X.media.publishing

Dick C. A. Bulterman
Lloyd Rutledge

X.media.publishing is an application-oriented series from Springer-Verlag that specializes in the presentation and publication of multimedia as well as digital and print media.

SMIL 2.0

Interactive Multimedia for Web and Mobile Devices

Springer

Amazon v. Audio Pod
US Patent 8,738,740
Amazon EX-1007

X.media.publishing



Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Dick C.A. Bulterman Lloyd Rutledge

SMIL 2.0

Interactive Multimedia for Web and Mobile Devices

With 105 Figures and 81 Tables



Springer

Dick C. A. Bulterman
Lloyd Rutledge

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Dick.Bulterman@cwi.nl
Lloyd.Rutledge@cwi.nl

http://www.XmediaSMIL.net/

ISSN 1612-1449
ISBN 3-540-20234-X Springer-Verlag Berlin Heidelberg New York

Library of Congress Cataloging-in-Publication-Data
Bulterman, Dick C.A., 1951–
SMIL 2.0: interactive multimedia for Web and mobile devices/Dick C.A. Bulterman, Lloyd Rutledge.
p. cm. – (X.media.publishing)
ISBN 3-540-20234-X (alk. paper)
1. SMIL (Document markup language) I. Rutledge, Lloyd, 1966– II. Title. III. Series.
QA76.76.H94B858 2004
006.7'4–dc22 2004041320

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German copyright law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag is a part of Springer Science+Business Media
springeronline.com

© Springer-Verlag Berlin Heidelberg 2004
Printed in Germany

The use of general descriptive names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: KünkelLopka, Heidelberg
Typesetting and data preparation by the authors
Printed on acid-free paper 33/3142 GF – 543210

Dedications

Dick C.A. Bulterman:

There are many people in my private life — old and young, near and far, and alive and dead — to whom I wish to dedicate this book, but singling any of them out would do a disservice to the rest. The choice in my professional life is much simpler: one person stands out as an influence that has changed the way I think and work. Andy van Dam is a gifted and insightful researcher and teacher. He has served as an inspiration (and occasionally a warning!) during the 30 years since I first showed up at Brown, and I am pleased to dedicate this book to him.

Lloyd Rutledge:

Writing a book is but one of life's milestones, splitting a cherished past from a future looked forward to. With this in mind, I have two dedications for this work. To the memory of my mother Helen, who passed on when work on this book began. And to Marie-Louise, with whom I married during this work's final stages.

Preface

SMIL 2.0: Multimedia for Web and Mobile Devices is a comprehensive introduction to — and resource guide for — the W3C SMIL 2.0 language. It covers all aspects of the SMIL specification and covers all of SMIL's implementation profiles, from the desktop through the new world of mobile SMIL devices.

Motivation for this Book

When the SMIL 2.0 recommendation was being developed, it was clear to us that the richness of the SMIL language could easily overwhelm many Web authors and designers. In the 500+ pages that the SYMM working group needed to describe the 32 SMIL elements and the 150 SMIL attributes, there was not much room for background information or extensive examples. The focus of the specification was on the components of the SMIL language, not on the rationale or the potential uses of SMIL's declarative power.

The existing literature on SMIL is not extensive. Other books published on aspects of the SMIL language are neither complete nor always correct in their handling of SMIL concepts or the various SMIL implementations. The documentation provided by major vendors of SMIL players is also not geared to the development of a complete picture of what SMIL can do — and what it can't do! The academic literature, while often containing thoughtful analysis of SMIL, is inaccessible to most of SMIL's users.

We wrote this book to bridge the gap between the theory and practice of creating SMIL presentations for multiple platforms and multiple profiles. In so doing, we hope that the use of SMIL's facilities and features moves beyond the trivial and enters the mainstream of Web multimedia.

Should You Read This Book?

We wrote this book for three communities:

- *Existing* SMIL *authors*: many existing SMIL users need and want more information on the language so that they can create better presentations, for more profiles, with less effort.
- *New* SMIL *authors*: SMIL is currently available on nearly a million desktop and mobile devices. It represents a media delivery platform that has unprecedented reach. Designers and authors who want to exploit the

potential of SMIL need to have a solid introduction to the language and to have a wealth of examples that can help them get started.

- *Developers and students of multimedia technology*: The SMIL standard is a complex document. For researchers, developers and students of internet media technology, it is easy to lose sight of the big picture when architecting delivery systems for SMIL technology. We have done our best to de-mystify SMIL and to make the basic concepts addressed by SMIL clear and accessible.

This book is intended to augment the documentation provided by individual SMIL player vendors. We have decided to focus our efforts on explaining SMIL technology; together with authoring guidelines and documentation provided and maintained by the W3C, RealNetworks, Microsoft, Oratrix, Apple and other player vendors, we are confident that you will be able to create, maintain, and debug a complete range of SMIL presentations, for a complete range of SMIL players.

Now that a number of open-source SMIL players are available, we also expect that this book — together with the SMIL 2.0 specification provided by W3C — will provide a comprehensive guide for designers and implementers of new or modified SMIL players.

Structure of the Book

This book is divided into four sections. *Part One* provides a general introduction to the SMIL language and a reader's guide to a (complex) SMIL example. It also contains background information on Web multimedia technology that, while not limited to SMIL, is useful for users who are new to networked multimedia. *Part Two* provides an overview of the basic elements and attributes used by all implementations of SMIL. It covers each of the functional groups of SMIL technology, from structure to transitions. *Part Three* contains a discussion of advanced SMIL features. These are elements and attributes that are either profile or player specific, or that make use of esoteric SMIL features. *Part Four* contains reference information on the composition of five of SMIL's profile implementations.

Each chapter is structured to meet the needs of the three target readership groups defined above. An XML-like structure of each chapter is:

```
<chapter>
  <BackgroundInformation />
  <ElementsAndAttributes>
    <ElementDefinitions />
    <AttributeDefintions />
    <SpecialValuesDefinitions />
  </ElementsAndAttributes>
  <Examples/>
</chapter>
```

We expect that many current SMIL authors will want to jump directly to the element and attribute definitions in each chapter; these provide detailed information that can answer questions quickly and (relatively!) painlessly about the

structure of the language and the permissible attribute values. For new SMIL users, the example sub-section provides a host of easy-to-digest uses of the technology; often starting here will give a good idea of what SMIL can do in the functional group covered by that chapter. For readers interested in not only the *how* of SMIL but also the *why*, the background information in each chapter provides the motivations and limitations of SMIL technology. Note that for chapters that explain lots of elements and/or attributes, the examples are presented in the context of the definition of the constructs. For chapters covering many concepts, the chapter structure is sometimes repeated as a local sub-structure instance.[1]

Notational Conventions

In order to provide a predictable presentation of SMIL constructs, this book adopts the following simple notational conventions:

- *Element definitions*: SMIL's XML elements are displayed in a mono-space font, together with their angle brackets. Examples are: `<body>`, `<transition>`, `<switch>`. Unless used in the content of an example, a trailing '/' is not used.
- *Attribute definitions*: SMIL's XML attributes are displayed in a mono-space font, without other ornamentation. Examples are: `clipBegin`, `dur`, `begin`.
- *Attribute values*: SMIL's XML attribute value definitions are displayed in an separate italic font. Examples are: *indefinite*, *media*, *auto*.
- *Code examples*: The code examples in the book are set in separate areas and have a colored background. (See the XML-like code on the previous page for an example.) When code fragments are embedded in a sentence, the fragment is set in a single mono-space font. An example: `src="snarf.mpg"`.

While the integration of proportional and mono-spaced fonts on the same line sometimes yields strange spacing artifacts, we have done our best to minimize these in the text. Our goal has been to provide clarity without generating unnecessary white space. (The book is long enough as it is!)

In order to facilitate the search for constructs in the text, each first definition of an element, attribute or special value is highlighted with a separate heading. Examples are:

Element: <audio>
Attribute: clipEnd
Value: *auto*

These definitions are placed in the Index as the primary reference for the element, attribute or value construct.

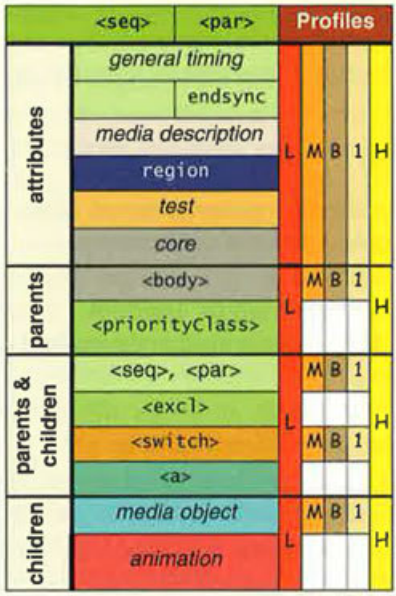
---

1 If all of this talk of elements, attributes and structure has you confused, make sure you read the introduction to XML structure in Chapter 2: *Understanding SMIL 2.0 Code*!

Element and Attribute Tables

Most chapters in *Part Two* and *Part Three* of this book contain element and attribute tables. These tables define either an element structure or a set of values for complex SMIL attributes.

An example of an element table is given below.



The table gives the name of the element and defines the major attributes, parents and children for that construct. It also defines which SMIL profiles make use of the construct's components. Each profile is represented by a single letter and a background color. The decoding of the profile colors and letters is:

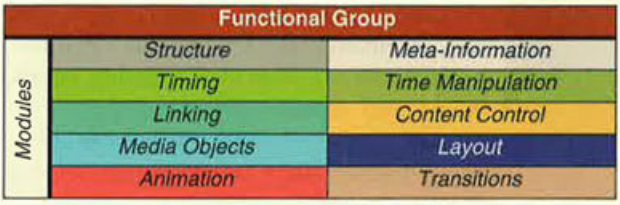
L	SMIL 2.0 Language profile
M	3GPP SMIL Mobile profile
B	SMIL 2.0 Basic profile
1	SMIL 1.0 Language
H	XHTML+SMIL

Those SMIL constructs used by the SVG language are indicated as:

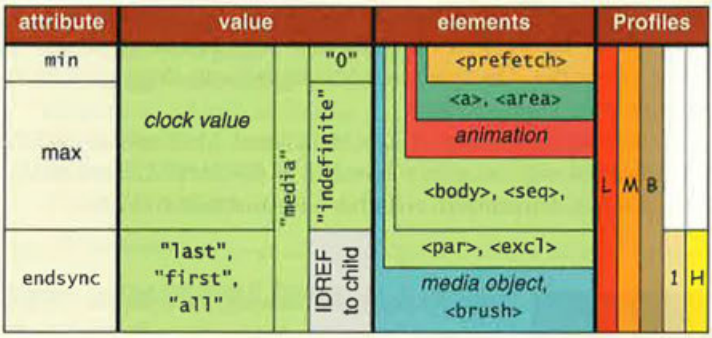
S	SVG pseudo-profile

Note that for the 3GPP SMIL mobile profile, several versions of the 3GPP standards exist. We use 3GPP/PSS5 as a guide.

Each of the attributes contain a background color code that defines the functional group to which they belong. When appropriate, a distinction is made between basic and advanced attributes by giving basic attributes a lighter tint. The color coding is:



An example of an attribute table is shown below.



In each attribute table, the default values for the attribute are shown in bold.

Quick Tips

In addition to being a reader's guide to the SMIL specification, our book is geared to helping you create SMIL presentations. Each section contains a number of *Quick Tips* that can be useful when you first work with SMIL. These tips, together with the examples, will get you started with multimedia on the Web in the shortest possible time.

Quick Tip

This is the format of a Quick Tip block.

On-Line Information

We maintain a Web site for this book at `http://www.XmediaSMIL.net/`. This site contains full versions of all of the examples used in our book plus links to various sites of interest. It will also contain a corrections and errata section that contains fixes for errors that inadvertently made their way into this publication.

About the Authors



Dick Bulterman is head of the research theme Convergent Media Interfaces at CWI, the Dutch national center for mathematics and computer science in Amsterdam. He joined CWI in 1988 as head of the Computer Systems and Telematics department. In 1991, he founded the Multimedia and Human-Computer Interaction research theme and directed the development of CWI's CMIF research from 1991-1998. From 1998-2001, he was managing director and CTO of Oratrix Development, a CWI spin-off company specializing in full-featured SMIL authoring systems and custom SMIL player engines.

Bulterman received a Ph.D. in computer science from Brown University in 1981. Before joining CWI, he was on the faculty of the division of Engineering at Brown, where he worked on computer architecture and high-speed signal processing networks. He holds an undergraduate degree in economics and mathematics from Hope College.

Dick Bulterman joined the W3C's Structured Multimedia (SYMM) working group in 1996. He was an active member of the SMIL 1.0 and SMIL 2.0 design teams. He lives in Amsterdam with his wife and two children.



Lloyd Rutledge is a researcher with the Multimedia and Human-Computer Interaction theme at CWI in the Netherlands. His research area is the semi-automatic generation of multimedia presentations. He received his Sc.D in computer science from the University of Massachusetts Lowell in 1996. He is CWI's advisory committee representative to W3C. Like his co-author, Lloyd Rutledge was part of the SYMM working group of the W3C, participating the development of SMIL 1.0 and 2.0. He lives with his wife in Amsterdam.

Table of Contents

Dedications v

Preface vii
Motivation for this Book vii
Should You Read This Book? vii
Structure of the Book viii
Notational Conventions ix
Element and Attribute Tables x
Quick Tips xi
On-Line Information xi
About the Authors xii

Table of Contents xiii

Common Acronyms and Terms xxvii

Acknowledgements xxix

Part One Introduction 1

1 An Overview of SMIL 2.0 3
1.1 Using SMIL 2.0: Six Examples 3
1.1.1 Hi Mom! 3
1.1.2 Welcome to New York 4
1.1.3 Happy Birthday! 5
1.1.4 Crossing the Bridge 6
1.1.5 The Evening News 7
1.1.6 Flags 8
1.2 SMIL 2.0 and SMIL 1.0 9
1.3 SMIL 2.0's Design Goals 9
1.4 SMIL Modules and Profiles 10
1.4.1 SMIL 2.0 Language Profile 12
1.4.2 SMIL 2.0 Basic 12
1.4.3 3GPP/PSS SMIL 13
1.4.4 XHTML+SMIL Profile 13
1.4.5 SMIL 1.0 14
1.4.6 SVG 14
1.4.7 XMT 14

1.5 Creating SMIL Presentations 15
1.5.1 Notepad (and Other Text Editors) 15
1.5.2 SMIL Syntax Validator 15
1.5.3 Perly SMIL 16
1.5.4 SMILGen 16
1.5.5 MAGpie 16
1.5.6 GRiNS 16
1.6 SMIL and Other Specifications 16
1.6.1 CMIF 17
1.6.2 Dynamic HTML 17
1.6.3 Flash 17
1.6.4 MPEG-4 17
1.7 Summary and Conclusion 18
1.8 Further Resources 19

2 Understanding SMIL 2.0 Code 21
2.1 Understanding XML Structure 21
2.1.1 XML Elements 22
2.1.2 XML Attributes 23
2.1.3 XML References 24
2.2 Flashlight: A SMIL Example Presentation 26
2.2.1 The Head and Body Sections 30
2.2.2 Media Content 31
2.2.3 Layout 32
2.2.4 Timing 34
2.2.5 Linking 36
2.2.6 Adaptivity 37
2.2.7 Putting it All Together 38
2.3 Encoding Flashlight in Different Profiles 40
2.3.1 SMIL Mobile Profiles 40
2.3.2 SMIL 1.0 Profile 44
2.3.3 XHTML+SMIL Profile 46
2.4 Summary and Conclusions 52
2.5 Further Resources 52

3 Local and Streaming Media 55
3.1 Classification of Media 55
3.2 Media Formats and Encodings 56
3.2.1 Media Formats for Compressed Discrete Media 57
3.2.2 Media Formats for Compressed Continuous Media 59
3.2.3 Tools for Creating Media 61
3.3 URIs: Locating and Naming Media Objects 62
3.3.1 Local URIs 63
3.3.2 Remote URIs 63
3.3.3 Relative URIs 64
3.3.4 Passing Parameters in URIs 65

3.4 Accessing Media Within a SMIL Presentation 65
3.4.1 Local access to media 65
3.4.2 Remote Access to Media 66
3.5 Summary and Conclusion 70
3.6 Further Resources 71

Part Two Basic SMIL Constructs 73

4 SMIL Structure 75
4.1 SMIL Language Structure Issues 75
4.2 SMIL Structure Elements and Attributes 76
4.2.1 SMIL Document Type Declarations 76
4.2.2 SMIL Top-Level Structure Containers 77
<smil> 77
<head> 78
<body> 79
4.2.3 Core Structure Attributes 80
id 80
xml:base 80
4.2.4 Core Attributes for Accessibility 80
4.2.5 Namespace Attributes 80
xmlns 81
4.3 Summary and Conclusion 82
4.4 Further Resources 82

5 Referencing Media Objects 85
5.1 Partitioning Content and Control 85
5.2 SMIL Media Object Elements and Attributes 88
5.2.1 Media Elements 88
<ref> 88
<brush> 89
5.2.2 Media Object Attributes 90
src 90
type 92
clipBegin and clipEnd 94
color 95
5.3 Timing Consequences of Media Object Use 97
5.4 Summary and Conclusion 98
5.5 Further Resources 98

6 SMIL Basic Layout 99
6.1 An Overview of Layout Concepts 99
6.2 SMIL Layout Models 102
6.2.1 SMIL 1.0's Approach to Layout 102
6.2.2 SMIL 2.0's Approach to Layout 103
6.2.3 Summary of SMIL Layout Concepts 104

6.3 SMIL Basic Layout Elements and Attributes 105
6.3.1 SMIL Basic Layout Elements 106
<layout> 106
<root-layout> 107
<region> 108
6.3.2 Basic Layout Attributes Architecture 109
6.3.3 <layout> Attributes 109
6.3.4 <root-layout> Attributes 110
height and width 110
backgroundColor 110
6.3.5 <region> Referencing Attributes 111
id 111
regionName 111
6.3.6 <region> Positioning Attributes 111
top and left 112
width and height 112
bottom and right 113
auto 113
6.3.7 Background Attributes 113
backgroundColor 113
showBackground 114
6.3.8 Stacking and Scaling Attributes 114
fit 114
z-index 116
6.3.9 Referencing Regions from Media Objects 117
region 117
6.3.10 SMIL Empty Layout Semantics 117
6.4 SMIL Audio Layout Elements and Attributes 117
6.4.1 Elements for Audio Layout 117
6.4.2 Attributes for Audio Layout 117
soundLevel 118
6.5 Examples Using SMIL Basic Layout 118
6.5.1 Defining the <layout> Section 118
6.5.2 Supporting Multiple Layouts in One Presentation 120
6.5.3 Referencing Media in the Presentation 120
6.5.4 Adjusting Rendering Properties 121
6.5.5 Rendering Multiple Copies of a Single Media Object 122
6.6 Summary and Conclusion 123
6.7 Further Resources 123
7 Basic SMIL Timing 125
7.1 SMIL Timing Model Basics 126
7.1.1 A Simple Slideshow Presentation 126
7.1.2 Media Object and Presentation Timing Definitions 127
7.1.3 Durations, Time and Timebases 131
7.1.4 Special Timing Values 135
indefinite 135
media 136
7.1.5 Interactive Timing and Events 136

7.2 Basic SMIL Timing Elements 136
7.2.1 Basic Time Containers 137
<par> 137
<seq> 138
7.2.2 Nested Composition of Timing Elements 138
7.3 Basic SMIL Timing Attributes 139
7.3.1 General Timing Control Attributes 141
begin 142
dur 143
end 145
7.3.2 Object Persistence Attributes 147
fill 147
7.3.3 Extended Timing Control Attributes 148
min 149
max 150
endsync 150
7.3.4 Repeating Objects and Sub-Structures 151
repeatCount 151
repeatDur 152
7.3.5 Advanced Timing and Synchronization Attributes 153
7.3.6 The General Timing Attributes 154
7.4 Summary and Conclusion 154
8 Basic Linking 157
8.1 An Overview of Linking Concepts 157
8.1.1 Links in HTML 158
8.1.2 SMIL: Adding Time to Links 161
8.2 SMIL Basic Linking Elements 162
<a> 162
<area> 163
<anchor> 166
8.2.1 SMIL Basic Linking Attributes Architecture 166
8.2.2 Specifying the Activation Period of Source Anchors 166
8.2.3 The Temporal Moment of the Destination Anchor 166
8.2.4 Attributes for Link Anchor Geometries 168
shape 168
coords 168
8.2.5 Attributes to Control the Playstate After Activation 170
destinationPlaystate 170
show 171
sourcePlaystate 171
8.2.6 Other Linking Control Attributes 171
actuate 172
target 172
external 172
sourceLevel and destinationLevel 172
alt 172
tabindex 172
accesskey 173

8.3 Other Uses of Links in SMIL 173
8.3.1 SMIL Linking and Activation Control 173
8.3.2 SMIL Linking and Message Passing 175
8.4 Summary and Conclusion 175
8.5 Further Resources 176

9 Content Selection and Control 177
9.1 Content Selection: Rationale and Scope 177
9.1.1 Understanding the Runtime Environment 177
9.1.2 Techniques for Creating Adaptive Presentations 180
9.2 Content Control Elements and Attributes 182
9.2.1 SMIL Basic Content Control Elements 182
<switch> 182
9.2.2 SMIL Basic Content Control Attributes Architecture 184
9.2.3 In-Line Use of System Test Attributes 185
9.2.4 User-Related System Test Attributes 186
systemLanguage 186
systemCaptions 187
systemOverdubOrSubtitle 188
systemAudioDesc 188
9.2.5 Environment-Related System Test Attributes 189
systemBitrate 189
systemScreenDepth 190
systemScreenSize 190
9.2.6 Resource-Related System Test Attributes 191
systemCPU 192
systemOperatingSystem 192
systemComponent 193
systemRequired 193
9.2.7 SMIL 1.0 Test Attribute Support 194
9.2.8 Static and Dynamic System Test Attribute Evaluation 194
9.3 Examples Using SMIL Content Control 195
9.3.1 Media and Presentation Design 195
9.3.2 The Structure of the Presentation 196
9.3.3 Using SMIL to Define the Adaptive Presentation 197
9.4 Summary and Conclusion 200
9.5 Further Resources 200

