Extended Body Part	MIME content-type	Reference
GeneralText	text/plain;charset=iso-8859-x	[RFC1494]
ODA	application/oda	[RFC1494]
mime-postscript-body	application/postscript	[RFC1494]
mime-jpeg-body	image/jpeg	[RFC1494]
mime-gif-body	image/gif	[RFC1494]

REFERENCES

[MacMime] Work in Progress.

[RFC1036] Horton, M., and R. Adams, "Standard for Interchange of USENET Messages", RFC 1036, AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[RFC1494] Alvestrand, H., and S. Thompson, "Equivalences between 1988 X.400 and RFC-822 Message Bodies", RFC 1494, SINTEF DELAB, Soft*Switch, Inc., August 1993.

[RFC1563] Borenstien, N., "The text/enriched MIME content-type". RFC 1563, Bellcore, January 1994.

[RFC1767] Crocker, D., "MIME Encapsulation of EDI Objects". RFC 1767, Brandenburg Consulting, March 1995.

[RFC1866] Berners-Lee, T., and D. Connolly, "Hypertext Markup Language- 2.0", RFC 1866, MIT/W3C, November 1995.

[RFC1873] Levinson, E., "Message/External-Body Content-ID Access Type", RFC 1873, Accurate Information Systems, Inc. December 1995.

[RFC1874] Levinson, E., "SGML Media Types", RFC 1874, Accurate Information Systems, Inc. December 1995.

[RFC1892] Vaudreuil, G., "The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages", RFC 1892, Octel Network Services, January 1996.

[RFC1894] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 1894, University of Tennessee, Octel Network Services, January 1996.

[RFC1895] Levinson, E., "The Application/CALS-1840 Content Type", RFC 1895, Accurate Information Systems, February 1996.

[RFC1896] Resnick, P., and A. Walker, "The Text/Enriched MIME Content Type", RFC 1896, Qualcomm, Intercon, February 1996.

[RFC1945] Berners-Lee, Y., R. Feilding, and H.Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945. MIT/LCS, UC Irvine, MIT/LCS, May 1996.

[RFC2045] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2046] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2077] Nelson, S., C. Parks, and Mitra, "The Model Primary Content Type for Multipurpose Internet Mail Extensions", RFC 2077, LLNL, NIST, WorldMaker, January 1997.

[RFC2122] Mavrakis, D., Layec, H., and K. Kartmann, "VEMMI URL Specification", RFC 2122, Monaco Telematique MC-TEL, ETSI, Telecommunication+Multimedia, March 1997.

[RFC2220] Guenther, R., "The Application/MARC Content-type", RFC 2220, Library of Congress, Network Devt. & MARC Standards Office, October 1997.

[RFC2298] Fajman, R., "An Extensible Message Format for Message Disposition Notifications", RFC 2298, March 1998.

[RFC2302] Parsons, G., et. al., "Tag Image File Format (TIFF) - image/tiff", RFC 2302, March 1998.

[RFC2311] Dusse, S., et. al., "S/MIME Version 2 Message Specification, RFC 2311, March 1998.

[RFC2318] Lie, H., Bos, B., and C. Lilley, "The text/css Media Type", RFC 2318, March 1998.

[RFC2327] Handley, M., and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1999.

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, XIson Inc, August 1998.

[RFC2388] Masinter, L., "Form-based File Upload in HTML", RFC 2388, Xerox Corporation, August 1998.

[RFC2421] Vaudreuil, G., and G. Parsons, "Voice Profile for Internet Mail - version 2", RFC 2421, September 1998.

[RFC2422] Vaudreuil, G., and G. Parsons, "Toll Quality Voice - 32 kbit/s ADPCM MIME Sub-type Registration", RFC 2422, September 1998.

[RFC2423] Vaudreuil, G., and G. Parsons, "VPIM Voice Message MIME Sub-type Registration", RFC 2423, September 1998.