10 Transition Effects 203
10.1 Concepts and Models of Transition Effects 203
10.1.1 Basic Transition Effects Concepts 204
10.1.2 Transition Models 205
10.1.3 Basic and Inline Transitions 209
10.2 Basic Transitions Elements and Attributes 209
10.2.1 Basic Transition Elements 210
<transition> 210

10.2.2 General Attributes for Basic Transitions 210
type 210
subtype 211
dur 211
startProgress 211
endProgress 211
direction 212
10.2.3 Type-Specific Attributes for Basic Transitions 212
fadeColor 212
10.2.4 Attributes for Integrating Basic Transitions 213
transIn 213
transOut 214
10.3 Inline Transitions Elements and Attributes 215
10.3.1 Inline Transition Elements 215
<transitionFilter> 216
10.3.2 General Attributes for Inline Transitions 216
type 216
subtype 216
mode 217
10.3.3 Timing Attributes for Inline Transitions 217
begin 217
dur 218
end 218
repeatCount 218
repeatDur 218
10.3.4 Animation Attributes for Inline Transitions 218
from 218
to 218
by 219
values 219
calcMode 219
10.3.5 Type-Specific Attributes for Inline Transitions 219
fadeColor 219
10.3.6 Attributes for Integrating Inline Transitions 220
targetElement 220
href 220
10.4 Modifying Basic and Inline Transitions 221
10.4.1 Common Attributes for Modifying Transitions 221
horzRepeat 221
vertRepeat 221
borderColor 221
borderWidth 221
10.5 Examples Using SMIL Transition Effects 221
10.5.1 Examples of Basic Transitions 221
10.5.2 Examples of Inline Transitions 224
10.6 Issues for Supporting Transition Effects 226
10.6.1 SMIL Timing and Transitions 226
10.6.2 The <par> Element and Transitions 228

10.6.3 The <seq> Element and Transitions 229
10.6.4 The <excl> Element and Transitions 229
10.6.5 Audio and Transitions 230
10.7 Transition Classes, Types and Sub-Types 230
10.7.1 Required Types/Sub-Types 230
10.7.2 Edge Wipe Transition Types and Sub-Types 231
10.7.3 Iris Wipe Transition Types and Sub-Types 232
10.7.4 Clock Wipe Transition Types and Sub-Types 233
10.7.5 Matrix Wipe Transition Types and Sub-Types 234
10.7.6 SMIL-Specific Transition Types and Sub-Types 235
10.8 Summary and Conclusion 235
10.9 Further Resources 235

Part Three Advanced SMIL Features 237

11 Subsetting and Extending Media 239
11.1 Cropping Media Spatially 239
11.1.1 Understanding Spatial Cropping for Linking 240
11.1.2 Elements for Spatial Cropping 242
<area> 242
11.1.3 Attributes for Spatial Cropping 242
shape 242
coords 242
11.2 Clipping Media in Time 243
11.2.1 Understanding Temporal Clipping of Media 243
11.2.2 Attributes for Temporal Clipping 244
clipBegin 244
clipEnd 245
11.2.3 Examples of Temporal Clipping 246
11.3 Clipping Media Using Media Markers 246
11.3.1 Understanding Media Clip Markers 247
11.3.2 Attribute Extensions for Media Markers 248
11.3.3 Examples Using Media Markers 249
11.4 Subsetting Media Via Fragment Identifiers 249
11.4.1 Understanding Fragment Identifiers 249
11.4.2 Language Support for Fragment Identifiers 250
11.4.3 Examples Using Fragment Identifiers 250
11.5 Initialization Attributes for Media Objects 251
11.5.1 Understanding the Use of Media Parameters 251
11.5.2 Elements for Media Initialization 252
<param> 252
11.5.3 Attributes for Media Initialization via <param> 252
name 252
value 253
valuetype 253
11.5.4 Attributes for Player Media Initialization 253
mediaRepeat 253

erase 253
sensitivity 254
11.5.5 Examples of Media Initialization 254
11.6 Summary and Conclusion 256
11.7 Further Resources 257

12 Advanced Layout Topics 259
12.1 Logical Media Object Alignment in Regions 260
12.1.1 Understanding Registration and Alignment Points 260
12.1.2 Elements for Layout Alignment 262
<regPoint> 262
12.1.3 Attributes for Layout Alignment 263
regAlign 263
regPoint 263
12.1.4 Other Aspects of Registration Point Alignment 264
12.1.5 Examples of Registration Point Alignment 265
12.2 Sub-Region Positioning 267
12.2.1 Understanding Sub-Region Positioning 267
12.2.2 Sub-Region Positioning Elements 268
12.2.3 Sub-Region Positioning Attributes 268
12.2.4 Examples of Sub-Region Positioning 269
12.3 HierarchicalLayout 271
12.3.1 Understanding Hierarchical Layout 271
12.3.2 Hierarchical Layout Elements 272
<region> 272
12.3.3 Hierarchical Layout Attributes 272
12.3.4 Examples of Hierarchical Layout 273
12.4 Media Object Overrides of Layout Positioning 274
12.5 Defining Multiple Top-Level Windows 274
12.5.1 Understanding Multiple Top-Level Windows 274
12.5.2 Multi-Window Elements 276
<topLayout> 276
12.5.3 Multi-Window Attributes 278
open 278
close 278
12.5.4 Aspects of Multi-Window Layout 279
12.5.5 Examples of Multi-Window Layout 280
12.6 SMIL Layout and CSS 281
12.6.1 CSS Attributes for SMIL Layout 281
class 282
12.6.2 CSS and SMIL Layout Comparison 282
12.7 Summary and Conclusion 282
12.8 Further Resources 284

13 Advanced SMIL Timing Behavior and Control 285
13.1 Event Based Timing 286
13.1.1 Understanding Events, Links and Interaction 286

13.1.2 Event Support in SMIL Profiles 292
13.1.3 Examples of Event Syntax and Semantics 294
13.2 Selective Time Composition 297
13.2.1 Understanding Selective Composition 297
13.2.2 Elements for Selective Composition 300
<excl> 301
<priorityClass> 302
13.2.3 Attributes for Selective Composition 303
peers 303
higher 304
lower 304
pauseDisplay 304
13.2.4 Examples of Selective Temporal Composition 305
13.3 SMIL 2.0 Timing Model, Revisited 310
13.3.1 Element Activation, Duration and Termination 310
13.3.2 Determining Schedules 313
13.3.3 Pathological Examples 316
13.4 Integrating SMIL Timing in Other Languages 318
13.4.1 Time Container Integration Issues 318
13.4.2 Time Container Integration Attributes 319
timeContainer 319
timeAction 320
13.4.3 Examples of Time Container Integration 321
13.5 Summary and Conclusion 325
13.6 Further Resources 326
14 Advanced SMIL Timing Attributes 327
14.1 Fill Behavior Control 328
14.1.1 Understanding the Fill Behavior 328
14.1.2 Fill Behavior Attributes 333
fill 333
fillDefault 334
14.1.3 Examples of Fill Behavior 334
14.2 Restart Behavior 341
14.2.1 Understanding Restart Behavior 341
14.2.2 Restart Behavior Attributes 343
restart 343
restartDefault 344
14.2.3 Examples of Restart Behavior 344
14.3 Synchronization Behavior Control 347
14.3.1 Understanding Synchronization Behavior Control 347
14.3.2 Synchronization Behavior Attributes 350
syncBehavior 350
syncBehaviorDefault 351
syncTolerance 352
syncToleranceDefault 352
syncMaster 353
14.3.3 Examples of Synchronization Behavior Control 353

14.4 Time Manipulation 354
14.4.1 Understanding Time Manipulations 355
14.4.2 Time Manipulation Attributes 357
autoReverse 357
speed 357
accelerate 358
decelerate 358
14.4.3 Examples of Time Manipulation 358
14.5 Summary and Conclusion 360

15 SMIL Animation 361
15.1 Overview of SMIL Animation 362
15.1.1 The SMIL Animation Model 362
15.1.2 SMIL Animation and the <animation> Element 367
15.2 Basic Animation Elements and Attributes 367
15.2.1 Basic Animation Elements 367
<animate> 368
<animateMotion> 368
<animateColor> 368
<set> 368
15.2.2 Basic Animation Attributes 369
15.2.3 Attributes for Specifying the Animation Target 369
targetElement 370
attributeName 370
attributeType 371
15.2.4 Attributes for Providing Animation Values 371
from 371
to 372
by 372
values 373
15.2.5 Attributes for Defining the Animation Function 373
additive 373
accumulate 374
calcMode 374
origin 374
15.3 Support for SMIL Spline Animation 375
15.3.1 Spline Animation Elements 375
15.3.2 Spline Animation Attributes 375
15.3.3 Attributes for Spline Specification 375
calcMode 376
keyTimes 376
keySplines 376
path 376
15.4 Examples of SMIL Attribute Animation 376
15.5 Animation Support in SMIL Profiles and SVG 381
15.5.1 Animation and the SMIL 2.0 Language Profile 382
15.5.2 Animation and the XHTML+SMIL Profile 383
15.5.3 Animation and SVG 383

15.6 Summary and Conclusion 384
15.7 Further Resources 384

16 Extended Content Control 385
16.1 Skip Content Control 385
16.1.1 Motivation: Future-Proofing SMIL 386
16.1.2 Skip Content Control Attributes 386
skip-content 386
16.1.3 Using the skip-content Attribute 387
16.2 Custom Test Attributes 389
16.2.1 Understanding Custom Test Attributes 389
16.2.2 Custom Test Elements 390
<customAttributes> 390
<customTest> 390
16.2.3 Custom Test Attributes 391
customTest 391
defaultState 391
override 391
uid 391
16.2.4 Examples Using Custom Test Attributes 392
16.3 Prefetch Control 394
16.3.1 Understanding Pre-Fetching of Media 394
16.3.2 Pre-Fetch Elements 395
<prefetch> 395
16.3.3 Pre-Fetch Attributes 395
mediaSize 395
mediaTime 396
bandwidth 396
16.3.4 Using Pre-Fetch Control 396
16.4 Summary and Conclusion 397
16.5 Further Resources 397

17 Meta-Information, Media Description and XML Accessibility 399
17.1 Meta-Information 399
17.1.1 Elements for Meta-Information 399
<metadata> 400
<meta> 400
17.1.2 Attributes for Meta-Information 400
name 400
content 401
17.1.3 Examples Using Meta-Information 401
17.2 Media Description 402
17.2.1 Media Description Attributes 403
abstract 403
author 403
copyright 403
title 403
xml:lang 403
17.2.2 Example Using Media Object Descriptions 404

17.3 XML Accessibility 404
17.3.1 Media Accessibility Attributes 404
alt 404
longdesc 405
readIndex 405
17.3.2 Using Media Accessibility Attributes 405
17.4 Summary and Conclusion 406
17.5 Further Resources 406

Part Four SMIL Family Reference 409

A SMIL 2.0 Language Profile Module Reference 411

B 3GPP SMIL Module Reference 413

C SMIL 2.0 Basic Profile Module Reference 415

D XHTML+SMIL Module Reference 417

E SMIL in SVG 421

F SMIL 2.0 Module and Construct Chart 423

Index 427

Common Acronyms and Terms

The following list of acronyms and common names are used throughout the book. They are usually identified in the text by the use of a SMALL CAPS font.

ADSL	Asymmetric Digital Subscriber Line
Ambulant	The CWI open-source player for SMIL 2.0
CDATA	Character Data
CSS	Cascading Style Sheets
DOCTYPE	Document Type definition string identifier
DTD	Document Type Definition
GR*i*NS	The Oratrix GR*i*NS player for SMIL 2.0
HTML	HyperText Markup Language
HTML+TIME	Microsoft's XHTML+SMIL player in IE-6
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
ID	Identifier string
IDREF	Reference to an ID
IE	The Microsoft Internet Explorer browser
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
MPEG-4	Multimedia standard developed by the Motion Picture Experts Group
NTSC	National Television Standards Committee *North-American and Asian Television Standard*
PAL	Phase Alternation Line *World-Wide Television Standard*
RealOne	The RealNetworks RealOne player for SMIL 2.0
RSVP	Resource Reservation Protocol
RTCP	Real-Time Control Protocol
RTP	Real-Time Transfer Protocol
RTSP	Real-Time Streaming Protocol
SMIL	Synchronized Multimedia Integration Language

SMPTE	Society of Motion Picture and Television Engineers
SVG	Scalable Vector Graphics language
TCP	Transmission Communications Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XHTML	XML-Compliant Hypertext Markup Language
XHTML+SMIL	The XHTML+SMIL profile of SMIL 2.0
XML	Extensible Markup Language

Acknowledgements

During the production and review of this book, we received a great deal of assistance from many people. Along with our families, we would like to thank the following organizations and individuals for their support.

Reviews of early drafts and the final technical reviews were performed by:

- *Sjoerd Mullender* of Oratrix/CWI;
- *Jack Jansen*, *Kleanthis Kleanthous* and *Kees Blom* of the GRiNS and Ambulant player development teams at Oratrix and CWI;
- *Michael Minnema;* and
- *Luiz Fernando Gomes Soares* and *Rogerio Rodrigues* of PUC-Rio.

While we are grateful to all of these people, the technical reviews performed by Sjoerd stood out in this high-class crowd. Sjoerd was the design and implementation leader of the GRiNS SMIL 1 and SMIL 2 players (which were also implemented by Jack Jansen and Kleanthis Kleanthous), and his energy and willingness to share his knowledge of even the most minor details of SMIL was a tremendous help in completing this manuscript.

Additional technical information and assistance for various SMIL profiles came from:

- *Jeff Ayars*, *Rob Lanphier* and *Eric Hyche* of RealNetworks, Inc.
- *Philipp Hoschka*, *Thierry Michel*, *Ivan Herman* and *Steven Pemberton* of W3C;
- *Aaron Cohen* of Intel and the Oregon State Health University;
- *Patrick Schmitz*, Microsoft (now at ludicrum.org);
- *Debbie Newman* and *Pablo Fernicola* of Microsoft;
- *Nick Dyer* and *Phil Sheppard* of Three/Hutchison 3G UK Limited;
- *Nick Roberts* of Vodafone; and
- *Antti Koivisto* of Nokia.

Out of this group, we are especially indebted to Aaron Cohen for his skillful leadership of the SMIL 2.0 working group. Without him, there probably would have been no SMIL 2.0 — and thus, no SMIL 2.0 book!

Many people supported the production of this manuscript, including:

- *Alain Uginet*, *Jeroen Tak* and *Katelijn Arnold* of Oratrix;
- *Teus Hagen* and *Frances Brazier* of Stichting NLnet;

- *Jan Schipper* and *Tobias Baanders* of CWI;
- *Michael Strang* and *Simon Plumtree* of Pearson Education Ltd.; and
- *Jutta Maria Fleschutz* and *Gabriele Fischer* of Springer Verlag.

We are very grateful for the flexibility offered by Springer in providing us with the opportunity to produce a full-color book and to bring various versions of the book to market. The nice chapter and margin icons were designed by Tobias Baanders. The rest were done by Dick Bulterman and Lloyd Rutledge.

We would also like to thank those organizations and people who allowed us to use their content in illustrations in our book.

- Microsoft Corporation;
- *Geoff Freed* of NCAM/WGBH;
- *Jacoba Alida, Henri J. François* and *Johanna Wilhelmina* of Singel Media;
- *Henri "Peaches" Bulterman* of Marietta, Georgia; and
- RTV Slovenia.

Finally, we thank our co-members of the W3C/SYMM team.

Introduction

This part consists of introductory and background information on SMIL 2.0. This section contains enough information to get you familiar with SMIL, including the basic constructs in the language and the XML environment in which SMIL is specified. It also contains a complete SMIL example that illustrates many of SMIL's main features.

Topics Covered



Chapter 1 provides a general introduction to SMIL and its various implementations. It illustrates what SMIL can do by sketching six applications of SMIL technology. It also describes the background of SMIL and contrasts SMIL with other multimedia formats. It also contains a review of authoring tools and players for SMIL 2.0.



Chapter 2 looks at the structure of SMIL presentations. We start with a brief review of the XML language and discuss how XML elements and attributes form the core of SMIL's definition. We then look at how SMIL presentations can be encoded for each of the SMIL 2.0 profiles.



Chapter 3 provides general information on encoding and accessing multimedia information in Web and stand-alone environments. This section contains background context that explains the operational considerations that need to be addressed when designing presentations for the Web.

Goals

By the time you finish this part, you should be able to look at any SMIL file and understand its basic structure. This is true for not only the SMIL 2.0 Language profile, but also for SMIL presentations written for the XHTML+SMIL and SMIL Mobile profiles.

Prerequisites

In order to get the most out of this part, you should have some affinity with the Web and the Internet. While experience with creating and sharing digital media is not essential, a basic understanding of how media is encoded and how Web servers and clients are structured will be helpful in putting information in this section to immediate practical use.

1

An Overview of SMIL 2.0





In August 2001, the World Wide Web Consortium (W3C) released version 2.0 of Synchronized Multimedia Integration Language, or SMIL (pronounced "smile"). SMIL 2.0 was developed to bring presentation-level interactive multimedia to the Web. It is a text-based, non-proprietary, XML standardized format that is woven into the W3C's XML-based family of cooperative and interdependent languages. SMIL 2.0 is the only format for encoding multimedia presentations that is released and supported by the W3C.

SMIL 2.0 is not an animation language or a media format, but a way of combining animation and media objects into a single coherent presentation. SMIL lets you add sophisticated selection, linking and activation control to the media objects in your presentation; in this way, it is a bit like a multimedia version of HTML. Unlike HTML or XHTML, however, SMIL has a stricter separation between content and structure and it provides non-invasive control of presentation linking and layout.

1.1 Using SMIL 2.0: Six Examples

Before jumping into the details of what SMIL 2.0 can do, this section gives an overview of six presentation examples that illustrate SMIL behavior from the presentation of a single media item to the definition of a complete interactive newscast. We will return to some of these examples later in this book.

Quick Tip

Each of the presentations discussed in this book can be found on our Web site: `http://www.XmediaSMIL.net`, in the section on book demos. We suggest that you play these demos to better understand the example and discussion presented in each chapter.

1.1.1 Hi Mom!

The first presentation represents perhaps the most pervasive use of SMIL in current players: the presentation of a single piece of video content. Figure 1-1 shows how a single media object can be placed in a player window and then

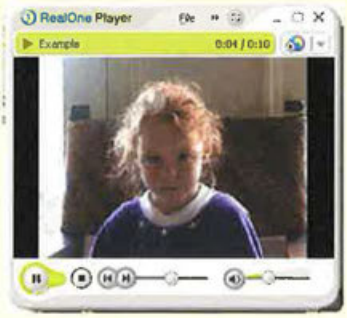


Figure 1-1. *Hi Mom!*, a single media presentation.

presented to a wide audience. The movie content can range from home videos to commercial content. The video may also be accompanied by audio.

For this example to work in a streaming environment, you need to place the media content on a streaming server and then access it via the SMIL player.

1.1.2 Welcome to New York

The second SMIL example is a slideshow that gives a short tour of New York. It is typical of an entry-level use of multimedia SMIL. Figure 1-2 gives two views of the presentation: one on a desktop player (using RealOne) and the other on a PDA SMIL player (using GRiNS) that supports the 3GPP mobile implementation of SMIL. (Don't worry about what 3GPP means; we'll get back to the various implementations of SMIL later in this chapter.)



Figure 1-2. *Welcome to New York*, a simple SMIL slideshow presentation.

The *New York* presentation consists of a sequence of slide objects, each of which contains an image, a text caption and an audio caption. A single piece of background audio plays as a background tune while each of the slide objects is shown.

Although the structure of this presentation is pretty straightforward, it is too complex for the most simple implementations of SMIL. In the MMS standard for mobile SMIL, for example, each of the slide objects could be shown (containing text, audio and image), but the background audio music would not be supported.

The *New York* presentation uses functionality that is essentially similar to that of SMIL 1.0. The most striking 'new' feature is the support for transitions between images, which gives the entire presentation a smoother visual feeling. In many ways, the combination of images and transitions provides many of the advantages of video at only a fraction of the bandwidth cost!

1.1.3 Happy Birthday!

The *Welcome to New York* SMIL demo shows a linear presentation in which the order of the content is determined by the author. The *Happy Birthday!* presentation demonstrates a presentation in which the viewer can select the content to be viewed.

Figure 1-3 shows two implementations of *Birthday!*: the version at left is for the SMIL 2.0 Language profile (and is rendered in the RealNetworks RealOne player), while the version at right is for the XHTML+SMIL profile (and is rendered in Microsoft's Internet Explorer browser). Both versions provide identical functionality — the viewer can select one of the four images on the left side, at which point a video is shown in the main display area on the right— but as we will see in Chapter 2: *Understanding SMIL 2.0 Code*, the encodings of the native SMIL presentations are totally different.

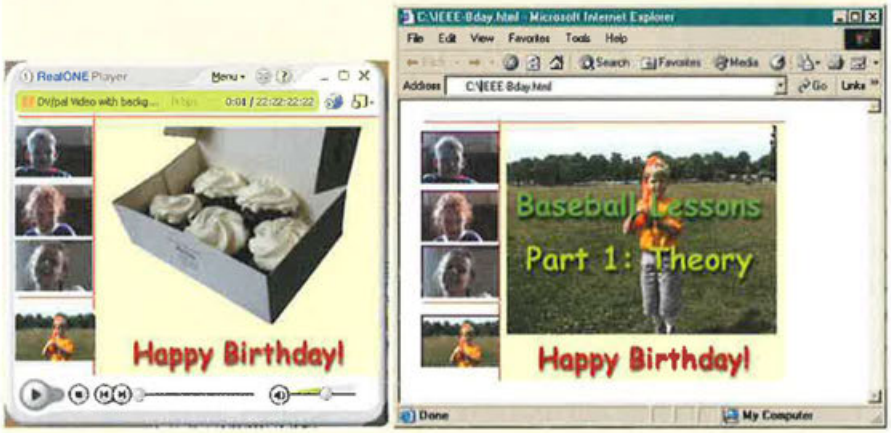


Figure 1-3. *Happy Birthday!*, as shown in native SMIL and XHTML+SMIL players.

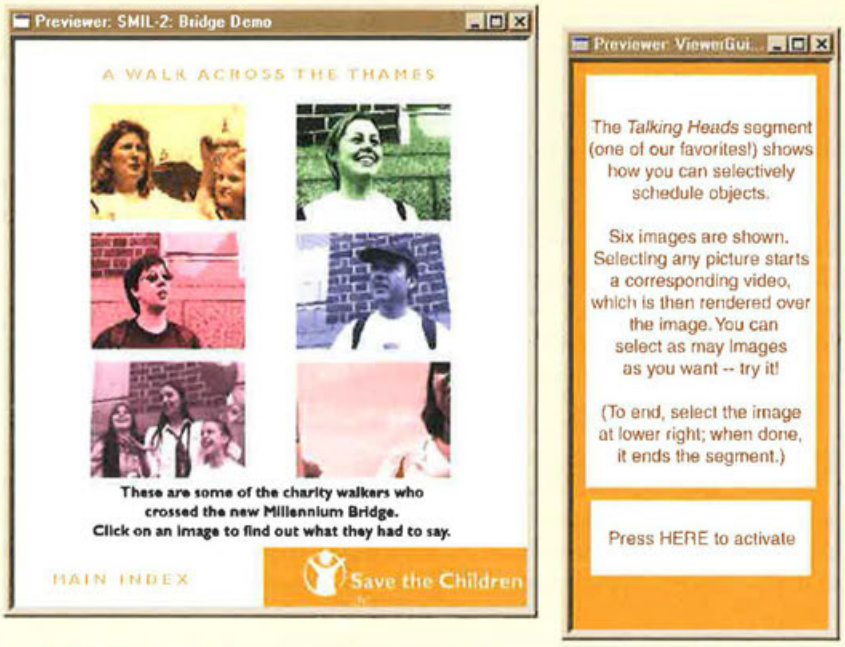


Figure 1-4. *Crossing the Bridge*, a multi-window, interactive presentation.

1.1.4 Crossing the Bridge

Although the SMIL 2.0 specification was approved by the W3C as a Recommendation in mid-August, 2001, the foundation for the specification was defined in mid-2000. Based on the interim specification, the first full SMIL 2.0 presentation was released in September 2000. *Crossing the Bridge*[1] was a collection of test cases that allowed SMIL 2.0's timing, interaction, layout, transitions and animation constructs to be evaluated by candidate SMIL 2.0 players.

Figure 1-4 shows a view of the presentation as rendered by the Ambulant desktop player. Two windows are shown: the main content window (shown at left) and a second window containing help text and general presentation control (at right). The two windows are used to partition presentation navigation and control. They are presented as two independent windows to the user, but they are still logically connected: the content across the multiple windows shares a common presentation timeline.

1 The *Bridge* presentation showcased the opening of the Millennium Bridge across the river Thames in London. As part of the ceremonies, several thousand people came to London as part of a drive organized by charitable organizations, one of which — Save the Children, UK — was highlighted in the presentation. A day later, the bridge was closed to the general public for safety reasons.

In the presentation fragment shown, a user is able to select one of six images; clicking on any of the images starts a corresponding video clip that is displayed on top of the image. The viewer determines which (if any) of the mini-presentations get activated, as well as the order in which they are shown. (This functionality is implemented using SMIL 2.0's *exclusive* construct.) Note that the user must first select the *Press HERE to activate* button in the right window before any of the images are activated at left. The user also can choose to go to another part of the presentation by clicking on the *Main Index* label at lower left, or to pause the presentation and surf to the *Save the Children* Web site by selecting the image at lower right. (Both types of navigation are implemented using SMIL's linking functionality.)

The functionality in this example could also be supported in SMIL 1.0, but with substantially more author effort.

1.1.5 The Evening News

The use of multiple profiles, as shown in the *Birthday!* example above, gives one dimension of adaptability of a SMIL 2.0 source document. Within a profile, many opportunities also exist to make content that adapts itself to the needs (and wishes) of the viewer. As an example of what you can do with SMIL 2.0, consider the evening news example in Figure 1-5.

Figure 1-5(a) shows a "desktop" version of the newscast. The user is presented with a number of picture icons, each representing a segment of the 35-minute news. A news story is started when the user clicks on the associated picture. The user may also view the entire newscast in a predetermined



Figure 1-5. *The Evening News*, an interactive newscast that can be tailored for various display sizes and user preferences.



sequence or use the general navigation buttons at bottom right to navigate across stories or within an item.

In this picture, English language titles and captions are shown, although the user could also have selected the original Slovenian titles. Each of the assets for all the available languages and resolutions are encoded in a single SMIL file, with the SMIL player selecting the appropriate media based on configuration files or user preference dialogues.

Note the three icons at the bottom portion of Figure 1-5(a). When selected, each will start a video that will interrupt the running item, and when the interrupting object is completed, the original item will pick up where it left off in the presentation. (This can be used for advertising inserts or other out-of-context messages.)

Figure 1-5(b) shows the same presentation, but this time visually packaged for a mobile device. The structure of the mobile version is the same as the desktop version, except that general 'story classes', such as *National*, *Local*, and *Weather*, are used instead of the individual story icons. (This is to conserve screen space.) The rocker panel of the device can also be used to select stories.

Perhaps the most important aspect of this example is that both the desktop and the mobile versions are encoded in the same SMIL file. This allows the document to be authored once and used in multiple configurations.

1.1.6 Flags

The final presentation in this section illustrates the use of SMIL timing to animate graphics encoded in the Web graphics format SVG. Figure 1-6 shows two views of a single presentation. If the base language selected within the player is American English, the flag at left is shown. If any European language is selected, then the European flag at right is shown. Although it is difficult to see in this printed version, each of the flags has stars that rotate during the presentation — this rotation is accomplished by applying SMIL 2.0's animation facilities in the GRiNS embedded SVG player.

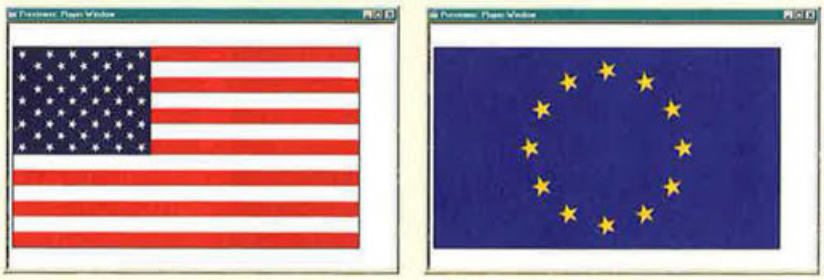


Figure 1-6. *Flags*, an example of SMIL-based animation in the SVG language.

1.2 SMIL 2.0 and SMIL 1.0



SMIL 2.0 is the successor to SMIL 1.0, which was the W3C's foundation for multimedia on the Web. During its three-year reign, SMIL 1.0 quickly gained widespread — if little-known! — use, primarily as the internal synchronization format for RealNetworks's RealPlayer. It was also supported in Apple's QuickTime player. Through these two multimedia players alone, SMIL 1.0 was installed on more than 200 million desktops worldwide.

SMIL 2.0 extends SMIL 1.0 in many key areas. The primary changes it provides are:

- improved synchronization and interactive activation facilities,
- a profile-based specification structure,
- new support for animation and media transitions, and
- extended support for content control and layout.

SMIL 2.0's elements and attributes are spread across 50 modules that are managed in over 10 functional groups. The partitioning of constructs makes it easy for the designers of new XML languages to select timing and presentation control primitives for integration and reuse.

The new functionality and specification structure in SMIL comes at a cost: where SMIL 1.0 was described in a document that spanned 29 pages, SMIL 2.0 needs over 500 pages to cover all of its elements and attributes. As a result, understanding how all the pieces fit together is not always an easy task. That's why this book was written: to provide a readable overview of SMIL's features and facilities, and to help designers create SMIL presentations for devices as wide-ranging as mobile telephones to sophisticated media servers.

This chapter provides a broad introduction to SMIL 2.0. Rather than jumping right into a discussion of SMIL code, we start with important background information that will help you understand SMIL in its various forms. We begin with a background discussion on the design goals that guided the development of SMIL 2.0. We then provide five examples that each illustrate different facets of SMIL functionality. A description of SMIL modularization and profiles comes next, followed by a sampling of some of the tools that are available to build and manage SMIL presentations. We close the chapter with a short comparison of SMIL and other important multimedia formats.

1.3 SMIL 2.0's Design Goals

SMIL 2.0 is a collection of XML elements and attributes that can be used to describe the temporal and spatial coordination of one or more media objects. SMIL 2.0 allows a user to define how independent media objects are to be integrated into a media presentation and it provides rules that define when (and if) various media objects are actually rendered for the end-user. The media presentation created by SMIL may be delivered via a streaming server or it may be played from components stored locally (or on a CD-ROM).



SMIL aims to do for multimedia what HTML did for hypertext: bring it onto every presentation device (PCs, mobile/handheld systems, even TVs) with an easy-to-author descriptive format that can be displayed on a variety of readily available cross-platform players. With SMIL, simple multimedia can be created simply and more complex behavior can be added in increments.

SMIL's syntax is fully XML compliant, making it a member of the family of XML standards from the W3C such as SVG, XPointer, XSLT, namespaces and XHTML (the XML encoding of HTML). SMIL provides the Web with new features not found in these other languages, such as adaptivity to different market groups, user abilities, system configurations and run-time system delays.

The goals for SMIL 2.0's design can be partitioned across three categories:

- *Maintain a declarative, XML format*: while the integration of multimedia content is often the province of scripts or other programmed definition, SMIL 2.0 was developed to remain a fully declarative rather than procedural format. In this way, a SMIL description doesn't define *how* a presentation is implemented, but rather *what* it is that the author wants — the implementation of the specification left up to the SMIL player. Also, in keeping with the first version of SMIL, SMIL 2.0 is fully XML compliant.[2]
- *Extend the functionality of* SMIL *1.0*: the design of SMIL 1.0 was kept simple and relatively frills-free. SMIL 2.0 provides some desirable additions, including: support for increased interaction, enhanced timing semantics, extended content control and layout facilities plus new animation and transitions primitives.
- *Introduce a module-based structure*: SMIL 2.0 is defined as a collection of over 50 modules that are partitioned into 10 functional groups. By using a module structure, key aspects of SMIL 2.0 can be integrated into other XML-based languages without requiring support for the entire SMIL 2.0 specification. Even before the release of the SMIL 2.0 Language specification, the modularization policy bore fruit: parts of SMIL were integrated into several other XML languages — such as SVG — and more examples of module reuse are expected.

In addition to these three classes of goals, SMIL 2.0 also removed some ambiguities that existed in SMIL 1.0 and SMIL 2.0 provided a more W3C-compliant approach to naming elements and attributes.

1.4 SMIL Modules and Profiles

As shown in Figure 1-7, SMIL 2.0 defines 10 major functional groupings of elements and attributes as the top-level sections of the specification. These are *Animation*, *Metainformation*, *Content Control*, *Structure*, *Layout*, *Timing and Synchronization*, *Linking*, *Time Manipulations*, *Media Objects*, and *Transition Effects*. Of

2 SMIL 1.0 was not simply an XML-compliant language, it was the *first* XML language released by the W3C when it was published in 1998.

these, *Timing and Synchronization* is the core of the SMIL specification. The number of modules per functional grouping varies from 2 to about 20. In general, the more modules per group, the finer the granularity of each module. The structure of this book corresponds to the functional grouping of SMIL. We will consider all of the elements and attributes of each functional group in detail in the relevant chapters.

For a designer using SMIL, the notions of modules and functional groups are not very important. What is important is the result of the grouping of modules into a number of SMIL *profiles*. Each profile provides a fixed collection of elements and attributes, drawn from one or more modules from various functional groups. The purpose of the profile model is to enable customizing the integration of SMIL's functionality into a wide variety of XML-based languages without requiring language authors to learn new timing semantics for each variant.

The major SMIL profiles are:

- SMIL *2.0 Language:* the host-language version of SMIL 2.0, containing the majority of SMIL's elements, attributes and features;
- SMIL *2.0 Basic*: SMIL 2.0 baseline subset for low-power devices with a minimal set of elements and attributes;
- *3GPP Mobile* SMIL: SMIL profile defined for mobile telephones that contains most of SMIL's features except animation and some interactivity;
- *XHTML+SMIL*: SMIL timing elements integrated into XHTML; and
- SMIL *1.0*: the original version of SMIL.

SMIL functionality is also integrated into a number of other languages. Although these languages are not SMIL profiles, they do form important examples of how SMIL can be integrated for use in other domains:

- SVG — graphics animated with SMIL timing
- *XMT* — SMIL subset for use with MPEG-4 multimedia streaming

The scope of each of these profiles and applications of SMIL are briefly discussed in the following sections.

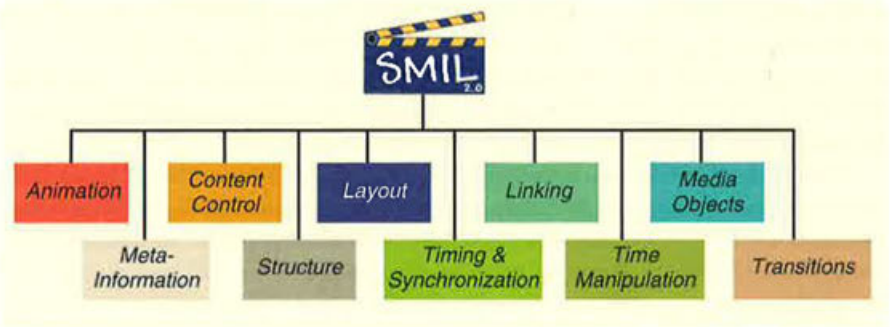


Figure 1-7. SMIL module architecture.



1.4.1 SMIL 2.0 Language Profile

The SMIL 2.0 Language profile includes essentially the entire set of elements and attributes defined in the SMIL 2.0 specification. Presentations encoded for this profile use SMIL as the document *host language*. (That is, the outer-most structure container is `<smil>`, not `<html>` or `<svg>`.) The SMIL Language profile exposes all of SMIL's functionality directly, using the element and attribute names defined in the specification. Many of the examples in this book are given using the SMIL 2.0 Language profile.

Oratrix's GRiNS/SMIL 2.0 player was the first player to support the SMIL 2.0 Language profile. RealNetworks began their support for SMIL 2.0 with the RealOne player. RealOne is the successor to the RealPlayer-6/7/8 series, and enjoys wide acceptance in the media publishing and media user communities.

Both the RealOne and GRiNS players required major engineering efforts to support the entire SMIL specification. This effort has proven to be a barrier for other groups hoping to implement SMIL. In order to stimulate the further acceptance of the SMIL 2.0 Language profile, both the RealOne player and the GRiNS player are being released in open-source versions. The open-source RealOne player is available as part of Real's Helix architecture; the code base and any extensions will become part of RealNetworks' intellectual property. The Ambulant player is an open-source implementation of SMIL 2.0 being developed (in part) by the team that built the GRiNS player. Ambulant is available under a GNU GPL open-source licensing agreement.

1.4.2 SMIL 2.0 Basic

The effort required to develop a complete SMIL 2.0 Language profile implementation is justified for desktop and server systems that need to support a wide range of media presentations, but the functionality of SMIL Profile is often considered to be beyond the needs of applications for low-power devices (such as PDAs and telephones). In order to manage the process of sub-setting SMIL functionality for these devices, the W3C developed the SMIL *2.0 Scalability Framework*. The SMIL 2.0 Basic profile defines the least complex subset of elements and attributes that still remains sufficiently SMIL-like to justify the use of a SMIL host language tag. As long as a SMIL subset supports at least this level of functionality, it can be considered to be part of the SMIL family. A key requirement of the scalability framework is that all presentations created for the sub-setted SMIL player should also present correctly in other SMIL players, including those that use the full SMIL 2.0 Language profile.

SMIL Basic is used in various SMIL players. X-Smiles (from the Helsinki University of Technology) is a Java implementation of parts of SMIL for integration into a browser environment. The INRIA PocketPC SMIL player is also based on SMIL Basic.

1.4.3 3GPP/PSS SMIL



The main standardization body for content on third-generation mobile telephones is 3GPP: the *Third Generation Partnership Project* consortium. This industry forum of device manufacturers selected SMIL as the basis for putting images, graphics, text and video on a wide range of mobile devices. The first versions of these specifications (which were called PSS for *Packet Switched Services*) were relatively bare-bones SMIL, but starting with version 5, a reasonable subset of the full SMIL 2.0 Language profile is offered that allow essentially all of the core functionality in Part 2 of this book to be supported

Various organizations, including CWI, InterObjects, Oratrix, PacketVideo and RealNetworks, have announced support for 3GPP SMIL players on various mobile telephone and PDA devices. Many other vendors hoping to cash-in on expected growth in the mobile market are also developing 3GPP SMIL players. This is an area in which significant growth can be expected.

In early 2001, the MMS-2.0 multimedia messaging service protocol was announced by a consortium of industry partners. MMS-2.0 is intended for mobile image slide shows. It is SMIL-like, but does not contain substantial SMIL functionality. Unlike SMIL Basic documents, MMS-2.0 presentations cannot play on general SMIL players.

Currently, the name MMS is used in the mobile telephone world to mean a particular standard for encoding messages (this is MMS-2.0), but also to mean the entire suite of future multimedia messaging (including the 3GPP/SMIL profile and its future extensions). To avoid confusion, we will use the term MMS to only mean the MMS-2.0 sub-SMIL language. 3GPP SMIL will be used for more generic multimedia messaging services for mobile devices.

1.4.4 XHTML+SMIL Profile

The XHTML+SMIL profile is an example of a profile that does not use SMIL as the host language. Instead, XHTML+SMIL integrates SMIL constructs directly with HTML constructs into a single XML format. XHTML+SMIL documents allow a high degree of local synchronization of text fragments and presentation components inside of an HTML base. Since, however, XHTML has no inherent notion of time, XHTML+SMIL documents usually have only limited temporal scope. The playing of XHTML+SMIL is best suited for multimedia presentations with heavy use of text within one predominantly textflow-based layout, and less dependence on other more time-intensive media. The XHTML+SMIL profile includes most of the timing, animation, media and content control facilities defined in the SMIL 2.0 specification. XHTML+SMIL does not use any SMIL layout constructs — it uses HTML and CSS constructs instead.

Even though XHTML+SMIL has not progressed to full W3C Recommendation status because only one implementation has been provided to W3C for testing, it enjoys very wide distribution: Microsoft's implementation of XHTML+SMIL is called HTML+TIME and has been available in *Internet Explorer* since version 5.5.



1.4.5 SMIL 1.0

SMIL 1.0 was the first version of SMIL. It is not technically a SMIL 2.0 profile, but since many SMIL 2.0 players can open and execute a SMIL 1.0 file, we include it in our discussion. SMIL 1.0 defines the core functionality supported in all of the functional groups of SMIL 2.0, but none of the SMIL 2.0 extensions. In nearly all functional groups, the "basic" module (such as the *BasicLinking* or *BasicLayout* module) has functionality that corresponds to the original behavior of SMIL 1.0. One of the major differences between SMIL 1.0 and SMIL 2.0 is that the names of many elements and attributes were changed from a hyphenated form (such as `background-color`) to a camel-case form (such as `backgroundColor`). Many current SMIL Language profile players support both forms, but mobile players usually do not.

SMIL 1.0 is supported by a wide range of media players. The first public implementation of SMIL 1.0 was in the CWI/Oratrix GRiNS/SMIL 1.0 player. By far the most important player (in terms of installed base) to support the profile was the RealNetworks RealPlayer (versions 6, 7 and 8). SMIL 1.0 is also supported by Apple's *QuickTime* (starting with version 4.1). There are also a variety of other SMIL 1.0 players, developed mostly by research labs and universities; these include *Soja (by Helio)*, *Schmunzel* (Salzburg Research and Sun Microsystems), *S2M2* (NIST: the U.S. National Institute of Standards and Technology) and *HPAS* (DEC/ Compaq/HP).

1.4.6 SVG

SMIL 2.0 constructs have also been integrated into *SVG (Scalable Vector Graphics)*, the W3C's Recommendation for vector graphics. These constructs from SMIL enable SVG constructs to be animated. This animation of XML-defined constructs by SMIL 2.0 is defined in the W3C Recommendation SMIL *Animation*.

Several players provide support for SMIL animation and SVG. Of these, the largest installed base belongs to Adobe's SVG *Viewer*. The Corel SVG player has limited support for SMIL animation.

1.4.7 XMT

The *eXtensible MPEG-4 Textual Format (XMT)* is the official XML encoding of MPEG-4 presentations. XMT represents higher-level author intent from which final-form (that is, adapted to a particular use and environment) MPEG-4 presentations can be generated. XMT can be parsed to SMIL for playback on SMIL browsers, or processed into the binary MPEG-4 final form format for playback on MPEG-4 browsers. IBM has made a converter from XMT to MPEG-4 as a research tool, but it is not yet publicly available.

1.5 Creating SMIL Presentations



This book is about the structure of SMIL 2.0 and SMIL's elements and attributes. There are several ways that you can use this information to create SMIL presentations. This section provides a brief sampling of editing alternatives in order of increasing functionality. It does not pretend to be complete.

> **Quick Tip**
>
> An up-to-date list of SMIL editors is at W3C's main SMIL page: `http://www.w3.org/AudioVideo/`.

1.5.1 Notepad (and Other Text Editors)

Clearly, the most popular SMIL editor to date has been a simple text editor. SMIL's XML format and its regular structure allow a text editor to be a convenient base for making simple presentations. The advantages of using a text editor include:

- *price*: a text editor is usually free;
- *ease of use*: if you can type, you can type SMIL; and
- *full SMIL 2.0 Language profile support*: you have access to all of SMIL elements and attributes when you use a text editor.

The disadvantages of using a text editor for creating SMIL include:

- *you are on your own*: a text editor doesn't help spot language errors or simplify data entry;
- *you get no performance feedback*: typing a SMIL file that presents six videos in parallel on a mobile phone is easy — but what you really want is an editor that says: *you're crazy to try this!*; and
- *multiple versions of a document require multiple versions of the text*: if you want to create for multiple devices, make sure your RSI insurance is paid up!

While full-feature SMIL editors are good for serious development, a text editor lets you get started with creating SMIL immediately.

1.5.2 SMIL Syntax Validator

CWI, the Dutch national center for mathematics and computer science in Amsterdam provides the SMIL *Syntax Validator*. For those that write their SMIL code directly with text editors, the validation of this code with the DTDs (Document Type Definitions) of SMIL family formats is essential. Tools like this help authors make sure the code they write conforms to SMIL. This validator checks code written for SMIL 1.0 and the SMIL 2.0 Language profile.



1.5.3 Perly SMIL

Another tool for the hand-coding inclined is the *Perly* SMIL Perl module. This is a module of functions in the programming language Perl. With Perly SMIL, Perl hackers can more quickly write programs that generate SMIL output. Perly SMIL can be downloaded free-of-charge.