[RFC2425] Howes, T., Smith, M., and F. Dawson, "A MIME Content-Type for Directory Information", RFC 2425, September 1998.

[RFC2442] Freed, N., Newman, D., Belissent, J. and M. Hoy, "The Batch SMTP Media Type", RFC 2442, November 1998.

[RFC2445] Dawson, F., and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998.

[RFC2483] M. Mealling and R. Daniel, "URI resolution services necessary for URN resolution", RFC 2483, January 1999.

[RFC2510] Adams, C., and S. Farrell, "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999.

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, May 1999.

[RFC2586] Salsman, J and H. Alvestrand, "The Audio/L16 MIME content type", RFC 2586, May 1999.

[RFC2616] Fielding, R., et. al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2652] Allen, J., and M. Mealling, "MIME Object Definitions for the Common Indexing Protocol (CIP)", RFC 2652, August 1999.

[RFC2793] Hellstrom, G., "RTP Payload for Text Conversation", RFC 2793, May 2000.

[RFC2833] Schulzrinne, H., "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[RFC2854] Connolly, D., and L. Masinter, "The 'text/html' Media Type", RFC 2854, June 2000.

[RFC2862] Civanlar, M., and G. Cash, "RTP Payload Format for Real-Time Pointers", RFC 2862, June 2000.

[RFC2910] Herriot, R., Editor, Butler, S., Moore, P., Turner, R. and J. Wenn, "Internet Printing Protocol/1.0: Encoding and Transport", RFC 2910, September 2000.

[RFC2935] Eastlake, D. and C. Smith, "Internet Open Trading Protocol (IOTP) HTTP Supplement", RFC 2935, September 2000.

[RFC3003] M. Nilsson, "The audio/mpeg Media Type", RFC 3003, November 2000.

[RFC3009] J.Rosenberg and H.Schulzrinne, "Registration of parityfec MIME types", RFC 3009, November 2000.

[RFC3016] Kikuchi, Y., T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP payload format for MPEG-4 Audio/Visual streams", RFC 3016, November 2000.

[RFC3023] M. Murata, S. St.Laurent, and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3028] T. Showalter, "Sieve: A Mail Filtering Language" RFC 3028, January 2001.

[RFC3029] Adams,C. , P. Sylvester, M. Zolotarev, and R. Zuccherato, "Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols", RFC 3029, January 2001.

[RFC3047] Luthi, P. "RTP Payload Format for ITU-T Recommendation G.772.1", RFC 3047, January 2001.

[RFC3073] Collins, J., "Portable Font Resource (PFR) - application/font-tdpfr MIME Sub-type Registration", RFC 3073, February 2001.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, February 2001.

[RFC3119] R. Finlayson, "A More Loss-Tolerant RTP Payload Format for MP3 Audio", RFC 3119, June 2001.

[RFCISUP] E. Zimmerer, J. Peterson, A. Vemuri, L. Ong, F. Audet, M. Watson, and M. Zonoun, "MIME media types for ISUP and QSIG Objects", RFC XXXX, Month Year.

[RFC3156] M. Elkins, D. Del Torto, R. Levien, and T. Roessler, "MIME Security with OpenPGP", RFC 3156, August 2001.

[RFC3161] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)", RFC 3161, August 2001.

[RFCAVTD] K. Kobayashi, A. Ogawa, S. Casner, and C. Bormann,"RTP Payload Format for DV Format Video", RFC XXXX, Month Year.