1.5.4 SMILGen

An interesting addition to the SMIL GUI editing scene is SMIL*Gen*, a RealNetworks product developed and distributed as free-of-charge and open source. It provides an XML-oriented GUI, facilitating creation of XML constructs in general, accounting for the possibilities and restrictions of individual DTDs and schemas, and with some SMIL 1.0- and 2.0-specific features built directly in. It is more general purpose than template-based SMIL 1.0 editors, but requires more XML and SMIL expertise. It doesn't have the intricate SMIL 2.0-specific interface that GR*i*NS has — but then, SMILGen is free-of-charge.

1.5.5 MAGpie

American public television's National Center for Accessible Media (NCAM) has developed *MAGpie* for the creation of closed-captions and audio descriptions. MAGpie makes it easier to get the timing information from the media to which captions and descriptions are synchronized. MAGpie generates captions and descriptions in SMIL 2.0 code. It comes with an embedded GR*i*NS previewer.

1.5.6 GR*i*NS

The industrial-strength power tool for SMIL 2.0 authoring is GR*i*NS, by Oratrix. GR*i*NS provides graphic user interfaces for viewing and editing the spatial and often intricate temporal structure of SMIL 2.0 presentations. The authoring environment also provides integration and playback of individual media components and portions of SMIL presentations. GR*i*NS gives performance feedback (providing network bandwidth analysis for multiple connection speeds and user profiles) and it exposes all of the SMIL language. Like its player counterpart, GR*i*NS is a complete SMIL 2.0 implementation. It can output code for the SMIL 2.0 Language profile, the 3GPP Mobile profile and XHTML+SMIL. Like most power tools, GR*i*NS is useful for the more expert authors of larger presentations, but it can have a higher learning curve than simple text editors. The GR*i*NS editor can be downloaded with a several week free-of-charge trial period. After the trial expires, a license needs to be purchased.

1.6 SMIL and Other Specifications

The potential of multimedia on the Web is tremendous. Even the most conservative commentators realize that the introduction of audio/video assets as

first-class Web objects can have a major impact on the way that people will consume information and entertainment in the future. Not surprisingly, SMIL is not alone in the multimedia standardization space. Often, multimedia standards are characterized by their content, rather than their composition. In this sense, there has been some confusion as to how SMIL relates to these other formats.



1.6.1 CMIF

In 1991, CWI in the Netherlands began a research project on network hypermedia. Late that year, CMIF (the CWI Multimedia Interchange Format) for synchronized multimedia was published that contained the seminal elements of what would later become SMIL. CMIF supplied the general timing semantics, structure, content control, layout, linking and interaction facilities that would be used in the SMIL language. (Some of these facilities made it into SMIL 1.0, others in SMIL 2.0.) Many of CMIF facilities were 'enhanced' by the committee-driven standardization process. (Some of these enhancements were also improvements!) The CMIF player/editor later formed the basis for the Oratrix GRiNS environment.

1.6.2 Dynamic HTML

Dynamic HTML (D-HTML) was introduced as a way of introducing local time and animation effects into a static HTML Web page. While some of the animation primitives in SMIL 2.0 resemble the functionality of some uses of D-HTML, SMIL's scope is much broader than the local nature of D-HTML.

1.6.3 Flash

Macromedia's *Flash* is a proprietary content media type that is primarily used for small animations. In contrast, SMIL 2.0 is not a content media type, in the sense that it does not define any particular type of media, such as vector or raster images, videos, text or audio data. Instead of media content, SMIL describes media *composition*. A SMIL presentation can include Flash objects.

SMIL has a number of advantages over proprietary formats such as Flash. One is that it is fully XML-based — it is able to describe adaptive content, and a single file can be targeted to several different target platforms, such as broadband, Web and mobile. The format is also truly open — no one company controls its destiny.

1.6.4 MPEG-4

MPEG-4 is a family of protocols that cover a wide range of media-related concerns, but not a specific solution to any one class of media presentation. It is often stated that the biggest competitor for SMIL 2.0 is MPEG-4. While it is tempting to be drawn into a feature-by-feature comparison, most of SMIL 2.0 and MPEG-4 are actually very complementary.



The major differences between MPEG-4 and SMIL lie in the approach to document structuring and in the separation between structure and content. MPEG-4 is essentially a final-form description of a presentation. It is saved in a binary format, contains extensive control information, and directly contains all of the media content of the presentation. A SMIL document provides a specification of the high-level and detailed synchronization, layout and content control requirements, but it contains no data — the actual media items are included by reference. As with Flash, the combination of content and control can provide a more precise implementation of a presentation, but the separation of content and control can provide a more flexible and reusable presentation architecture.

Both SMIL and MPEG-4 use a profile-based architecture. The profile architecture of MPEG-4 is significantly more complex than SMIL's, resulting in a wide variability in the components that are actually implemented in any particular MPEG-4 player. MPEG-4 also carries with it substantial licensing issues, as much of its component standards contain intellectual property that must be licensed from the IP owners. SMIL 2.0, on the other hand, has been implemented under a royalty-free agreement with its developers.

During the development of SMIL 2.0, close cooperation was maintained with a portion of the MPEG-4 community. Since MPEG-4 is a binary-encoded standard, it was felt that SMIL 2.0 could be used as the text-format for MPEG-4 presentations. The result is the coordinated development of SMIL 2.0 and MPEG-4 through the *XMT (eXtensible MPEG-4 Textual Format)* specification. Here, MPEG-4 encodes low-level objects, while SMIL 2.0 encodes high-level, XML-based object composition.

1.7 Summary and Conclusion

SMIL 2.0's release makes full-fledged multimedia accessible to every user, every author, every implementer and to the infrastructure of the Web itself. It is emerging as a major force in making the existing Web and its various interfaces more animated, engaging, accessible and informative.

SMIL 2.0 is widely supported and distributed. The full-powered SMIL 2.0 Language profile is supported in the Web multimedia mainstay RealOne and the open-source Ambulant players. The 3GPP SMIL profile is supported on a wide range of telephone and PDA devices. XHTML+SMIL is supported in Internet Explorer, the world's most widely distributed network browser. SMIL timing with graphics in SVG is available in the Adobe and Corel SVG players. Large-scale SMIL 2.0 presentation design and maintenance is available through GRiNS, while several template-based editing systems are available for making smaller presentations.

The purpose of this chapter has been to introduce you to the SMIL environment. All of the presentations discussed in this chapter are available on the book's Web site. This site also contains a list of links to SMIL-related products and services. We suggest that you try to experiment with our demonstration

examples and that you then move on to our next chapter, which looks at the structure of the SMIL format.



1.8 Further Resources

W3C Audio/Video Pages

The W3C keeps an up-to-date SMIL Web site with information on players, editors and tutorials at: `http://www.w3.org/AudioVideo/`.

SMIL Players and Editors

Our book's Web site has pointers to various SMIL tools and a downloadable version of the Ambulant SMIL 2.0 player. See:
`http://www.XmediaSMIL.net/` .

Vendor SMIL Pages

Many SMIL player vendors have pages that contain SMIL documentation, including example documents and tutorials.

3

Local and Streaming Media



SMIL allows you to take one or more media objects and integrate them into a single multimedia presentation. From SMIL's perspective, the location of the media — whether local or spread across the Web — is not particularly important. From a designer's perspective, however, there is a big difference between information that is stored locally and information that needs to be fetched from half-way across the network, especially when multiple media streams need to be synchronized.

This chapter gives background information on local and streaming media issues. It will help you understand the general organization issues that impact all media presentations. We begin with a discussion of media types and media formats. We then discuss conventions for referencing local and remote media. We conclude with a discussion of synchronization and streaming media issues.

Very little in this chapter involves the particular elements and attributes of SMIL. (These are the subject of the rest of this book.) Instead, we focus on understanding the basic — and potential problems and pitfalls — of media organization that will help you make more effective presentations.

3.1 Classification of Media

Most viewers of multimedia presentations would probably say that there are two kinds of multimedia media data: *fast* and *slow*. (This is a tremendous improvement over what users would have said a few years ago: *slow* and *loading*.) Presentations with "fast" assets load quickly and run seamlessly. Presentations with "slow" assets take forever to load and then rarely run without stopping or pausing. Unfortunately — or, perhaps fortunately — not all Web users experience the same network performance. Depending on the connections available, the same piece of media may load very quickly or very slowly.

Media engineers typically take a more technical approach to classifying media. Within this world, there are two media classes: *discrete* and *continuous*.

- *Discrete media*: media in which there is no inherent duration associated with presenting the media. Discrete media includes images, text and most (structured) graphics. Discrete media can be very large (such as a 1600x1024x32 image of a landscape) or it can be very small (such as a one-bit GIF image).

- *Continuous media*: media in which time plays an essential role. Continuous media includes audio, video and some animated graphics. A piece of continuous media usually consists of an ordered set of discrete media components, each of which is usually called a frame or a sample. Continuous media objects can be very large (such as 4 minutes of 1600x1024x32 video) or it can be relatively small (such as a 4 second sound file that is encoded at a very low bitrate).



Table 3-1 provides an overview of the sizes of various discrete and continuous media objects. Some media are hybrid types: an SVG (or Flash) animation consists of a timed sequence of discrete images. Since animations are typically fetched as a single block, they are usually treated as discrete media.

Media Type	Media Class	Size (Kbytes)
Text (1 pg, raw)	Discrete	5.1
Text (1 pg, formatted)	Discrete	51.2
Color image (256x256)	Discrete	65.5
Color image (full screen)	Discrete	3,100.1
Audio sequence (10 seconds)	Continuous	160.0
Audio sequence (1 minute)	Continuous	10,500.2
Video sequence (short)	Continuous	70,000.4
Video sequence (long)	Continuous	151,500.3

Table 3-1. Typical media object types, classes and sizes.

For a media perspective, there is no direct relationship between a media type (continuous or discrete) and a user's performance experience. It is tempting to think of continuous media as always presenting more resource use problems within a multimedia presentation than discrete media, but this is not necessarily true: discrete media typically needs to be presented all at once, while continuous media can be fetched and presented in semi-autonomous blocks. A large image may be "slower" than the same-sized streamed video object. Continuous media does present one problem that is not encountered with discrete media: continuous media requires strict inter-frame synchronization to ensure that media playback is "correct". (A video or audio object will lose its semantic meaning if information is not rendered at a constant rate.)

3.2 Media Formats and Encodings

One of the common frustrations for many new users of SMIL is that the language does not contain a default set of media formats that are supported by all SMIL players and browsers. There are several reasons why SMIL was forced to be media-agnostic: first (and foremost), at the time SMIL was specified, there was no set of common continuous media formats that were available on all

platforms without some form of server or licensing restrictions. The W3C did not want to force organizations implementing SMIL to adopt proprietary media formats, especially if these were not freely available. Of course, the fact that several companies on the SMIL standardization committee each had commercial interests that precluded the definition of a common set of media types didn't help, either! (The problems of standardizing on media were also true for SVG and HTML, but since these formats make little or no use of continuous media, the issue was not as obvious as with SMIL.)

The media-agnostic nature of SMIL does not mean that the selection of media formats is not a key media concern. The main reason that media formats are important is that, in raw form, both discrete and continuous media are typically too large to efficiently retrieve, transfer and process across a digital network. Even a relatively small raw image of 320x240x8 bits can result in significant transfer times across low- and moderate-speed networks. They also require lots of storage space on network servers and clients. As a result, various forms of compression have been developed for different media types. The following sub-sections consider compressed media formats for both discrete and continuous media.

3.2.1 Media Formats for Compressed Discrete Media

The typical types of discrete media found in multimedia presentations are photographic images, computer-generated (artificial) images, animated images and text. While many formats for each of these media objects exist, there are several formats that have taken on the role of *de facto* standards that are supported by most SMIL players and browsers. The choices are summarized in Table 3-2.

SMIL Profile	Player/ Browser	Images			Text
		Photo	Artificial/ Bitmap	Vector/ Animation	
SMIL Language	RealOne	JPEG	PNG, GIF	Flash, SVG	RealText, Plain
	Ambulant	JPEG	PNG	SVG	Plain
	GRiNS	JPEG	PNG	Flash, SVG	Plain
SMIL Mobile	MMS-2	JPEG	GIF	---	Plain
	3GPP	JPEG	PNG, GIF	SVG-Tiny	Plain
SMIL 1.0	QuickTime	JPEG	PNG, GIF	Flash	QuickText
XHTML +SMIL	IE-6 (HTML+Time)	JPEG	PNG, GIF	Flash, SVG	Plain, HTML

Table 3-2. Preferred discrete media, per profile and player.

Photographic Images

JPEG (a compression standard developed by the Joint Pictures Expert Group) is currently the most widely used image format across the Web for photographic images. It has excellent compression characteristics that can be tailored to meet the needs of the image content. While compression ratios of between 10:1 to 50:1 are common with JPEG, this compression comes at a price: JPEG has a lossy compression algorithm, which means that at higher compression rates, the resulting image can be considerably less crisp and clean than the original. In practice, for Web uses this is rarely a problem unless the compressed image is resized (made larger) as part of a SMIL scaling operation. Since nearly all image processing packages — and all digital cameras — support JPEG, it should be the primary choice for photographic image media.



Another popular format for images is TIFF (the Tagged Image File Format). TIFF images have the advantage that they provide a loss-less encoding of the original image (which is useful if the image is to be resized), but even compressed TIFF files often are too large for use in Web-based multimedia.

Artificial Images

Where JPEG is useful for photographs, it is often less optimal for computer-generated bitmaps that contain simple geometries and large fields of solid colors. (Screen-dumps or images generated from raster data are typical examples.) For these images, the PNG (Portable Network Graphics) format is usually a better choice. PNG also supports complete and partial transparency — allowing an image to be overlaid or blended on top of another media object. PNG is an open, royalty-free format that is supported by nearly all SMIL renderers and image manipulation packages.

Another standard for bitmap images is GIF (the Graphic Image Format). GIF files are well-known and widely used, but unless you need the animation facilities in GIF that allow sequences of sub-images, it is not a particularly good format for future use. GIF uses licensed technology which may impede its implementation in some SMIL renderers.

Vector-Based Animated Images

Unlike bitmaps, which contain all of the bits rendered in the image, a vector format describes shapes based on end-points, line segments, line thickness and fill colors. As a result, vector files describe how an image is to be generated rather than containing a painted version of the image itself. This often makes the size of vector images small — and, more importantly, it provides an ability to resize the image without quality loss.

There are two formats in wide use that support vector-based images: SVG and Flash. SVG is an open standard from W3C (the same people who brought you HTML and SMIL), while Flash is a closed standard that is controlled by Macromedia. Both SVG and Flash are usually supported in SMIL players via

plug-in renderers. Many small devices are committed to supporting the SVG-Tiny profile.

A choice between SVG and Flash will most often rest on philosophical rather than technical grounds. While Flash encodings can be more compact than SVG — this because Flash uses a binary rather than text encoding format — some SMIL players/browsers provide native implementations of SVG, which means that parts of the timing of the SVG media can be controlled using SMIL's constructs. The ultimate choice between the formats will often rest on the availability of plug-in renderers for each format within a SMIL player/browser.



An alternative to both SVG and Flash is MPEG-4. MPEG-4 is a complete media encoding and scheduling solution that contains aspects of functionality found in both SMIL and SVG/Flash. Unlike SMIL, which uses a user-editable open format, MPEG-4 packages all media content in a closed, final-form format. In practice, mixing the use of SMIL and MPEG-4 is probably not very useful, although there are advantages to transforming SMIL into MPEG-4 and vice versa.

Plain and Formatted Text

In theory, text would seem like a very portable, ubiquitous media type. In practice, rendering formatted text in many media players is a difficult and resource consuming task that is usually minimally supported. The main reason for this is that the support for scalable fonts and text-based re-flow of content based on dynamic margins is very resource intensive. For all practical purposes, formatted text data is usually encoded as an image in PNG or GIF format. Small amount of incidental text can be encoded directly in the player's base font, but this is the exception rather than the rule on other than very small devices.

Alternatives for formatting text are the use of SVG or XHTML. Both of these solutions require activating external renderers. For this reason, unless lots of text is presented in the presentation, image encoding of text is usually used in the SMIL Language and Mobile profiles. Implementations of the XHTML+SMIL profile are much more suited to including large amounts of text. Note that the use of text for captions is typically done by treating text as a continuous media object. See *Timed Text* on page 61.

3.2.2 Media Formats for Compressed Continuous Media

The typical types of continuous media found in multimedia presentations are video clips, audio files and timed text. There are several formats available for each of these media types, but unlike discrete media, there has been little standardization in the use of continuous media. The principal barriers to standardization remain technology licensing issues and format compatibility with proprietary streaming media servers.

Apart from the issue of standardization, the primary distinguishing factor for the processing of continuous media is the transfer model used by the format. As we discuss more fully in *Accessing Media Within a SMIL Presentation*

on page 65, two alternatives exist: a *download-and-play* model and a *streaming* model. While most streaming media servers can provide some measure of streaming support for nearly all media types, these servers often work best when they stream the — often proprietary — formats that were specially developed for the server.

In the following discussion, we consider both download-and-play and streaming media formats.



Audio

There are dozens of download-and-play audio formats but only relatively few streaming audio formats that can be used in a wide range of media players. The audio format selected will be driven by the type of platform being used and the type of audio being encoded. Table 3-3 lists the major formats supported by platform and player at the time this book was written. Note that many mobile players will support proprietary media formats (such as RealAudio or Windows Media) if the service provider has made special technology licensing deals with either RealNetworks or Microsoft.

SMIL Profile	Player/ Browser	Speech Audio		Music Audio	
		Download-and-Play	Streamed	Download-and-Play	Streamed
SMIL Language	RealOne	MP3	RealAudio	MP3	RealAudio, MP3
	Ambulant	MP3	Ogg Vorbis	MP3	Ogg Vorbis
	GRiNS	MP3	.rm, .wmf	MP3	.rm, wmf
SMIL Mobile	MMS-2	AMR	---	MIDI	---
	3GPP	AMR	--	MIDI	Impl. Dependent
SMIL 1.0	QuickTime	AIFF	AIFF	MP3	MP3
XHTML +SMIL	IE-6 (HTML+Time)	WAV	WM	MP3	WM

Table 3-3. Preferred audio media, per profile and player.

Video

As with audio, video can be either downloaded in its entirety before use or it can be streamed to a player/browser. While many video formats exist, the most common formats across the Web are: MPEG-1/2/4, AVI and RealVideo. Table 3-4 provides an overview of the primary and secondary formats available on major platforms as of the writing of this book. Note that MPEG-based formats can use a variety of codes and support a range of bitrates. Not all players may support all variants.

Timed Text

As mentioned above, most incidental text in multimedia presentations (such as headings and titles) will be formatted as bitmap images. An exception to this is the use of timed text for captions. Captioning has often been embedded within video media objects (so-called closed captions), but SMIL allows multiple caption objects to be specified within a media file as peer-level media objects. The advantage of this approach is that captions can be supported in multiple languages or at multiple levels of detail, and that the end-user can choose whether — and which — captions are visible.



Most of the solutions supported for captioning of content have been proprietary. Of these, *RealText* (developed by RealNetworks), *QuickText* (developed by Apple Computer) and *SAMI* (supported by Microsoft) are the most widely used. In mid-2002, the W3C started an activity on standardizing on a common timed-text format based on general SMIL timing constructs.

Table 3-4 provides a summary of timed text options available under various platforms and profiles. For those organizations participating in the W3C Timed Text activity, this format has been listed as a secondary format under the name TTF.

SMIL Profile	Player/ Browser	Video		Timed Text
		Download-and-Play	Streamed	
SMIL Language	RealOne	MPEG-2	RealVideo	RealText, TTF
	Ambulant	Div-X	Div-X	TTF
	GRiNS	MP4	.rm	.rt
SMIL Mobile	MMS-2	---	---	---
	3GPP	H.263	Impl. Dependent	Impl. Dependent
SMIL 1.0	QuickTime	MPEG-2	MPEG-4	QuickText
XHTML +SMIL	IE-6 (HTML+Time)	AVI	MPEG-4	SAMI

Table 3-4. Preferred video and timed text media, per profile and player.

3.2.3 Tools for Creating Media

The most time consuming part of making multimedia presentations is usually creating the media objects, not the SMIL file. While a detailed consideration of media creation is beyond the scope of this book, Table 3-5 lists a number of tools and software packages that can be used to create and edit media.

Media	Type	Formats	Software Tools	Hardware Tools
Images	Photo	JPEG, TIFF	Adobe Photoshop, miscl image editors	Dig. Cameras, Scanners
	Artificial	PNG, GIF		Scanners, tablets
	Vector	SVG Flash	Adobe Illustrator, Corel Draw Macromedia Flash	--
Text	Plain	txt (X)HTML	text editor Dreamweaver, Go Live!	--
	Timed	RealText, SAMI	Magpie	proprietary
Audio	Download	mp3, wav, aiff	SoundForge, MediaCleaner	Microphone, mixer panel, equalizers
	Streaming	RealAudio		
Video	Download	MPEG, AVI	Adobe Premiere, MediaCleaner	Dig. Video Camera Firewire
	Streaming	RealVideo WMV		

Table 3-5. Representative media creation tools.

3.3 URIs: Locating and Naming Media Objects

Once media objects have been created, they need to be named and placed at a location somewhere in the distribution environment. The *URI (Uniform Resource Identifier)* is a standardized syntax for identifying and locating such resource. SMIL, and other W3C Recommendations such as HTML and SVG, often use URIs as values for attributes that locate information, typically for presentation to the user. Figure 3-1 shows the basic components of a URI. The three core components of an Internet URI are the *scheme*, which states the protocol to use for transferring the media object, the *server*, which gives the name of the computer holding the media, and the *path*, which says where on the server the media is stored.

In this section, we go over three types of URIs: URIs to files stored on the local computer, URIs to resources stored across the Internet and URIs that are relative to a shared base location.

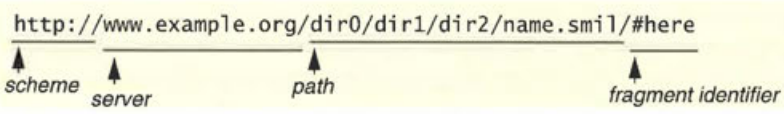


Figure 3-1. The basic components of a URI

3.3.1 Local URIs

The simplest type of URI specifies a file on the file system local to the browser. Such a local system path-name has the URI scheme prefix "`file://`", followed by a third slash and the path (without a server) to ensure that URI processors recognize it as a local path-name. (Note that the server name is included between the second and third slashes, but because this name is blank, it takes up no space!) If desired, a name of `localhost` may be used.

Presentations that use local file URIs are not portable — they only work on computers with the given files. This is fine for local presentations not using the Web, such as overhead presentations for groups or presentations distributed on CD-ROMs. However, SMIL presentations posted on the Web for playback at any location should not use local file URIs.

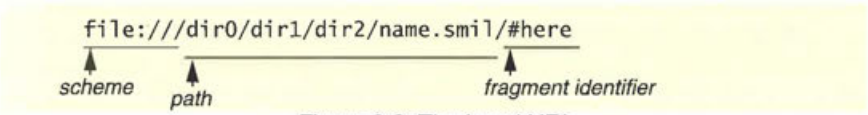


Figure 3-2. The Local URI

3.3.2 Remote URIs

Internet URIs can locate any object on the Web from anywhere on the Web. They are crucial for presentations that can play from any connected computer. In the bullets below, we discuss how URI schemes, servers and paths are used with remote URIs:

- *URI schemes*: The scheme of a URI states what protocol is being used to transfer the file or resource from the server to the browser. The server computer needs to know not just what to send, but how to send it, and the scheme tells the server how. The most commonly used transfer protocol on the Web is HTTP (Hypertext Transfer Protocol); this is a download-and-play protocol. This protocol goes with the URI prefix "`http://`".

  As a multimedia format, SMIL often incorporates streaming media, for which a streaming protocol is required. The streaming protocol most often used is RTSP (Real Time Streaming Protocol); this protocol allows incremental downloading and playback of media. This protocol goes with the URI prefix "`rtsp://`".
- *URI servers*: The second component of a typical URI is the server name. This is the name of the computer on the Internet that holds the media being retrieved. It tells the browser which computer on the Internet it must communicate with to get the media item.
- *URI paths*: The server uses the path to determine what from its system to send. This path is typically a path-name for a directory or file on the server's local file system. It can, however, be instead another type of string that the server can process to determine a media item to return.

- *URI fragment identifiers*: Fragment identifiers are placed after the '#' characters in the path name; they locate portions, or fragments, of files. Their typical use in SMIL is as named components of XML files, particularly HTML, SMIL and SVG. The effect of loading a named fragment of any of these files for display is that the whole presentation is loaded and its presentation state is "forwarded" to the located fragment. (How this "forwarding" occurs depends on the format). Fragment identifiers are not part of the URI.

Some players allow URIs to be extended with embedded parameters. SMIL has a separate mechanism to support such parameter passing, as described on page 65.



3.3.3 Relative URIs

The URIs presented up to now have been *absolute URIs*. An absolute URI by itself contains all the information needed to locate its resource from anywhere on the Web. *Relative URIs* are incomplete addresses that are applied to a base URI to complete the location. The portions omitted are the URI scheme and server, and potentially part of or all of the URI path as well.

The primary benefit of using relative URIs is that documents and the directories containing them can be moved or copied to other locations without requiring changing the URI attribute values within the documents — easing Web site maintenance. (However, these particular benefits assume the default behavior of relative URI paths.)

There are three types of relative URI specifications in SMIL: *internal URI references*, *relative paths* and `xml:base` *paths*.

- *Internal URI references:* URIs that begin with '#' characters are internal URI references. Such a reference locates some portion of the current document, unless overridden by an `xml:base` attribute assignment.
- *Relative URI paths:* Relative URI paths are typically used as a short means of locating media files stored in the same directory as the current document, or in a directory close to it. They often consist of just the filename (optionally with a fragment identifier into that file). They can also have a relative directory path before the filename.
- `xml:base` *paths:* By default, the base URI for a relative URI is the location of the current document. The `xml:base` attribute overrides this default: it explicitly sets a base location from which relative paths can be constructed. Use of this attribute enables, for example, the media for a presentation to be loaded from a media repository somewhere else on the Web than the SMIL document's site. Thus, no matter where on the Web the SMIL presentation is copied or moved to, it will get its media content from the same repository. Furthermore, if the media repository is moved, then only changing the single `xml:base` assignment is necessary to update the referencing of multiple `src` and `href` attributes.

```
hello.txt
file:///hello.txt
http://www.example.org/hello.txt
rtsp://www.example.org/hello.rm
data:,Greetings!
```

Example 3-1. Valid URI constructs for locating and referencing media.

Example 3-1 provides examples of URI structures. The SMIL-related aspects of URI use are discussed in Chapter 5: *Referencing Media Objects*.



3.3.4 Passing Parameters in URIs

The purpose of a URI is to locate and identify a network resource, such as a media object. In practice, simply pointing to a media object is often not enough to activate an efficient transfer between a server and a client. Many network protocols allow extensions to the URI to support the passing of transfer control parameters. These often are structured as query strings (that is, they contain a '?' followed by a set of expressions such as: `lang=NL`). These strings are placed at the end of the location of the resource. SMIL defines the `<param>` element to provide a more structured means of passing parameters to media servers (or to other parts of a presentation). For details, see *<param>* on page 252.

3.4 Accessing Media Within a SMIL Presentation

The continuous and discrete media objects used in a SMIL presentation can be stored on the same computer as the SMIL player (this is called *local media*) or it can be stored on a server somewhere across the network (this is *remote media*). All media is accessed via a URI (a uniform resource identifier). Local media is usually accessed via the local file system. Remote media can be accessed via a network file system, or via a streaming server.

3.4.1 Local access to media

It is tempting to think of local media as being fast and free (in the resource-use sense of 'free'), but this is only the case for desktop systems with large local disks. For systems within limited local storage, media assets can also be placed on local removable storage devices (such as CD-ROMs), but doing so will result in media that takes longer to load and access than would be the case on a local disk.

On mobile devices, such as PDAs and telephones, local access is further limited by the fact that storage size is severely limited, and the fact that the storage space that is available is relatively slow. (Slower devices typically use less battery power.)

While most SMIL presentations will be constructed to use remote media, local storage can provide an attractive alternative in some situations:

- *CD-ROM distribution*: the oldest media presentations were CD-ROM based, but even modern presentations (such as games or encyclopedias) are often distributed via hard-copy media. This is done for both commercial reasons (to protect the contents) and because of performance constraints.
- *Pre-loaded distribution*: for some applications, such as music, movies or hourly sports/news summaries, an interesting distribution model is pre-loaded local storage. Here, the entire presentation (plus assets) are pre-loaded onto a portable device (such as a PDA or higher-end telephone) over a possibly long period of time — overnight, or while the device sits passively in a desktop cradle. The presentation can then be accessed locally by the user while away from the network connection — such as on a bicycle, in a car or on an airplane.



Although there are many uses for locally-created SMIL presentations, these will be the exception rather than the rule for most Web-based uses.

3.4.2 Remote Access to Media

SMIL was originally created to manage the presentation of media assets in a network environment. This involves integrating an architecture for standard resource naming (that is, providing a way to reference media objects) and defining a set of temporal and spatial rules for governing when and where the presentation client presents those objects. Given the diversity of Web information servers, client playback environment and the heterogeneity of the Internet itself, SMIL also gave early attention to facilities for customizing a presentation so that it can adapt to the resources and restrictions present when the SMIL file is activated.

Individual assets can be downloaded in their entirety during a presentation, or they can be fetched incrementally via a streaming server. Without streaming, remote access combines all the disadvantages of the local media (a lot of space is required to hold complete copies of media) with the disadvantages of the network (you need to wait a long time before all your media is present). For single media presentations — presentations that consist of a single audio or video file — some of the local storage and network delay problems can be minimized by using highly compressed formats, but this approach does not scale well if the presentation has lots of media components. It also isn't optimal for wireless and low-resource devices, because the transmission delay can still be substantial for whole media and the storage requirements too taxing for the device.

We consider various remote access issues in the following sections.

The Network Infrastructure

From the perspective of the user, there are two important aspects of the connection to 'the network': the speed of the connection itself and the connection model, or the method in which the user is attached to the infrastructure.

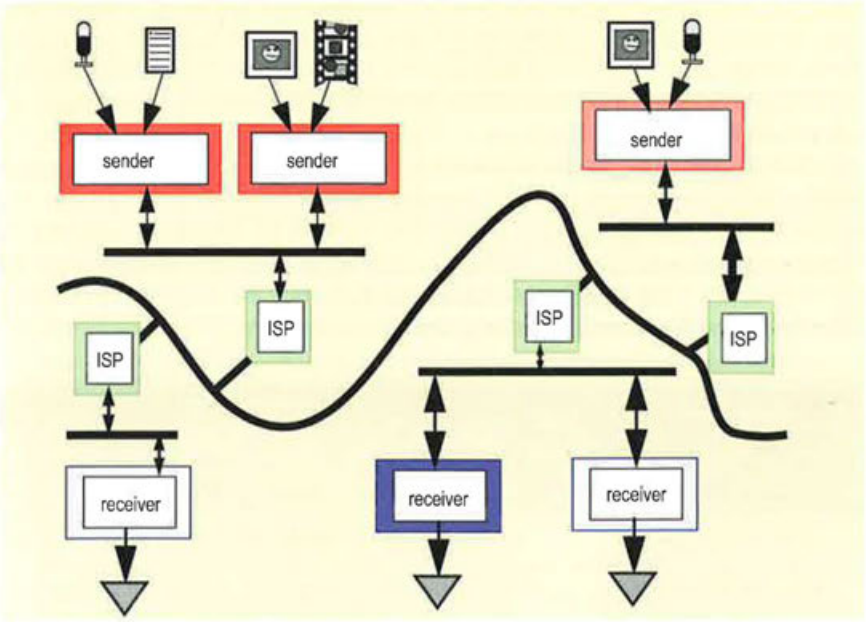


Figure 3-3. An overview of the network environment.
(The [Internet] backbone is not as homogeneous as shown.)

For remote multimedia, two connection models have dominated: one is where each client has direct internet connection (perhaps via a private local area network) and the other is the indirect connection to the network through a modem and a (wireless) telephone line. In both cases, a local internet service provider (ISP) connects the user to the network: this ISP can cache multimedia data — usually under contract with an information provider — or it can simply provide pass-through access to information that is transferred via a hierarchy of ISPs.

For most purposes, the connection through multiple levels of ISPs is transparent to the user, but it brings with it a performance penalty: the greater the number of intermediate ISPs, the greater the delay that is generated through the network. (For a typical connection from Amsterdam to Cambridge, England, it is not unusual to go through 15 levels of connections between the source and destination.) For non-trivial presentations, there may be several data paths through the network active concurrently. For example, a piece of video may come from a server in California, but the associated Dutch subtitles may come from a site in Amsterdam. The delay characteristics of each connection will likely vary considerably.

The transfer speed of any one connection is determined by the *available* bandwidth of each of the active network links and the processing power of both the source and destination to service any single request. For some trans-

fers, even a very fast connection will not improve the perceived performance from server to client *if* the server is not able to process all requests adequately. On the other hand, even a very fast server will not be able to provide adequate perceived response if the intermediate network link(s) are not free to carry the data involved.

A bandwidth comparison is shown in Table 3-6 along with the resulting transfer times for various types of complex data. It is important to realize that the future expected acceptance of wireless devices as delivery platforms for multimedia will effectively result in a return to connection speeds of the mid-1990's. The only justifiable conclusion that one can draw is: the Internet will remain heterogeneous for a long time to come.



Data Type	Size (Kbytes)	Access Time			
		Mobile Modem[a]	ISDN[b]	Cable Modem[c]	ADSL[d]
Text (1 pg, raw)	5	0:00:01.71	0:00:00.42	0:00:00.01	0:00:00.00
Text (1 pg, formatted)	51	0:00:21.39	0:00:05.34	0:00:00.10	0:00:00.05
Color image (256x256)	65	0:00:27.30	0:00:06.83	0:00:00.15	0:00:00.07
Color image (full screen)	3,100	0:21:50.72	0:05:27.50	0:00:07.25	0:00:03.50
Audio sequence (10 sec.)	160	0:01:06.66	0:00:16.43	0:00:00.41	0:00:00.16
Audio sequence (1 minute)	10,500	0:58:40.00	0:14:39.12	0:00:23.43	0:00:11.73
Video sequence (short)	70,000	6:28:53.34	1:37:12.75	0:02:46.31	0:01:17.78
Video sequence (long)	151,500	42:00:00.00	9:58:51.67	2:52:48.73	1:23:20.00

Table 3-6. Comparative speeds of transfer technologies.
[a] Assuming a 28.8Kb Modem with an average transfer rate of 3KByte/second.
[b] Assuming a 128Kb Modem with an average transfer rate of 14KByte/second.
[c] Assuming a 4Mb Modem with an average transfer rate of 450KByte/second.
[d] Assuming a 8Mb Modem with an average transfer rate of 900KByte/second.

Network Protocols and Media Access

The current Internet can be partitioned by the protocols it uses to transfer information among its constituent computers. These are TCP (the *Transport Control Protocol*) and IP (the *Internet Protocol*). All computers that connect to the Internet use these protocols, which were developed in the late 1960's and early 1970's to handle (relatively) small amounts of text-based network traffic.

The early Internet was used to send files and e-mail messages as bulk, batch transfers. When the Web became popular in the mid-1990's, the Internet began to be used for massive on-line access to formatted information. This informa-

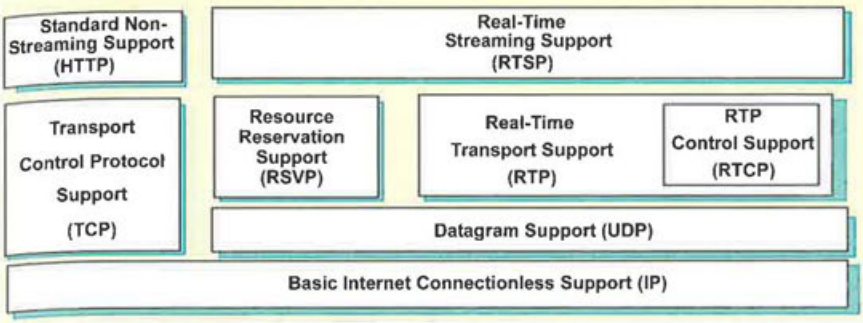


Figure 3-4. Types and relationships among network protocols for multimedia.

tion was still sent as a batch (typically, a page at a time), and could easily be transferred on top the existing TCP/IP protocol stack. Nearly all non-streaming media transfers that use a whole-file model make use of these base protocols, neither of which has any notion of time, tempo or delivery guarantees.

When the first multimedia computers arrived in the early 1990's, it became clear that new protocols were needed to guide continuous data from servers to clients. In order to address these needs, several new protocols were developed to provide support for streaming media. IP is still in wide use, and remains the dominant backbone protocol of the Internet. Above IP in the protocol stack comes a very simple protocol for packaging user data called *user datagram protocol*, UDP. (UDP is also a first-generation Internet protocol that has regained favor because of its lightweight nature.) Above these are streaming protocols: the *real-time streaming protocol* RTSP, the *resource reservation protocol* RSVP, and the combined *real-time transfer* and *real-time control protocols* RTP/RTCP. The full protocol stack is summarized in Figure 3-4.

Using these protocols, a server can request a guaranteed amount of bandwidth to be reserved before a transfer starts (using RSVP), then package the information for 'real time' transfer using RTP/RTSP and then allow information to be incrementally given to the user via RTSP. At present, some form of each of these protocols is in use.

Unfortunately, these streaming protocols have brought new problems to the Internet. Consider RSVP. Without going into detail on how the protocol works, it is obvious that the notion of resource reservation presents a concern for the various hub ISPs that make up the Internet backbone. The core issue is that if one customer says "please reserve 5% of your capacity for me for my use in the (near) future", the hub ISP has a major problem: it has no guarantee that the server will actually use the reserved bandwidth and it has no way of deciding among reservation requests when they arrive. For this reason, substantial use of resource reservation will not be seen across the generic Internet very soon. The resource reservation potential for mobile and wireless units is greater. This is a consequence of the lower bandwidth available at the device (meaning that

without some sort of resource management, the entire infrastructure will collapse) and the fact that more of a substantial portion of the delivery infrastructure is managed by single, large entities who can impose their own management policies.

Another problem comes with RTP and RTCP. Unlike TCP, UDP and IP, RTP/RTCP are *framework* protocols. RTP does not specify how to move data between a server and a client in 'real time', it simply defines how a server and client are to format their requests. There is no guarantee that all servers and clients will support the same dialects of RTP and RTCP, which may make the protocols useless for the general Internet community. Considering the time it has taken for TCP/IP to become standard parts of computer system support, it may take some time before general end-to-end support for continuous media is available.



Streaming servers and streaming media encoding

While generic support for streaming media using open protocols seems to be a distant vision, special proprietary implementations of streaming media support dominate Internet media use. These implementations typically combine proprietary media compression implementations with proprietary implementations and extensions to basic RTP/RTSP/RTCP protocols. The proprietary nature of these implementations guarantees a high-degree of end-to-end compatibility for users of the media formats, but it also provides a lock into various media player vendors such as Apple, Microsoft and RealNetworks. This makes it difficult for third party media players and media infrastructures to break into the Internet infrastructure.

3.5 Summary and Conclusion

The SMIL specification makes very little demands on the data types and protocols used to transfer media between clients and servers. At the same time, since it imposes no restrictions, it also can't provide guarantees that every SMIL player will be able to play every SMIL presentation. This situation may be a short-term problem. The wireless market is providing a measure of openness in the multimedia protocol market. These vendors, who compete based on connect time instead of media encoding, want to have a wide-open network for data (but they want the delivery pipeline to be as closed as possible!). If the wireless world is able to have a major commercial impact, it may also result in a more open desktop Internet as well.

3.6 Further Resources

HTTP Specification

Hypertext Transfer Protocol, Version 1.1. W3C RFC-2616, June 1999.
`http://www.w3.org/Protocols/rfc2616/rfc2616.html`.

IP Specification

Internet Protocol (IP). IETF RFC-791, September 1981.
`http://www.ietf.org/rfc/rfc0791.txt`.



TCP Specification

Transmission Control Protocol (TCP). IETF RFC-793, September 1981.
`http://www.ietf.org/rfc/rfc0791.txt`

UDP Specification

User Datagram Protocol (UDP), IETF RFC-768, August 1980.
`http://www.ietf.org/rfc/rfc0768.txt`

RSVP Specification

Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification, IETF RFC-2205, September 1997. `http://www.ietf.org/rfc/rfc2205.txt`

RTP Specification

A Transport Protocol for Real-Time Applications (RTP), IETF RFC-3550, July 2003.
`http://www.ietf.org/rfc/rfc3550.txt`

RTSP Specification

Real Time Streaming Protocol (RTSP), IETF RFC-2326, April 1998.
`http://www.ietf.org/rfc/rfc2326.txt`

Ambulant Player

For information on the SMIL 2.0 compatible Ambulant player, see:
`http://www.ambulantPlayer.org/Player/`.

GRiNS Player

For information on the SMIL 2.0 compatible GRiNS player, see:
`http://www.oratrix.com/`.

Quicktime Player

For information on the SMIL 1.0 compatible QuickTime player, see:
`http://www.apple.com/quicktime/`.

RealNeworks RealOne Player

For information on the SMIL 2.0 Language Profile compatible RealOne player, see: `http://www.real.com/`.

HTML+TIME Browser

For information on the SMIL 2.0 XHTML+SMIL compatible HTML+TIME player, see:

`http://msdn.microsoft.com/library/default.asp?url=/workshop/author/behaviors/reference/time2/htime_node_entry.asp`.



Basic SMIL Constructs

This part covers the basic functionality of SMIL 2.0. The elements and attributes described in chapters 4 through 9 probably will constitute 90% of the standard SMIL use. Nearly all of the material in this part can be used with the SMIL Language profile (including desktop and mobile SMIL) and — with the exception of layout — with XHTML+SMIL as well.

Topics Covered



Chapter 4 takes an in-depth look at the overall structure of a SMIL presentation. We describe the elements and attributes that are needed to encode the framework of a presentation in the various SMIL profiles. We also look at the XML facilities available to make a SMIL presentation future-proof.



Chapter 5 considers the inclusion of media assets in a presentation. It describes how media are classified, how they are referenced and how they are controlled from within a SMIL presentation. We look at the various types of media references supported by SMIL and the timing consequences of adding media to presentations.



Chapter 6 describes the basic facilities supported by all SMIL 2.0 Language profile versions for creating and managing presentation layout. We look at the layout elements and attributes in the head section and then consider how these elements can be applied to various types of media.



Chapter 7 provides a description of the temporal control of media objects within a SMIL presentation. We look at the basic time containers used by SMIL and then consider the basic set of timing attributes available to SMIL authors. This chapter covers the timing assets most used in basic SMIL presentations, in all SMIL profiles.



Chapter 8 describes the linking facilities in SMIL. It starts with a discussion of how links are used in HTML and then contrasts this with the time-driven linking facilities available in SMIL 2.0. It considers how SMIL defines anchors and link attributes and illustrates how SMIL linking can be used for message passing and activation.



Chapter 9 provides an overview of basic content control facilities within a SMIL presentation. It starts with background information on content selection and then describes the elements and attributes that SMIL provides that allow a presentation to be tailored to the needs of the network infrastructure and the target player.



Chapter 10 concludes this part with a discussion of the visual transitions provided in SMIL 2.0. We begin with examples of transition effects and then look at how transitions can be defined. We then look at the elements and attributes provided by SMIL for basic and in-line specification of transitions for all media.

Goals

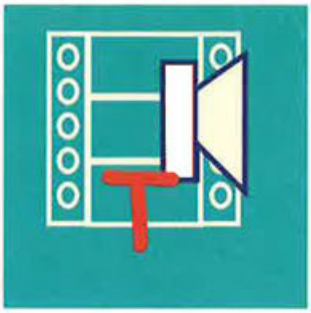
By the time you complete this part, you should be able to author presentations that make use of all of SMIL's basic multimedia presentation structuring facilities. You will also be able to understand and maintain SMIL presentations written by others for the various SMIL 2.0 profiles. The key to this part is that we only consider attributes and elements that are widely supported by SMIL's various players and profiles.

Prerequisites

In order to get the most out of this part, you should have read and understood the material in Part One of this book. You should also be able to open an existing SMIL presentation and send it to one of the major SMIL players for rendering.