PEOPLE

[Adams] Greg Adams <gadams@waynesworld.ucsd.edu>, March 1997.
[Allen] Steve Allen <sla@ucolick.org>, September 1997.
[Alvestrand] Harald T. Alvestrand <Harald.T.Alvestrand@uninett.no>, Jan 1997.
[Applebaum] David Applebaum, <applebau@battelle.org>, February 1997.
[Aubrey] Steve Aubrey, <steve_aubrey@hp.com>, July 1998.
[Babits] Attila Babits, <ababits@graphisoft.hu>, April 2000, May 2000.
[Bartram] Chris Bartram, <RCB@3k.com>, May 1998.
[Becker] Scott Becker, <scottb@bxwa.com>, April 1996, October 1998.
[Ben-Kiki] Oren Ben-Kiki, <oren@capella.co.il>, October 1998.
[Berners-Lee] Tim Berners-Lee, <timbl@w3.org>, May 1996.
[Borcherding] Malte Borcherding, <Malte.Borcherding@brokat.com>, Aug 2000.
[Borenstein] Nathaniel Borenstein, <NSB@bellcore.com>, April 1994.
[Bradley] Dan Bradley, <dan@dantom.com>, October 2000.
[Bruno] Eric Bruno, <ebruno@solution-soft.com>, June 2001.
[Buettgenbach] Gert Buettgenbach, <bue@sevencs.com>, May 1997.
[Buis] Roger Buis, <buis@us.ibm.com>, March 2001.
[Butler] Tim Butler, <tim@its.bldrdoc.gov>, April 1996.
[Larry Campbell]
[Chandhok] Ravinder Chandhok, <chandhok@within.com>, December 1998.
[Chase] Brad Chase, <brad_chase@bitstream.com>, May 1996.
[Cicelsky] Shay Cicelsky, <shayc@everad.com>, May 2000.
[Cole] Pete Cole, <pcole@sseyod.demon.co.uk>, June 1996.
[Dave Crocker] Dave Crocker <dcrocker@mordor.stanford.edu>
[Terry Crowley]
[Dani] Asang Dani, <adani@solution-soft.com>, June 2001.
[Daniel] Ron Daniel, Jr. <rdaniel@lanl.gov>, June 1997.
[Dearnaley] Roger Dearnaley, <x_dearna@ugsolutions.com>, October 2000.
[Dellutri] Steve Dellutri, <sdellutri@cosmocom.com>, March 1998.
[Devasia] Alex Devasia, <adevasia@mobius.com>, March 2001.
[Doggett] Jay Doggett, <jdoggett@tiac.net>, February 1997.
[Domino] Michael Domino, <michaeldomino@mediaone.net>, February 1997.
[Duffy] Michael Duffy, <miked@psiaustin.com>, September 1997.
[Eastlake] Donald E. Eastlake 3rd, <Donald.Eastlake@motorola.com>, April 1995, May 2000.
[Faltstrom] Patrik Faltstrom <paf@nada.kth.se>
[Fleischman] Eric Fleischman <ericfl@MICROSOFT.com>, April 1997.
[Floersch] Dick Floersch <floersch@echo.sound.net>, March 1997.
[Flurry] Henry Flurry <henryf@mediastation.com>, April 1999.
[Fox] Michael Fox, <mfox@nuera.com>, August 2000, January 2001.
[Francis] Alan Francis, A.H.Francis@open.ac.uk, December 1995.
[Fujii] Kiyofusa Fujii <kfujii@japannet.or.jp>, February 1997.
[Fuldseth] Arild Fuldseth, <Arild.Fuldseth@fast.no>, June 2000.
[Gales] Christopher Gales, <christopher.gales@informix.com>, August 2000.
[Gandert] April Gandert <gandert.am@pg.com>, April 1999.
[Gill] Sukvinder S. Gill, <sukvg@microsoft.com>, April 1996.
[Glazer] David Glazer, <dglazer@best.com>, April 1995.
[Gotoh] Tadashi Gotoh, <tgotoh@cadamsystems.co.jp>, Feb 2000,June 2000.
[Gu] Yu Gu, <guyu@rd.oda.epson.co.jp>, December 1998.
[Gupta] Aloke Gupta <Aloke_Gupta@ex.cv.hp.com>, April 1999.
[Gurak] Tom Gurak, <assoc@intercon.roc.servtech.com>, March 1997.
[Gurney] John-Mark Gurney <jmg@flyidea.com>, August 1999.
[Guy] David Guy, <dguy@powersoft.com>, June 1997.
[Helmee] Nor Helmee, <helmee@my.cybank.net>, November 1998.
[Hepler] J. Scott Hepler, <scott@truebasic.com>, May 2000.