5

Referencing Media Objects



"Integration" is SMIL's middle name. SMIL does not define any media itself — it defines the glue that holds media together in multimedia presentations. SMIL's *BasicMedia* module and its constructs, the media object elements and the `src` and `type` attributes, provide the doorways though which media is brought into SMIL document structure. SMIL can define a bit of its own media with the `<brush>` element, which paints color on an area of the display and makes it a media object in its own right.

This chapter describes the basic alternatives for identifying the media objects to be contained in a presentation. We won't discuss layout or timing issues yet — these come in later chapters — but we will review everything you need to understand how SMIL references external media objects.

5.1 Partitioning Content and Control

A SMIL specification contains elements and attributes that allow media objects to be located and scheduled for presentation. (It also defines how linking and overall resource control can be integrated across the media items.) The SMIL specification *does not* define how the actual media is to be formatted. For SMIL implementations where SMIL is the host language (such as those based on the Language and Mobile profiles), there is a strict separation between the control information in the SMIL file and the content in media objects. For implementations of SMIL where another language serves as host (such as XHTML or SVG), the separation between content and control is less strict. The advantage of having a strict separation between content and control is that the SMIL file remains small: this allows the file to be parsed and handed off to the scheduler with a minimum of extra bits. (This is obviously important for small devices.) The advantage of mixing control elements and content is that a presentation can be easier to hand-edit: everything is in one file and the mixed control/content conventions that are used in HTML are maintained. For presentations that are largely text-based (such as those using the XHTML+SMIL profile), mixing control and content makes sense. For presentations that focus on images, audio and video, the separation of content and control is much more useful. The ultimate choice is more philosophical than technical.

```
<smil>
 <head>
  ...
 </head>
 <body>
  ...
  <text src="file:///hello.txt" type="text/plain" ... />
  ...
 </body>
</smil>
```

(a) using <text> and an external media object



```
<smil>
 <head>
  ...
 </head>
 <body>
  ...
  <ref src="http://www.example.net/hello.txt" type="text/plain" ... />
  ...
 </body>
</smil>
```

(b) using <ref> and an external media object

```
<smil>
 <head>
  ...
 </head>
 <body>
  ...
  <text src="data:,Greetings!" type="text/plain" ... />
  ...
 </body>
</smil>
```

(c) using <text> and an embedded media object

Example 5-1. Partitioning control and content in the SMIL Language profile.

In order to understand the various options that SMIL provides for identifying media objects, look at the three code fragments in Example 5-1. All of these examples use the SMIL Language profile — this is because SMIL profiles that use another host language (such as HTML) will use that language's media referencing primitives. In each example, we want to display the text: Greetings!

In Example 5-1(a), we see the typical case for including media in a SMIL presentation. The element <text> specifies a media object reference, the location and type of which are given by the src and type attributes. The actual text string is placed inside the referenced file. (Note that the src attribute uses conventional URI syntax. Please review *URIs: Locating and Naming Media Objects* on page 62 for a discussion on URIs.)

Example 5-1(b) shows a subtle variant of the SMIL example, in which the <text> element has been replaced by the <ref> element. Functionally, these two fragments are identical: in the full SMIL Language profile, the <text> element is simply a synonym for <ref>. (More on this starting on page 88.)

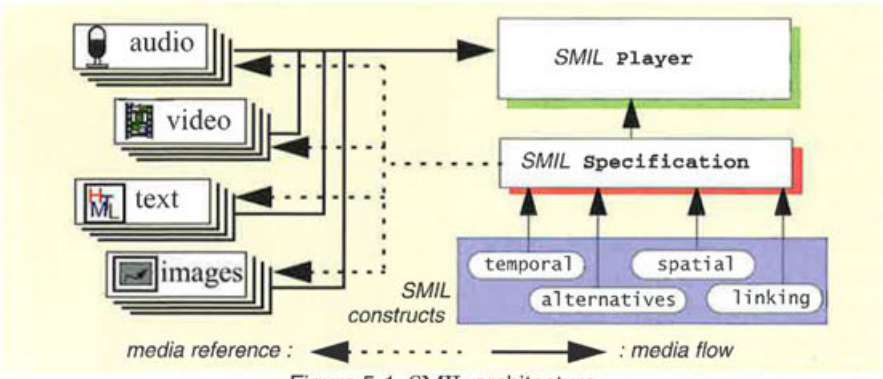


Figure 5-1. SMIL architecture.

Although most media used in a SMIL presentation will be referenced as an external object, it is also possible to embed media directly in the SMIL file. Example 5-1(c) gives an example of this using a *data* URI: the `src` attribute starts with the characters `data:,` (including the ',!), followed by the actual content of the media.[1] For short text strings, this is a convenient mechanism, but for a 10MB piece of video, it is not particularly appropriate!

The basic architecture of SMIL in the SMIL Language profile is presented in Figure 5-1. The SMIL specification contains references to media (and other control information), but the media itself is located in external files.

We close our discussion on separating content and control with an example of non-SMIL. Many new users to SMIL expect that the following code fragment should be valid:

```
...
<text>
  Greetings!
</text>
...
```

It is important to realize that *this is not valid* SMIL. Whatever the merits of such a structure, it is not supported in the SMIL 2.0 Language profile. In other profiles, such as XHTML+SMIL, the `<text>` element is not used, but embedded content using the `<P>` element (and its equivalents) is more common.

1 The data URI can also accept parameters before the actual data. Since the separator character for these parameters is a comma, a final comma is expected before the actual data encoding starts. This is why the comma is required in the data string.

5.2 SMIL Media Object Elements and Attributes

The SMIL specification's media object modules define three sets of elements for referencing and controlling media objects in a presentation: *media object elements*, a *media object initialization element* and a *brush element*. Of these, the media object elements and the brush element represent basic SMIL functionality and are discussed in this chapter. The media object initialization element (which is named `param`) is advanced SMIL functionality and is treated in Chapter 11: *Subsetting and Extending Media*.

SMIL's media object elements are defined in the SMIL *BasicMedia* module. This module also defines the `src` and `type` attributes. All SMIL profiles include the *BasicMedia* module. The `<brush>` element is defined in the *BrushMedia* module. This module also defines the `color` attribute. Only the SMIL Language profile makes use of the *BrushMedia* module.

The XML structure of the media object and brush elements is shown in Table 5-1.

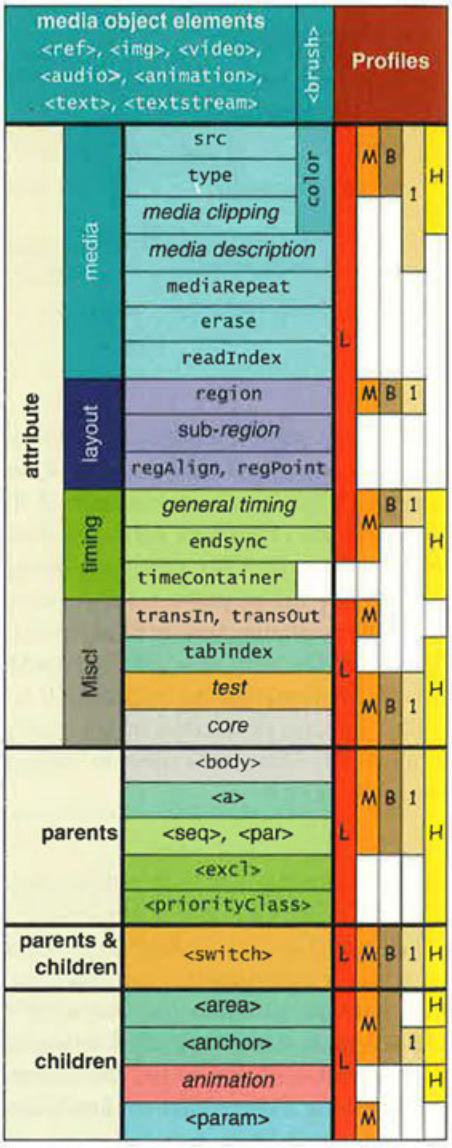


Table 5-1. Structure of the media object elements

5.2.1 Media Elements

This section defines the media object elements in SMIL.

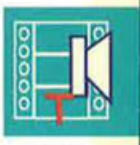
Element: <ref>

SMIL defines one media object element to specify which media objects are included in a presentation: the `<ref>` (reference) element. In order to make reading SMIL files easier, the language also defines six synonyms for `<ref>`: `<img>`, `<video>`, `<audio>`, `<animation>`, `<text>` and `<textstream>`. These seven elements are identical in every way except for their names — they take the same attributes, can have

elements of the same types for parents and children, and behave the same. In fact, these elements are totally interchangeable.

The SMIL Language profile uses different names as an authoring convenience (so that someone can read the SMIL file and understand what is going on), but they are not used by the SMIL player to determine the type of media content. This means that a (perverse!) author could use a <video> element to reference a file containing audio media. While not recommended, this works because a SMIL player is not allowed to use the name of the media object element to determine media types. Instead, either the type attribute or the media server should determine the type (and inform the player).

One exception to the flexible use of media element names applies to some mobile implementations of SMIL. In order to minimize parsing and processing overhead, these implementations often require that only an appropriately typed media object element be used in a SMIL file. These restrictions and the general rules for determining media types are discussed on page 92.



The placement of a media object element in the <body> element hierarchy determines *when* (and *if*) it gets played. We'll spend much of the rest of this book discussing the many intricacies of how this is determined. But for now, here's a (very) short summary how the type of a media object element's parent helps determine when it plays:

- <body>, <seq> — before or after its siblings
- <par> — along with its siblings
- <switch>, <excl>, <priorityClass> — instead of its siblings
- <a> — in response to user interaction

The potential child elements of media object elements play a variety of roles in relation to the media. The <a> and <area> elements (and <area>'s deprecated SMIL 1.0 equivalent <anchor>) make portions of the media item "clickable" by the user — the user can activate these portions of the media to make the presentation lead elsewhere. The <param> element specifies particular modifications to the media for its playback in the presentation. Child animation elements define animations, or progressive changes in the presentation structure, that occur while the media plays. The <switch> element gives alternative specifications of these other elements. We describe all these element types in full detail later in the book.

As shown in Table 5-1, all of the media object references can accept a large number of attributes. Of these, only a few deal with media object selection: src, type and various media clip attributes. Each of these are described in *Media Object Attributes* on page 90. All of the other attributes that are available are discussed in other chapters in this book.

Element: <brush>

The <brush> element is just like a media object element except that it defines a fill color for a rendering region rather than an external media resource. Since it

doesn't refer to external media, the <brush> element doesn't need the src and type attributes. However, since it defines color display internally, it has an attribute other media object elements don't: color.

The code fragment in Example 5-2 illustrates the use of the <brush> element. A rendering space (in this case, a SMIL layout region named *window1*) is defined in line 2. The brush element in line 4 says: take the specified color (in this case, *blue*) and use it to fill the entire referenced region (in this case, *window1*).

Valid definitions for the color attribute are discussed on page 95.



```
1 ...
2 <region regionName="window1" ... />
3 ...
4 <brush color="blue" region="window1" ... />
5 ...
```

Example 5-2. Use of the <brush> element.

5.2.2 Media Object Attributes

Media object elements have many attributes, as shown in Table 5-1, which define various aspects of how media get presented in SMIL. These include the source and type of the media and (for <brush>) the rendering color. They further specify the subset of the media object to be integrated by selecting portions of the located media file and making modifications to its own intrinsic timing. The layout attributes other than region define fine-tuning of spatial positioning unique to the media object that go beyond what is specified for it in the document layout as a whole. General timing attributes fine-tune when the media item plays beyond the broader synchronization defined for it by its parent. The attributes transIn and transOut define visual transitions that occur on the media as it starts and finishes. The user can use multiple taps on the tab key to highlight different media objects with tabIndex. Finally, the test attributes help determine if the media should be elected for presentation to a given user.

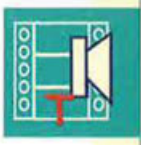
This section focuses on those attributes that relate directly to the selection of the media object. We start with the source attribute, then consider the type attribute and then discuss attributes for defining sub-parts of a media object. We close with a discussion of the color attribute.

Attribute: src

The src (source) attribute gives the location of a media item to be retrieved for integration into a presentation. As in HTML, its value is a URI. Unlike HTML, SMIL's src attribute can reference any media type.

The src attribute will typically reference an external file containing either a single media object or a composition of media objects in some high-level

encoding. (An HTML file containing text and images, a SMIL file containing a sub-presentation or a Flash file containing an animation are three examples of higher-level compositions.) As a language, SMIL does not place any restrictions on the media object identified in the `src` attribute, but certain SMIL players may have local restrictions on the types of media they can support. For example, mobile SMIL players may not support full HTML media, and XHTML+SMIL browsers may not support RealMedia proprietary datatypes. See Chapter 3: *Local and Streaming Media* for a discussion of media encoding issues.



Example 5-3 illustrates the use of the `src` attribute for typical cases in the SMIL Language profile. The reference on line 4 illustrates the fetching of an MPEG video via the RTSP streaming protocol. Note that both the SMIL client (your player/browser) and the media server need to support RTSP for this transfer to be successful. The reference on line 5 illustrates the fetching of an SVG animation. The transfer is controlled using the HTTP protocol, which means that the entire file will need to be downloaded before rendering can begin — assuming, of course, that the SMIL player supports the SVG format. The reference on line 6 illustrates the fetching of a local audio file. Line 7 also fetches a file that is located in the same directory on the same host as was the SMIL file. Although this is a valid use of the `src` attribute, it is only recommended if you are sure the relative location of the media won't change. Line 8 illustrates the use of a data URI to encode information directly into the SMIL file. If this is used, make sure you only use it with small amounts of media to avoid generating very large SMIL files.

The references on line 4 through 8 represent standard use cases for SMIL. The reference on line 9 illustrates another aspect of SMIL behavior: the `src` attribute is an optional part of a media object specification. If no `src` attribute is given, this is not an error — although, at first glance, it is pretty useless! If a media object reference is made to an object without a corresponding source file, no media is fetched (obviously), but the reference fully participates in the timing structure of the presentation. In effect, line 9 defines a temporal delay in the presentation of 10 seconds.

```
1 ...
2  <body>
3    ...
4    <ref src="rtsp://www.example.net/meAndMyDog.mpg" ... />
5    <animation src="http://www.example.net/ourAddress.svg" />
6    <audio src="file:///myCar.mp3" ... />
7    <image src="myGarage.jpg" ... />
8    <text src="data:,Hi Mom!" ... />
9    <video dur="10s" />
10 ...
11 </body>
12 ...
```

Example 5-3. Uses of the `src` attribute in the SMIL Language profile.

Attribute: type



Once the player identifies and acquires a media object, it needs to know how to render it. In order to do this, the player selects either an internal or external rendering engine that is compatible with the format of the media. The SMIL player will contain a set of preferences that indicate which rendering engine should be used for each media type. The question is: how does the player know what kind of media is stored in the file.

The primary means of stating this is with MIME Media Types (or, more simply, *mimetypes*). A mimetype is a character string that defines the class of media (audio, video, image, text, application) and a media encoding type (such as jpeg, mpeg, RealVideo, etc.). For each media object used in the SMIL player, someone or something has to associate a mimetype with a media object. Table 5-2 defines common mimetype definitions.

Media Class	MIME Type
audio	audio/basic audio/mp3 audio/mpeg audio/vnd.rn-realaudio audio/x-aiff audio/x-mp3 audio/x-ms-wma audio/x-pn-realaudio audio/x-realaudio audio/x-wav
video	video/mpeg video/msvideo video/quicktime video/vnd.rn-realvideo video/x-msvideo video/x-ms-wmv
image	image/bmp image/gif image/jpeg image/png image/svg-xml image/tiff image/vnd.rn-realflash image/vnd.rn-realpix image/x-cmu-raster image/x-emf image/x-rgb image/x-wmf
text	text/html text/plain text/vnd.rn-realtext
composite	application/vnd.rn-realmedia application/x-shockwave-flash application/smil application/xhmtl+smil

Table 5-2. Common *mimetype* definitions.

Mimetypes may be *registered* or *informal*. Registered mimetypes are controlled by the Internet Assigned Numbers Authority (IANA). Informal mimetypes are not registered with IANA, but are defined by common agreement; they usually have an "x-" before the media type name.

The mimetype association for a media object can be made by:

- *the local computer's operating system*: each operating system typically keeps a table that associates mimetypes with part of a media object's name (usually a three- to five-character suffix, such as "txt" or "smil");
- *the user*: a user can define preferences in the player to associate media object names with mimetypes;
- *the author*: the designer of the presentation can explicitly state the mimetype of the media object, perhaps overriding the default processing that is associated with a particular name; or

- *the server*: the mimetype can be supplied by the media server when the object is transferred. (If it is local media, then the operating system can assign a mimetype.)

While associating a mimetype with a piece of media may seem like a pretty simple operation, there are all sorts of complexities involved. For example, some protocols may allow content substitution, so that what you get may not be exactly what you requested. The type of the media may also be unknown at author time, and it may be so new that the operating system may not know about it either.

SMIL defines a set of rules for mimetype association that distributes responsibilities across several parties. The general approach used is:



- If the transfer protocol defined in the `src` attribute is RTSP, then the value of the `type` attribute will be used to select an appropriate version of the request media object on the server. If no such object exists, the server returns what it thinks is an appropriate media object (and gives its type).
- If another transfer protocol is used (such as HTTP or FTP), the value of the `type` attribute takes precedence over any other means to determine the media type.
- For other protocols, the value of the `type` attribute should take precedence unless the protocol itself identifies a particular media type.

After the SMIL specification was published, updates to the standards for Web transfer protocols defined new precedence rules for determining mimetypes. These rules essentially suggest that the player should always trust the transfer protocol to identify the correct mimetype. Only if the protocol explicitly neglects to provide a mimetype should the value of the `type` attribute be used — and only if no `type` attribute is available should the file type suffix be used to determine media type. Note that in no cases may the name of the media element (`text`, `audio`, `video`, etc.) be used to distinguish media type.

Example 5-4 illustrates the use of the `type` attribute for typical cases in the SMIL Language profile. Line 4 illustrates how, using the RTSP protocol, the type can be used to select among various media encodings on the server. Line 5 illustrates how, using the HTTP protocol, the SMIL `type` attribute can be used to override that default association of a media object's file extension. Line 6 shows how the media element name (text) is ignored and overridden as a GIF image.

```
1 ...
2  <body>
3    ...
4    <video src="rtsp://www.example.net/myDog" type="video/mpeg" ... />
5    <audio src="http://www.example.net/myCar.txt " type="audio/mpeg" ... />
6    <text src="...ffc2" ... />
7    ...
8  </body>
9 ...
```

Example 5-4. Uses of the `type` attribute in the SMIL Language profile.

As of the writing of this book, the future treatment of mimetypes by new SMIL players was still under discussion. For standard media types (such as *.jpeg* or *.txt*) there will probably be little problem, but if your application integrates new media types or new transfer protocols, you should check the fine-print of your SMIL player's documentation.

Attribute: clipBegin and clipEnd



The attributes presented thus far allow a media object to be identified and, optionally, associated with a media type. For many simple uses of media, these are all the (media) attributes that you will need to start playing media in your presentation. In some presentations, however, you may not want to play an entire media object. Instead, you may want to play only a particular fragment. SMIL supports this via the media clipping attributes that are defined in SMIL's *MediaClipping* module.

Playing a subset of a media object typically only makes sense for continuous media, such as audio and video. You can identify the media object you want to take the clip from by using the `src` attribute. You can then specify the place where the clip is to be using the `clipBegin` attribute. (If no `clipBegin` is specified, the object always starts at the beginning.) You can also specify the end of a clip by using `clipEnd`. (If no `clipEnd` is defined, then the media object plays to completion.)

The only major issue to be resolved when using `clipBegin` and/or `clipEnd` is: how do you specify the temporal point of the beginning/end of the clip? Do you use seconds (or minutes, or hours) or do you identify individual samples (or frames) in the source media? SMIL lets you do both.

Example 5-5 illustrates the use of the `clipBegin` and `clipEnd` attributes for typical cases in the SMIL Language profile. For purposes of example, assume that the video *myDog.mpg* is 60 seconds long. Line 4 shows a clip that starts 20 seconds into the video and then plays to the end. (The metric *npt* stands for "normal play time".) It says: start rendering the video 20 seconds into the video's normal play time. Line 5 also starts about 20 seconds into the

```
1 ...
2  <body>
3    ...
4    <video src="rtsp://www.example.net/myDog" type="video/mpeg"
         clipBegin="npt=20s" ... />
5    <video src="rtsp://www.example.net/myDog" type="video/mpeg"
         clipBegin="smpte-25=0:00:19:01" ... />
6    <video src="rtsp://www.example.net/myDog" type="video/mpeg"
         clipEnd="npt=40s" ... />
7    <video src="rtsp://www.example.net/myDog" type="video/mpeg"
         clipEnd="smpte-30-drop=0:00:39:24" ... />
8    <video src="rtsp://www.example.net/myDog" type="video/mpeg"
         clipBegin="20s" clipEnd="40s" ... />
9    ...
10 </body>
11 ...
```

Example 5-5. Uses of the `clipBegin`/`clipEnd` attributes in the SMIL Language profile.

video, but in this case, it identifies a particular frame in the video using the SMPTE/PAL encoding format "smpte-25". (In this example, the second frame of the 20th second is selected — that's pretty precise for a dog!) Line 6 and 7 show how the end of the clip can be identified using both NPT and SMPTE encodings (in this case, using SMPTE/NTSC notation). Line 8 shows that, when no timing metric is given, 'npt' is assumed.

If the NPT metric is used (or implied), its value is a SMIL clock value. These are defined in Chapter 14: *Advanced SMIL Timing Attributes*. SMPTE encodings are a bit more complex; see Chapter 11: *Subsetting and Extending Media* for a description.



There are some important performance and licensing implications of using `clipBegin` and `clipEnd` in a presentation. First, identifying a short media fragment (such as a 10 second clip out of a two hour video) will not necessarily mean that only 10 seconds of video are transferred between the server and the client. Since `clipBegin` and `clipEnd` are SMIL attributes (not transfer protocol attributes), it could well be that the entire video is transferred across the network and then only a short fragment played on the local machine. (Some players and protocols may be able to optimize this performance, but this is a player issue, not a SMIL issue.) Second, if you are accessing media from a pay-per-view server, you may still be charged for a complete media access even if only a short part of the media is used.

Quick Tip

In SMIL players that support the full SMIL 2.0 specification, both the attribute names `clipBegin` and `clip-begin` (as well as `clipEnd` and `clip-end`) must be supported. `clip-begin` and `clip-end` are the SMIL 1.0 names for the media clip attributes. Note also that SMIL 1.0 did not default its clip metric to 'npt'; strictly speaking, clip-end="10s" is an invalid specification.

Attribute: color

The `color` attribute of the `<brush>` element and the `backgroundColor` attribute in `<region>` specifies a color that is to be displayed. The syntax of the values is illustrated in Figure 5-2. *Color values*, the only practical values of the `color` attribute, are described next.

A *color value* can be the name of a color, the numeric definition of a color or a color setting on the user's Web browser. The syntax of the color values is illustrated in Figure 5-2.

Color Names. SMIL, and the Web as a whole, establish 16 color names as short cuts for defining colors. These color names are shown in Figure 5-2. Be careful when using color names because some Web browsers use additional non-standard color names that other browsers may not recognize — make sure to use only one of these 16 standard names when using color names.

RGB Color Specifications. The Web presentation standards share not only the 16 color names but also the means of defining all of the colors the Web can present. This scheme is based on the use of *red*, *green* and *blue* as the primary colors for creating color combinations. From these primary colors comes the name of the scheme: *RGB*. An RGB specification states how much of each primary color is used in making the color defined. The intensity levels of the primary colors are stated in Web color definitions with three primary color coordinates. There are three ways to state these three coordinates: *percentage triplets*, *decimal triplets* and *hexadecimal notation*, described below. Of these, hexadecimal notation is the most often used and the most frequently recommended, since it is the most compact and is completely precise.



- *RGB percentage triplets*: The most easily learned way to define colors on the Web is with *RGB percentage triplets*. These state what percentage of each primary color is mixed into the color shown. These consist of the letters "*rgb*" followed by a pair of parentheses, known as *RGB functional notation*. In between these parentheses, there are three percentage values. Each is a number, either integer or floating point, from 0 to 100, followed by a percent sign. These percentage values are separated by commas followed by optional white space characters, which typically consist of a single space.
- *RGB decimal triplets*: A more precise way to specify a desired Web color is with an *RGB decimal triplet*. RGB percentage triplets can be inaccurate for color representation because not all percentage values correspond with one of the 256 8-bit values computers use for color. RGB decimal triplets have the same syntax as RGB percentage triplets, except that the three coordinates are not percentages but integers in the range from 0 to 255, and that they are, of course, not followed by percent signs. This numeric range of 256 different values ensures that each RGB decimal triplet corresponds directly with one possible computer display color, and vice-versa.

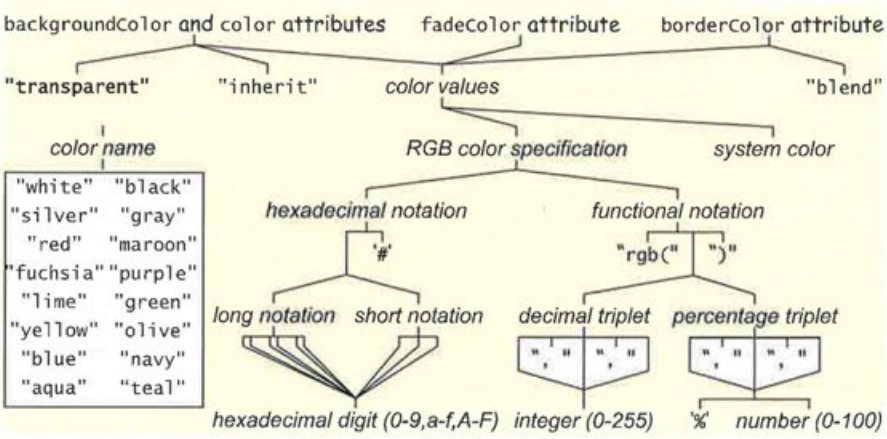


Figure 5-2. Syntax for values of attributes using color

- *RGB hexadecimal notation:* The most concise, most common, and most highly recommended color definitions are in *RGB hexadecimal notation*. They have the same accurate one-to-one correspondence that RGB decimal values have, but with fewer characters. RGB hexadecimal notation typically consists of seven characters, though a short form usable for some colors consists of only four characters. RGB hexadecimal notation always begins with a '#' followed by six hexadecimal numbers. Hexadecimal notation uses 16 "digits" that increase from `'0'` to `'9'` and then from `'A'` to `'F'` or `'a'` to `'f'`.

  For the full seven-character notation, called *long notation*, two of these hexadecimal digits are used for each primary color coordinate. The first digit puts you in a neighborhood of 16 adjacent primary color intensities, and the second chooses one of these 16. The four-character *short notation* for RGB hexadecimal consists of a `'#'` followed by three or six hexadecimal digits, one for each primary color coordinate. The trade-off of this shorter notation is that it defines fewer colors.



System Colors. The `Color` attribute can also be assigned any of the *system colors* defined by CSS. These colors are not set by the author but by the user, or by the browser on behalf of the user. System colors enable users to set a color scheme for their presentations that best suites them, perhaps by being tailored to their visual disabilities. They also allow presentations to blend in with the system's and windowing environment's color scheme. In SMIL, use of system colors provides visual consistency when integrating CSS-styled media that also uses system colors. Authors should be careful when using both system colors and author-defined colors. System colors can vary considerably from user to user and in some cases may clash with the colors set by the author. Similarly, system colors may clash with colors set in visual media integrated into the SMIL presentation.

5.3 Timing Consequences of Media Object Use

Whenever a media object is added to a SMIL specification, it is assigned a duration. The value of this duration depends on a number of factors. For continuous media objects such as audio or video, the intrinsic duration of the media is used if this can be determined by the player. (This information is often — but not always — stored in the header of the media object.) For discrete media objects, the duration of the object defaults to 0 seconds, which is not very long. It may seem strange that a media object is assigned a duration of 0 seconds: if an author wanted a 0 second object, it would seem simpler to not specify any media at all! For a SMIL perspective, using 0 seconds makes perfect sense: first, if you didn't use 0, what else would you use? Second, SMIL makes a distinction between an object's *timing duration* and its *rendering duration*. (By default, an object with duration of zero will stay visible until its time container ends.) Both of these topics are discussed in detail in Chapter 7: *Basic SMIL Timing*.

5.4 Summary and Conclusion

The *BasicMedia* module and its constructs, the media object elements and the `src` and `type` attributes, locate media and place it in SMIL presentation structure. The `src` attribute states where the media is and how to download it. The `type` attribute lets the browser choose the right process for rendering the media into the presentation. A media object element's placement in the body tree determines much of when and if the media object is played. Other attributes of a media object element determine the rest of the object's timing. The *BasicMedia* module is used by SMIL 1.0 and all SMIL 2.0 profiles. Unique to the SMIL Profile, the *BrushMedia* module and its constructs, the `<brush>` element and its `color` attribute, allow the SMIL author to define rectangles of color as media objects.



5.5 Further Resources

URIs

RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, Tim Berners-Lee, Roy Fielding, Larry Masinter, IETF (Internet Engineering Task Force), 1998, `http://www.ietf.org/rfc/rfc2396.txt`.

Relative URIs

RFC 1808: Relative Uniform Resource Locators, Roy T. Fielding, IETF (Internet Engineering Task Force), *June 1995*, `http://www.ietf.org/rfc/rfc1808.txt`.

XML *Base*

XML *Base*, Jonathan Marsh (ed.), W3C Recommendation, 27 June 2001, `http://www.w3.org/TR/xmlbase/`.

SVG

Scalable Vector Graphics (SVG) 1.0 Specification, Jon Ferraiolo (ed.), W3C Recommendation, 4 September 2001, `http://www.w3.org/TR/SVG`.

HTML

XHTML 1.0: The Extensible HyperText Markup Language (Second Edition), Steven Pemberton, (ed. chair), W3C Recommendation, 1 August 2002, `http://www.w3.org/TR/xhtml1`.

CSS

Cascading Style Sheets, Level 2, Bert Bos, Håkon Wium Lie, Chris Lilley and Ian Jacobs (eds), W3C Recommendation, 12 May 1998, `http://www.w3.org/TR/CSS2`.

7
Basic SMIL Timing



SMIL timing defines when elements in a presentation get scheduled and, once scheduled, how long they will be active. The SMIL timing facilities are the core contribution of the SMIL standard: using the elements and attributes defined in the 19(!) timing modules, time can be integrated into any XML language.

Many multimedia authors are familiar with basic timing concepts. Tools to create individual media assets — such as video and audio objects — have long integrated some notion of timing when constructing and combining audio and video assets. Unlike most media editing, however, SMIL takes timing several steps further: in a SMIL-based document, every media object and nearly every structural element has a specific timing scope. While media timing plays an important role in determining the overall duration of a presentation, the structure of the document is also used to simplify and optimize the rendering of media presentations. SMIL timing defines a collection of elements that determine the relative start and end times of document objects and a collection of attributes that control the duration, persistence, repetition and accuracy of timing relations in a document. SMIL can be used directly as the host language for a document (as is done in the SMIL Language profile and in SMIL Basic), but it can also serve as the basis for integrating time-based coordination of otherwise static elements (as is done in SVG animation and in the XHTML+SMIL profile).

In order to understand SMIL and to use its facilities efficiently, this chapter starts with a discussion on the basic aspects of the SMIL timing model. We then discuss the basic temporal elements and attributes available with SMIL, plus some powerful extensions. We then provide some examples of how timing can be integrated into various SMIL-based presentations. Timing is a complex subject. For this reason, our discussion is centered around a series of small examples, each of which illustrates an important timing element or attribute.

Quick Tip

The best way to understand SMIL timing features is to construct small demonstration programs of your own and to play them on a SMIL player. The book's Web site has a set of initial demos to get you started.

7.1 SMIL Timing Model Basics

The timing model used by SMIL to specify the (relative) begin times of media objects and their durations is based on a *structured timing model*. This means that the nested presentation structure in a SMIL document — and not only hard-coded clock values — are used to define the high-level activation and synchronization of objects. For many simple SMIL documents, you will never have to explicitly specify a begin time or a duration of a media object: the SMIL player can figure all of these out at presentation time. Of course, if you want more precise control over your presentation (such as inserting delays or specifying interactive behavior), SMIL also provides more complex timing mechanisms.

In order to use SMIL effectively, it is important to understand several key principles of SMIL's timing and synchronization model. This section provides an overview of the concepts used by SMIL and it gives background information on why SMIL does things the way it does. Later sections provide more detailed information on all of SMIL's timing facilities. The timing model is covered in more detail in Chapter 13: *Advanced SMIL Timing Behavior and Control*.

7.1.1 A Simple Slideshow Presentation

We introduce the timing issues addressed by SMIL in terms of the slideshow presentation depicted in Figure 7-1. This presentation contains a single background image on top of which a sequence of slides is placed, each with an accompanying image containing a text label and an audio file containing spoken commentary. The presentation also contains a single background music object that is played throughout the presentation.

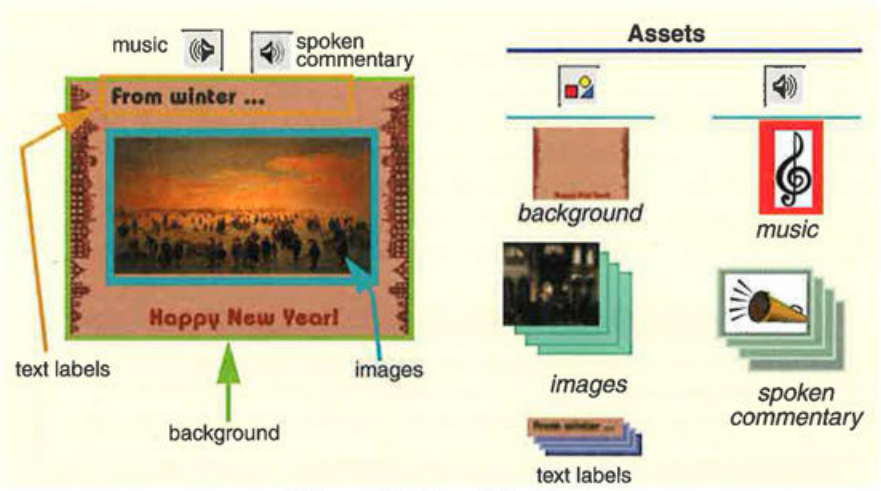


Figure 7-1. Happy New Year

The timing in this presentation is dominated by two objects: the background music object determines the duration of the total presentation and the spoken commentary objects are used to determine the duration of each of the image slides. This means that we have an outer time base for the entire presentation and a set of inner time bases for each slide in the presentation.

7.1.2 Media Object and Presentation Timing Definitions

A basic property of multimedia presentations is that they require some degree of temporal coordination among the objects being presented. The more complex a presentation — either in terms of number of simultaneous objects or number of synchronization control points — the greater the amount of control information required. SMIL uses *timing attributes* and *timing elements* to provide the activation and synchronization control information in a presentation. In general, timing attributes are used to control the timing behavior of media object and timing elements are used to control the behavior of the presentation as a whole.

Media Timing



As we saw in Chapter 3: *Local and Streaming Media*, multimedia presentations contain two types of media objects: *discrete* media and *continuous* media. Discrete media objects, such as the text labels, the background image and each of the slide images in Figure 7-1, have no implicit duration. If referenced in a SMIL file without any additional timing attributes, their duration will be 0 seconds — which is not very long. *Continuous* media, such as the background music object and each of the spoken commentaries in Figure 7-1, have implicit durations that are defined within their media encodings. If referenced in a SMIL file without any additional timing attributes, they will be rendered for the full duration defined by the object.

In our example, we want each slide image and the associated text labels to be displayed for a duration that is defined by the accompanying spoken commentaries. (Each slide/text/audio group will have different durations, since not all spoken commentary is equally long.) SMIL provides a range of attributes that allow the duration of objects to be explicitly defined and refined, and it provides a general inheritance model in which the durations of both discrete and continuous media can be obtained by the context in which a media object is presented in relation to other objects. This allows the durations of the images and text to match that of the spoken audio.

Presentation Timing

A SMIL file contains references to one or more of media objects and a set of timing primitives that determine when these objects get started relative to one another. The total timing of each of the media objects, plus any additional timing control defined in the SMIL file, determines the duration of the composite

presentation. Sometimes this composite duration can be calculated in advance, but often it can't.

The basic timing of the slideshow presentation described in Figure 7-1 is deterministic: that is, we can determine the full timing in advance of the presentation's execution by evaluating the timing of each of the continuous media objects. It is important to understand, however, that our presentation is only deterministic if we ignore several potential presentation-time delays — these include any streaming delays associated with bringing the media object from a server to the presentation device, or any delays at the client associated with decoding and rendering individual media objects. For local presentations, such as CD-ROM multimedia, it is safe to assume that all the delays in the system can be predicted in advance and factored into the presentation timing. For Web-based multimedia, where the delays when obtaining media may be considerable (and unpredictable) and where there may be wide variability in the performance of end-user devices, assuming that presentations are fully deterministic is a dangerous strategy.

A presentation with uncertain timing characteristics is called *non-deterministic*. In addition to the network streaming delays discussed above, non-deterministic timing can also be the result of content substitution within the presentation or as a result of using interactive, event-based presentation timing. We discuss the impact of content substitution on timing in Chapter 9: *Content Selection and Control* and interactive timing in Chapter 13: *Advanced SMIL Timing Behavior and Control*.

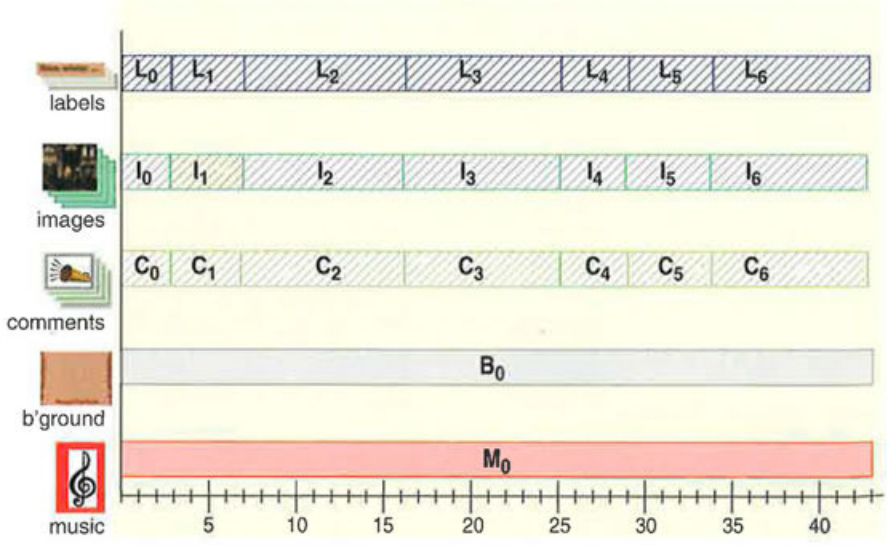


Figure 7-2. A timeline representation of the presentation in Figure 7-1.

SMIL and Timelines

A timeline metaphor is often used to model presentations. A timeline is a simple graph showing time on one axis and one or more media objects on the other axis. An example timeline, showing the elements and objects in Figure 7-1 is shown in Figure 7-2. This timeline shows the media sorted by layout: the left axis shows the various classes of media objects used, and the bottom axis shows the cumulative duration. We could have also defined separate lines for each media object, but this is usually less space efficient.

A timeline exposes the exact temporal relationships among media items. We can translate this to a text format by assigning explicit begin times for each media object and defining durations to discrete objects. One such encoding is the time-list structure of Example 7-1. The background audio and image objects (lines 1 and 2) are followed by the set of slide images (lines 3-9), the spoken commentary (lines 10-16) and the image-encoded text labels (lines 17-23). The begin times are determined by the duration of the spoken commentary objects. While Example 7-1 could be used as the basis for SMIL timing, this would be unwise. A better approach is to use structure-based timing primitives.

SMIL and Structure-Based Timing

The main advantage of the timeline model is that it is an easy-to-understand representation of continuous media objects under deterministic timing conditions. As such, it is a representation used often in video and audio media editors. Unfortunately, while deterministic timing is good for modelling video tape, it doesn't scale well to most Web environments: if one of the image objects arrives later than planned, or if the client is slow in rendering the audio, the timeline doesn't really help in maintaining order among objects. Things

```
1  <audio id="M0" begin="0s"                ... />
2  <img id="B0" begin="0s"  dur="43s"... />
3  <img id="I0" begin="0s"  dur="3s" ... />
4  <img id="I1" begin="3s"  dur="4s" ... />
5  <img id="I2" begin="7s"  dur="9s" ... />
6  <img id="I3" begin="16s" dur="9s" ... />
7  <img id="I4" begin="25s" dur="4s" ... />
8  <img id="I5" begin="29s" dur="5s" ... />
9  <img id="I6" begin="34s" dur="9s" ... />
10 <audio id="C0" begin="0s"               ... />
11 <audio id="C1" begin="3s"               ... />
12 <audio id="C2" begin="7s"               ... />
13 <audio id="C3" begin="16s"              ... />
14 <audio id="C4" begin="25s"              ... />
15 <audio id="C5" begin="29s"              ... />
16 <audio id="C6" begin="34s"              ... />
17 <img id="L0" begin="0s"  dur="3s" ... />
18 <img id="L1" begin="3s"  dur="4s" ... />
19 <img id="L2" begin="7s"  dur="9s" ... />
20 <img id="L3" begin="16s" dur="9s" ... />
21 <img id="L4" begin="25s" dur="4s" ... />
22 <img id="L5" begin="29s" dur="5s" ... />
23 <img id="L6" begin="34s" dur="9s" ... />
```

Example 7-1. Time-list translation of the timeline in Figure 7-2

become even more troublesome if we don't know the implicit duration of the audio items when constructing the timeline or if the duration of the object changes over the lifetime of the presentation. (Since the SMIL file doesn't contain the media — it contains a pointer to the media — the timing and the update history of the media object are de-coupled from its use.) Finally, if we add content substitution or interactive timing to the presentation (such as having the follow-on slide begin on a mouse click rather than at a fixed time) the timeline representation loses almost all of its utility.

In order to provide a more realistic framework for Web documents, SMIL is based on a *structured* timing framework in which the structured relationships among objects can be used to define most timing. SMIL encodes its timing relationships by defining a logical timing hierarchy rather than an exact timeline. The hierarchy for Figure 7-1 is shown in Figure 7-3. Here we see a set of green logical parallel nodes (P_0-P_7) and one blue logical sequential node (S_0). The parallel components say: activate the sub-components together as a unit, and the sequential component says: activate the sub-components sequentially. The SMIL textual encoding of the presentation hierarchy is shown in Example 7-2.

While a timeline can state that objects I_2, C_2 and L_2 all start at 7 seconds into the presentation and that they each have a duration of 9 seconds, the SMIL hierarchy can state what is really going on logically:

- that objects C_i, I_i and L_i are to be treated as a logical group that get scheduled together (that is, they begin and end together);
- that the duration of I_i and L_i depend on the duration of C_i,
- that all three objects are to begin after object C_{i-1} — and, by extension, I_{i-1} and L_{i-1} — end.

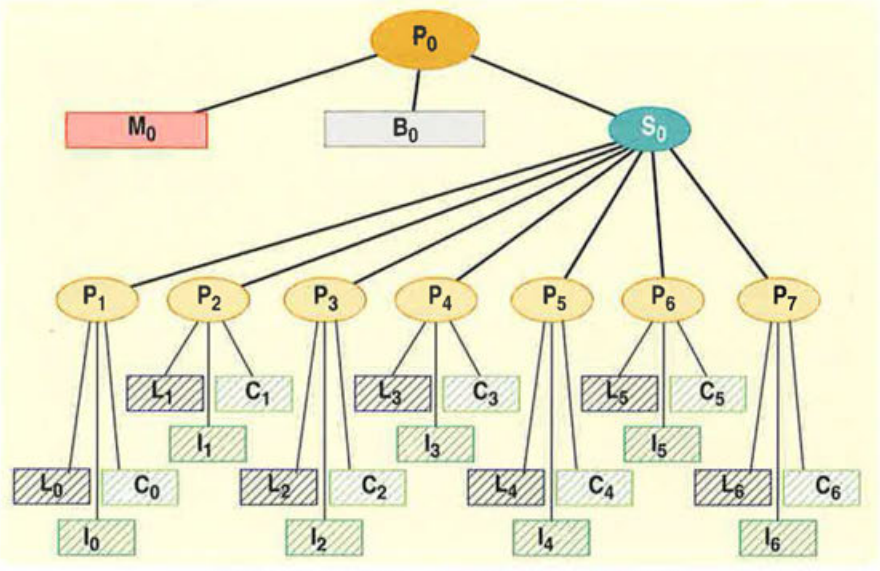


Figure 7-3. The SMIL hierarchy for Figure 7-1.

```
<par id="P0" >
  <audio id="M0"      ... />
  <image id="B0"      ... />
  <seq id="S0" >
    <par id="P1" >
      <img id="I0"    ... />
      <audio id="C0"  ... />
      <img id="L0"    ... />
    </par>
    <par id="P2" >
      <img id="I1"    ... />
      <audio id="C1"  ... />
      <img id="L1"    ... />
    </par>
    ...
      <par id="P7" >
      <img id="I6"    ... />
      <audio id="C6"  ... />
      <img id="L6"    ... />
    </par>
  </seq>
</par>
```

Example 7-2. SMIL structure for Figure 7-1.