[Herzberg] Amir Herzberg, <amirh@haifa.vnet.ibm.com>, February 1997.
[Hess] Robert Hess, <hess@vidsoft.de>, March 2001.
[Hodge] Tim Hodge, <thodge@curl.com>, August 2000.
[Hohensee] Reinhard Hohensee <rhohensee@VNET.IBM.COM>, September 1997.
[Holstage] Mary Holstage <holstege@firstfloor.com>, May 1998.
[Hoshina] Shoji Hoshina <Hoshina.Shoji@exc.epson.co.jp>, Jan 1999, Sept 2000.
[Hurtta] Kari E. Hurtta <flexstor@ozone.FMI.FI>
[Imoucha] Philippe Imoucha <pimoucha@businessobjects.com>, October 1996.
[IPTC] International Press Telecommunications Council (David Allen), <m_director_iptc@dial.pipex.com>, July 2000.
[Jones] Randy Jones, <randy_jones@archipelago.com>, January 2001.
[Joseph] Todd Joseph <todd_joseph@groove.net>, Feb 2000, Mar 2000, Apr 2000.
[Juckes] John Juckes, <johnj@ugsolutions.com>, October 2000.
[Kangaslampi] Petteri Kangaslampi, <petteri.kangaslampi@nokia.com>, Mar 2001.
[Katz] Steve Katz, <skatz@eshop.com>, June 1995.
[Klos] Steven Klos, <stevek@osa.com>, February 1997.
[Knowles] Martin Knowles, <mjk@irepository.net>, June 2001.
[Kohlhepp] Bayard Kohlhepp, <bayard@ctcexchange.com>, April 2000.
[Korver] Brian Korver <briank@terisa.com>, October 1996.
[Kumar] Rajesh Kumar, <rkumar@cisco.com>, August 2001.
[Lamb] Charles P. Lamb, <CLamb@pvimage.com>, June 2001.
[Laramie] Michael Laramie <laramiem@btv.ibm.com>, February 1999.
[Laun] Juern Laun <juern.laun@gmx.de>, April 1999.
[Le Bodic] Gwenael Le Bodic <Gwenael.Le_Bodic@alcatel.fr>, March 2001.
[Ledoux] Eric Ledoux, <ericle@microsoft.com>, August 2000.
[Lefevre] Franck Lefevre, <franck@k1info.com>, August 2000.
[Leow] Steve Leow <Leost01@accpac.com>, April 1999.
[Levitt] Glenn Levitt <streetd1@ix.netcom.com>, October 1996.
[Lines] John Lines <john@paladin.demon.co.uk>, January 1998.
[Lubin] Dovid Lubin <dovid@acucobol.com>, October 1997.
[Lubos] Mikusiak Lubos <lmikusia@blava-s.bratisla.ingr.com>, October 1996.
[Lundblade] Laurence Lundblade <lgl@qualcomm.com>, October 1996.
[Manning] Dave Manning <dmanning@uwi.com>, January, March 1999.
[Martin] Bruce Martin <iana-registrar@uplanet.com>, November 1998.
[Martin] Steven Martin <smartin@xis.xerox.com>, October 1997.
[Matsumoto] Takashi Matsumoto <takashi.matsumoto@fujixerox.co.jp>,
            February 2000
[Matthewman] David Matthewman <david@xara.com>, October 1996.
[McDaniel] Braden N. McDaniel, <braden@endoframe.com>, October 2000.
[McGinty] Tom McGinty <tmcginty@dma.isg.mot.com>
[McLaughlin] Ann McLaughlin <amclaughlin@comversens.com>, April 1999.
[Minnis] James Minnis <james-minnis@glimpse-of-tomorrow.com>, Feb 2000
[Moline] Jodi Moline, <jodim@softsource.com>, April 1996.
[Mori] Hiroyoshi Mori, <mori@mm.rd.nttdata.co.jp>, August 1998.
[Moskowitz] Jay Moskowitz, <jay@aethersystems.com>, March 2001.
[Muto] Shin Muto, <shinmuto@pure.cpdc.canon.co.jp>, May 2000.
[Mutz] Andy Mutz, <andy_mutz@hp.com>, December 1997.
[Nagatomo] Yasuhito Nagatomo <naga@rd.oda.epson.co.jp>, January 1998.
[Natarajan] Satish Natarajan, <satish@infoseek.com>, August 1998.
[Nelson] Jan Nelson, <jann@microsoft.com>, May 2000.
[Nilsson] Martin Nilsson, <nilsson@id3.org>, October 2000.
[O'Brien] Michael O'Brien <meobrien1@mmm.com>, January 1998.
[Ogita] Masumi Ogita, <ogita@oa.tfl.fujitsu.co.jp>, October 1997.
[Okudaira] Seiji Okudaira <okudaira@candy.paso.fujitsu.co.jp>, Oct 1997.
[Olsson] Thomas Olsson <thomas@vinga.se>, April 1998.
[Onda] Masanori Onda <Masanori.Onda@fujixerox.co.jp>, February 2000.
[Ozaki] Yutaka Ozaki <yutaka_ozaki@gen.co.jp>, January 1999.

```
[Paul Lindner]
[Parker] David Parker <davidparker@davidparker.com>, August 1999.
[Parks] Curtis Parks, <parks@eeel.nist.gov>, April 1995.
[Parsons] Glenn Parsons <gparsons@nortelnetworks.com>, February 1999.
[Patton] Mark Patton <fmp014@email.mot.com>, March 1999.
[Patz] Frank Patz, <fp@contact.de>, September 2000.
[Peacock] Gavin Peacock, <gpeacock@palm.com>, November 2000.
[Pentecost] Bob Pentecost, <bpenteco@boi.hp.com>, March 1997.
[Pharr] Paul C. Pharr <pharr@diehlgraphsoft.com>, February 2000.
[Powers] Michael Powers, <powers@insideout.net>, January 1998.
      <pow@flatland.com>, January 1999.
[Pratt] Jason Pratt, <jason.pratt@autodesk.com>, August 1997.
[Pruulmann] Jann Pruulmann, <jaan@fut.ee>, December 1998.
[Rabinovitch] Boris Rabinovitch <boris@virtue3d.com>, February 2000.
[Randers-Pehrson] Glenn Randers-Pehrson <glennrp@ARL.MIL>, October 1996.
[Recktenwald] Heiko Recktenwald, <uzs106@uni-bonn.de>, November 2000.
[Reddy] Saveen Reddy <saveenr@miscrosoft.com>, July 1999.
[Rehem] Yaser Rehem, <yrehem@sapient.com>, February 1997.
[Resnick] Pete Resnick, <presnick@qualcomm.com>, February 2000.
[Risher] Mark Risher, <markr@vividence.com>, December 2000.
[Rose] Marshall Rose, <mrose@dbc.mtview.ca.us>, April 1995.
[Rosenberg] Jonathan Rosenberg, <jdrosen@dynamicsoft.com>, October 2000.
[Rungchavalnont] Khemchart Rungchavalnont,
<khemcr@cpu.cp.eng.chula.ac.th>, July 1997.
[Sandal] Troy Sandal <troys@visio.com>, November 1997.
[Santinelli] Paul Santinelli, Jr. <psantinelli@narrative.com>, Oct 1996.
[Scrathcley] Greg Scratchley <greg_scratchley@intuit.com>, October 1998.
[Searcy] Meredith Searcy, <msearcy@newmoon.com>, June 1997.
[Shapiro] Ehud Shapiro
[Shilts] Reed Shilts <reed.shilts@sybase.com>, February 1999, August 1999.
[Simon] Ben Simon, <BenS@crt.com>, September 1998.
[Simonoff] Steven Simonoff <scs@triscape.com>, August 1999.
[Simpson] Ray Simpson <ray@cnation.com>, January 1998.
[Slawson] Dean Slawson, <deansl@microsoft.com>, May 1996.
[Slusanschi] Horia Cristian Slusanschi <H.C.Slusanschi@massey.ac.nz>,
             January 1998.
[D.Smith] Derek Smith, <derek@friendlysoftware.com>, November 2000.
[Smith] Nick Smith, <nas@ant.co.uk>, June 1995.
[Smolgovsky] Roman Smolgovsky <romans@previewsystems.com>, April 1999.
[Solomon] Monty Solomon, <monty@noblenet.com>, February 1997.
[Spencer] Marc Douglas Spencer <marcs@itc.kodak.com>, October 1996.
[Henry Spencer]
[Stark] Peter Stark <stark@uplanet.com>, March 1999.
[Strazds] Armands Strazds <armands.strazds@medienhaus-bremen.de>, Jan 1999.
[Sugimoto] Masahiko Sugimoto <sugimoto@sz.sel.fujitsu.co.jp>, Oct 1997.
[T.Sugimoto] Taisuke Sugimoto <sugimototi@noanet.nttdata.co.jp> Apr 1999.
[Sweet] Michael Sweet <mike@easysw.com>, July 1999.
[Swenson] Janine Swenson <janine@novadigm.com>, January 1998.
[Szekely] Etay Szekely <etay@emultek.co.il>, October 1996.
[Taguchi] Yasuo Taguchi <yasuo.taguchi@fujixerox.co.jp>, July 1998.
[Tanaka] Manabu Tanaka <mtana@iss.isl.melco.co.jp>, September 1997.
[Tarkkala] Lauri Tarkkala, <Lauri.Tarkkala@sonera.com>, May 2000.
[Tidwell] Paul Tidwell <paul.tidwell@ericsson.com>, February 2000.
[Togashi] Nobukazu Togashi <togashi@ai.cs.fujitsu.co.jp>, June 1997.
[Tomasello] Luke Tomasello <luket@intertrust.com>
[Tupper] Jeff Tupper <tupper@peda.com>, February 2000.
[Vaudreuil] Greg Vaudreuil <gregv@lucent.com>, January 1999.
```

[Walleij] Linus Walleij, <triad@df.lth.se>, July 2000.
[WAP-Forum] WAP Forum Ltd. <wap-feedback@mail.wapforum.org>, Feb 2000.
[Wattenberger] Paul Wattenberger <Paul_Wattenberger@lotus.com>, June 1997.
[Webb] Steve Webb <steve@wynde.com>, October 1996.
[Wedel] Eric Wedel <ewedel@meridian-data.com>, October 1996.
[Welsh] Linda Welsh, <linda@intel.com>, November 2000.
[Wexler] Mike Wexler, <mwexler@frame.com>, April 1996.
[Widener] Glenn Widener <glennw@ndg.com>, June 1997.
[Wohler] Bill Wohler, <wohler@newt.com>, July 1997.
[Wolfe] John Wolfe, <John_Wolfe.VIVO@vivo.com>, April 1996.
[Wright] Don Wright, <don@lexmark.com>, August 2001.
[Van Nostern] Gene C. Van Nostern <gene@wri.com>, February 1995.
[Yellow] Mr. Yellow <yellowriversw@yahoo.com>, March 1998.
[Yoshitake] Jun Yoshitake, <yositake@iss.isl.melco.co.jp>, February 1997.
[Zilles] Steve Zilles <szilles@adobe.com>, October 1996.