Note that none of these relationships depend on the exact duration of any of the objects — you can construct a SMIL file before you know anything about the actual media being used.

A single timeline for one instance of a SMIL specification (that is, for one of the — potentially many — run-time uses of the presentation) can be constructed by combining the structured composition of objects with a model of the execution environment that contains information on the performance of the network connection, the preferences of the user, etc. A timeline based on the implicit media timings alone is not rich enough to model the various structured paths throughout a SMIL presentation.

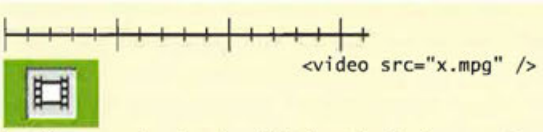
7.1.3 Durations, Time and Timebases

One of the most powerful features of SMIL is a flexible time model in which various aspects of an element's behavior can be determined by the context in which it is being presented. In order to use this model, it is important to understand a number of temporal distinctions and constraints applied by the SMIL model. These include defining the active period of an element and defining the way that delays and relative starting/ending times can be expressed in a document.

Defining the Active Period of an Element

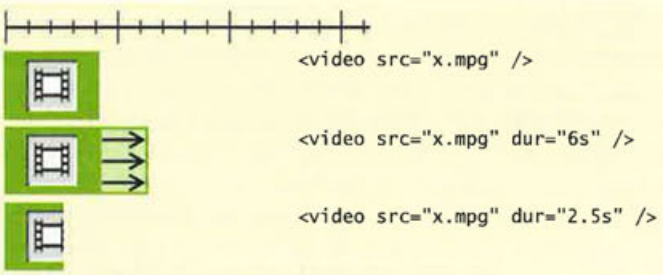
Most media formats require that the duration of all of the component media objects are explicitly defined. SMIL has several attributes that support direct duration definition, but it also provides attributes that allow you to specify or limit several layers of logical object durations. These layered durations, which are illustrated in Figure 7-4, are:

(a) Implicit Duration



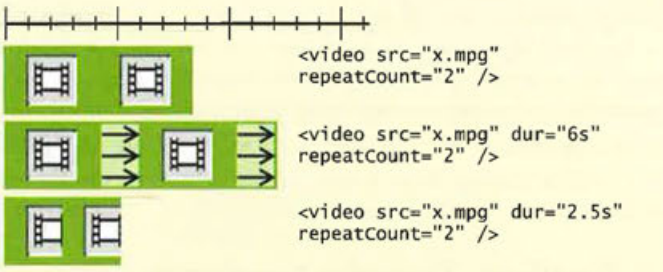
In this example, the *implicit duration* is 4 seconds.

(b) Simple Duration



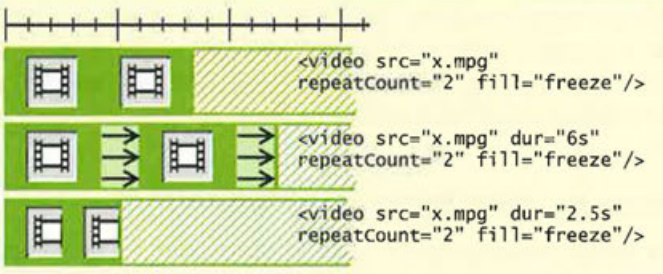
Implicit Duration modified by dur.

(c) Active Duration



Simple duration modified by a repeat count or duration.

(d) Rendered Duration



Active duration plus fill behavior.

Figure 7-4. SMIL durations.

- *Implicit duration*: This is the duration of the media object as defined when the object was created. Most discrete media items have an implicit duration of 0 seconds; some quasi-discrete media — such as animated images — may have a longer duration, but they are modelled as having an implicit duration of 0 seconds. Continuous media objects have a duration that is equal to the temporal length of the object. Many media formats (but not all!) define the implicit duration explicitly in the media encoding. If this encoding is not available, the SMIL player typically will have to scan the entire object to determine its duration. (This can be a time consuming process.) Since a SMIL file does not contain the actual media, the SMIL timing model cannot make very many assumptions about an object's duration. This is why other logical timing durations have been defined within SMIL. Note that the implicit duration is also often called an object's intrinsic or inherent duration. See Figure 7-4(a).
- *Simple duration*: It is possible to modify an object's implicit duration with an explicit duration via SMIL's `dur` attribute. The result of applying an explicit duration (if any) to an object yields its simple duration. (If the implicit duration is not modified by a `dur` attribute, then the implicit and simple durations are the same.) The simple duration of an object may be longer or shorter than the implicit duration. Simple durations can also be defined to have special values that logically limit or stretch the duration of objects; these are the *media* and *indefinite* values, and are considered on page 135. See Figure 7-4(b).
- *Active duration*: SMIL defines a number of attributes that allow an element to be repeated. These attributes modify the element's simple duration, and the resulting repeated duration is called the object's active duration. If an element's simple duration is shorter than its implicit duration, only the first part of the element will be repeated. If the element's simple duration is longer than its implicit duration, the entire element plus the temporal difference between the implicit and simple durations will be repeated. (During this "extra" time, either nothing will be rendered or the final frame/sample of the media object will be rendered: the behavior depends on the media type.) The `end` attribute can be used to define when the active duration ends. If an element does not repeat, its simple and active durations are the same. See Figure 7-4(c).
- *Rendered duration*: The active duration of an element ends after its `dur`/`end` and `repeat` attributes have been applied. This does not mean that an object disappears at the end of its active duration. SMIL provides an attribute to control the persistence of an object after its active duration has ended: the `fill` attribute. If `fill` is set to *freeze*, the element will remain rendered until the end of its parent time container. If `fill` is set to *remove*, the object is removed from the screen as soon as its active duration ends. For discrete media with a `fill="freeze"` attribute, the object will simply be rendered as if its active duration was extended; for visual continuous media, the last

frame or sample of the object will be rendered. (For an example, look at lines 17-19 in Example 7-2 on page 131: images I_6 and L_6 remain visible even though they have an active duration of 0 seconds because the effective default value for `fill` on these objects is *freeze*.)[1]

It is important to understand that each of these durations apply to every temporal element in SMIL. Every element has an implicit, simple, active and rendered duration. Luckily, most of these durations can be managed by the SMIL player, but understanding each of these durations, and their impact on presentation timing, can help you understand why a SMIL player behaves the way it does for your presentation.

Clock Values

Many timing attributes are based on clock values. These values can take several different forms, and they may serve as all or part of an attribute's time value. All clock values represent a relative time, and have meaning only within the context of a time container.

Clock values may be given in four general forms:



- *Full clock values*: These are times represented as a colon-separated list of hours, minutes, seconds and fractions of a second. (If you also need to specify days, months and years, use *wallclock timing* instead.) A full clock value of 5:22:19.94 means 5 hours, 22 minutes, 19 seconds and 940 milliseconds. (The fractional component is optional, all the others are required.) This is a relative time, and has meaning only within the context of a time container's *syncbase*. (Syncbases are described below.)
- *Partial clock values*: These are times represented as a short-hand notation containing minutes, seconds and (optionally) fractions of a second. A clock value of 11:10.95 means 11 minutes, 10 seconds and 950 milliseconds.
- *Timecount values*: These are numbers with an optional type string and an optional fractional component. An integer clock value with no type string (such as '10') implies a timecount in seconds; it is equivalent to '10s'. Allowed type strings are: "h", "min", "s" and "ms".
- *Wallclock values*: These are times represented in three parts: a date field, a time field and an (optional) timezone field. The format for the date field is: YYYY-MM-DD, followed by a required 'T' character. The time field is the same as in the full clock value, above. The timezone offset format, when used, is: {+|-}hh:mm; it specifies an offset from UTC. If a timezone offset is not given, the local time on the player's host is used. An example of wallclock time is: 2004-01-04T10:14:48.12+1:00.

SMIL also allows relative times within media elements to be specified in a time notation that is relevant to that media. This is considered in Chapter 14.

---

1 Our discussion in this chapter on the freeze and remove values for fill considers basic functionality. We discuss the details of fill more completely in Chapter 14: *Advanced SMIL Timing Attributes*.

Syncbases

Except in the special case of wallclock timing, every clock value in SMIL is relative to some other part of the document. The child elements of a parallel container are started relative to the start time of the parent, while the child elements of a sequential container are started relative to the end of their predecessor (except for the first child, which starts at the beginning of its parent). Every element in a SMIL specification has a specific temporal reference point: its *syncbase*. Most elements never have to explicitly specify their syncbase reference since the common SMIL time containers do this by default. Sometimes, however, an element may want to specify a non-default syncbase as its reference object. SMIL supports this functionality using *explicit syncbase timing*. An explicit syncbase is a named element (within the same host document) that has a temporal context, and which is not a child of the element in which it is being referenced. (This is less complex than it reads.) A syncbase timing reference contains a temporal event that is used as the scheduling base for the referencing object. This timing reference can be further modified with a clock value.

In order to fully appreciate the role of syncbase timing, we first need to consider the elements and attributes defined for less complex operations, but to give a taste of what's ahead, consider the SMIL fragment in Example 7-3.

```
1 <smil ...>
2 ...
3       <video id="a" src="a.mpg" />
4 ...
5         <img id="name" begin="a.begin+10s" end="a.end" src="name.png" />
6 ...
7 </smil>
```

Example 7-3. An overview of syncbase timing.

This example contains an element on line 3 that starts an associated video object. In some other part of the document, an image containing a text label is started 10 seconds after the video begins. The label remains visible until the end of the video.

Syncbase timing can be very complex and its use in simple documents is rare. Still, for certain applications, it can be a powerful construct.

7.1.4 Special Timing Values

SMIL defines two special timing values that can be used to specify the temporal context of an element. These are *indefinite* and *media*.

Value: *indefinite*

The indefinite value can be used to indicate that a timing dependency (either a begin/end time or a duration) is to be determined outside of the element in which the indefinite value appears. While many SMIL authors (and some SMIL implementers!) assume that indefinite is synonymous with "forever," it is really synonymous with "I have no local idea ...". As we will see later in this

chapter, an attribute assignment of `dur="indefinite"` simply means that some other part of the SMIL specification — either the parent time container or some other element — will determine the element's duration. Only if no other part of the document constrains the duration will an indefinite value result in an unlimited duration, but even then, a value of indefinite can never result in a presentation that a user (or user agent) cannot end.

Value: *media*

The attribute value *media* can be applied to obtain the desired timing value directly from the associated media item. Although it is not often required to explicitly use the media value (mostly because this will be the default behavior), it is sometimes useful to set a duration or end time to *media* when you want to exert explicit control in complex timing situations. We will give examples of this value for the attributes `begin`, `end`, `dur`, `max`, `repeatCount` and `repeatDur` when these are discussed later in this chapter.

7.1.5 Interactive Timing and Events



In the normal course of processing, the activation hierarchy of a SMIL 2.0 document determines the rendering of document elements. The user can influence the elements selected by using SMIL 2.0 events. The event architecture allows document components that are waiting to be activated or terminated to actually start or stop. There are several uses of events allowed in SMIL 2.0, but perhaps the most important new semantic in the language is the combination of events and the begin/end attributes. In further combination with the `<excl>` element, events provide a very powerful mechanism for conditional content activation. Interactive timing is treated in Chapter 13: *Advanced SMIL Timing Behavior and Control.*

SMIL 2.0 also supports a rich hyperlinking architecture. Unlike links in XHTML, the fundamental concept of the SMIL link is that it models a temporal seek in a presentation. Rather than simply activating a target element, the target play state that is activated is identical to the state that the presentation would have been in if the target point had been arrived at 'naturally'. (One exception is that all event-based activation is ignored.) This means that all nodes temporally between the source and destination of the link need to be evaluated to see if they would have contributed to the final target play state. The temporal seeking and activation facility allows very polished presentation construction — but its implementation in the player is not for the faint-hearted! SMIL's linking facilities is the subject of Chapter 8: *Basic Linking*.

7.2 Basic SMIL Timing Elements

SMIL 2.0 extends the elements and attributes available for creating XML-structured multimedia presentations defined for SMIL 1.0. At the same time, SMIL 2.0 introduced a profile-based architecture in which various timing (and other

SMIL) components are made available to non-SMIL host languages. This section provides an overview of each of the elements and attributes available in all profiles that include any SMIL timing. They are defined in the SMIL *BasicTimeContainers* module. *BasicTimeContainers* is a stand-alone module that can be easily integrated into other languages.

A SMIL presentation is a structured composition of autonomous media objects. The high-level composition is achieved using three timing elements: the <par>, the <seq> and the <excl>. Of these, the <excl> is an advanced timing element that is not supported in all profiles. Therefore, we discuss it separately in Chapter 13: *Advanced SMIL Timing Behavior and Control*.

		<seq>	<par>	Profiles				
attributes	general timing			L	M	B	1	H
	endsync							
	media description							
	region							
	test							
	core							
parents	<body>			L	M	B	1	H
	<priorityClass>							
parents & children	<seq>, <par>			L	M	B	1	H
	<excl>							
	<switch>				M	B	1	
	<a>							
children	media object			L	M	B	1	H
	animation							

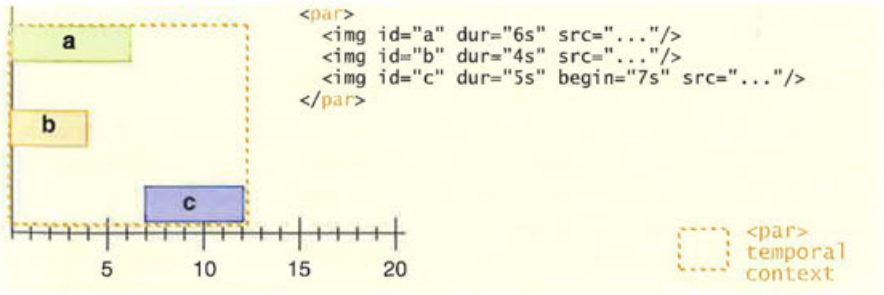
Table 7-1. Structure of the basic time container elements.

7.2.1 Basic Time Containers

The <par> and the <seq> elements form the core of SMIL timing and are discussed below.

Element: <par>

The most general of SMIL's timing containers is the <par>. The <par> defines a local time container that can be used to activate one or more child elements. The children of a <par> container are all rendered in "parallel". In terms of SMIL's timing model, this does not mean that they get rendered at the same time, but that they share a common syncbase defined by the <par> container. Any or all of the children may be active at any time that the parent <par> is active. The structural context of the <par> element is illustrated in Table 7-1.




```
<par>
  <img id="a" dur="6s" src="..."/>
  <img id="b" dur="4s" src="..."/>
  <img id="c" dur="5s" begin="7s" src="..."/>
</par>
```

Figure 7-5. SMIL's <par> timing element.

The basic timing structure of the <par> is illustrated in Figure 7-5. The default syncbase of the <par> is the beginning of the element. That is, by default all children of a <par> element start when the <par> itself starts. (This is illustrated by nodes "a" and "b".) As discussed below, timing attributes can be used to specify other begin times. By default, the <par> ends when the last child ends, although the exact ending behavior of the <par> is determined by the endsync attribute.

Element: <seq>

A relatively unique timing container is the <seq> element. The children of a <seq> are rendered in such a way that a successor child never can begin before its predecessor child completes. In other words, the syncbase of each child is the end of the active duration of its predecessor. A successor element may have an additional start delay, but this delay cannot resolve to be negative in relation to the end time of its predecessor. The <seq> ends when its last child has ended. Unlike the <par>, the <seq> does not support the endsync attribute: since there is only one child active at a time in a <seq>, there is no need to select among children to determine the container's end. The structural context of the <seq> element is illustrated in Table 7-1.

The basic timing structure of the <seq> is illustrated in Figure 7-6. The <seq> is especially useful when describing timing in a non-deterministic environment; by specifying that a set of elements logically follow one another, then a delay in one element can be easily passed to its successors.

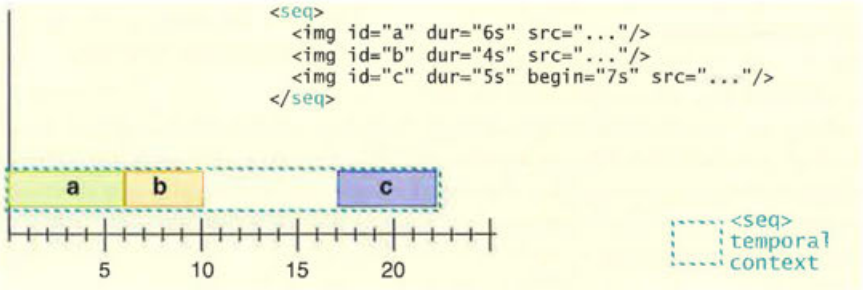


Figure 7-6. SMIL's <seq> timing element.

7.2.2 Nested Composition of Timing Elements

The basic time containers can be nested in a presentation hierarchy. That is, any child of either a <par> or <seq> (or <excl>) can be a simple media object or an embedded time container. As in SMIL 1.0, the hierarchy can represent relatively static presentation timing. The introduction of event-based activation/termination in SMIL 2.0 also allows a dynamic activation path to be defined. Most of the children of a time container will influence the timing behavior of that container, but some children — such as the <a> element, or the <switch> — are

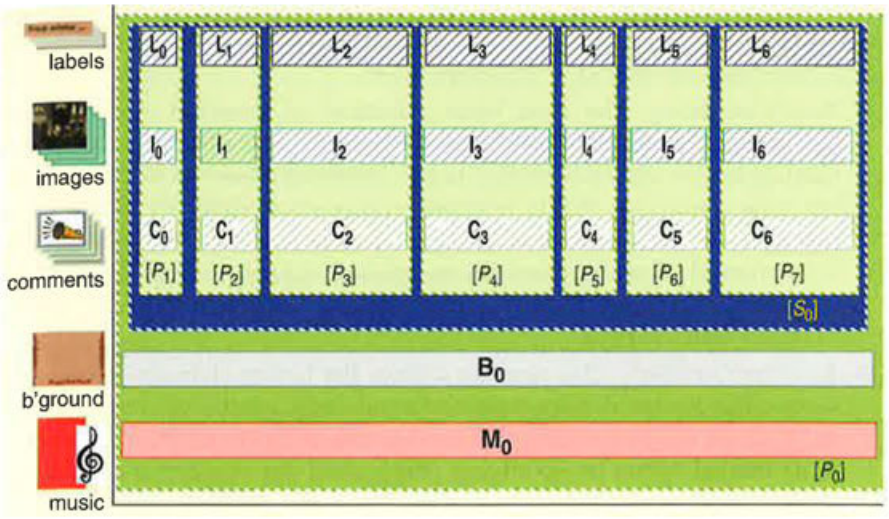


Figure 7-7. A structured timeline representation of the presentation in Example 7-2.

timing transparent. This means that they do not contribute to the determination of the container's duration.

Figure 7-7 illustrates the temporal composition of the example presentation in Figure 7-1 on page 126, based on the SMIL encoding in Example 7-2 on page 131. Note that the <par> and <seq> elements are nested as green and blue boxes, respectively. (The node labels of the structure containers are placed in brackets at the bottom of each container.) This type of structure presentation of multimedia documents — in which the structural relationships between objects are highlighted rather than the (often unknown) timing characteristics — was pioneered by the Oratrix GRiNS authoring environment.

7.3 Basic SMIL Timing Attributes

The *BasicTimeContainer* module of SMIL 2.0 defines the synchronization relationships available with temporal composition elements. The timing provided by the <par> and <seq> elements can be further refined by using a wealth of timing control attributes. Unlike other SMIL modules, where functionality is grouped in relatively broadly-based collections of elements and attributes, the partitioning used by SMIL timing is fine-grained. Functionality is spread across 19 timing modules, many of which contribute no more than a few values for attributes defined in other modules. The reason for such fine-grained partitioning is that complex, advanced timing is usually not needed by all languages that may want to reuse SMIL timing. By isolating functionality in collections of modules, language designers can flexibly determine the exact set of timing features that they wish to include (and specifically exclude!) in their adoption of

SMIL timing. This 'count me out' philosophy didn't hurt the SMIL standardization process, either.

The modules considered in this chapter are:

- *BasicInlineTiming*: The most basic collection of temporal attributes are grouped in the *BasicInlineTiming* module. These attributes define when an element begins and ends relative to the default syncbase defined by its parent time containers. While temporal composition elements define most timing based on logical structure, the inline timing attributes rely heavily on numerical values to determine an instant in a presentation. This module is supported by the SMIL Language profile, SMIL Basic, 3GPP SMIL, XHTML+SMIL and SVG.
- *BasicTimeContainers*: This module defines the timing elements `<par>` and `<seq>`, but it also defines two important SMIL attributes: `endsync` and `fill`. These attributes allow the structural timing defined by the elements to be refined, either by specifying which child out of a group of children controls the end of a parent container (this is the `endsync` attribute) or by allowing the perceived duration of an object — or collection of objects — to extend past its active duration (the `fill` attribute). All profiles and languages that support SMIL timing semantics will support these attributes.



- *MinMaxTiming*: Many SMIL timing attributes allow timing to be specified or modified within the context of an element's structural place in the document hierarchy. The *MinMaxTiming* module provides two attributes that provide control over the duration of an element outside the context of its temporal structure: the duration can be bound to a minimum duration or a maximum duration. The *MinMaxTiming* module is supported by the SMIL Language profile, SMIL Basic and XHTML+SMIL.
- *MultiArcTiming*: This module allows the begin and end attributes to accept a list of begin or end times. This can be useful for objects that are to be repeated, but it is especially useful for handling event-based timing.
- *SyncbaseTiming*: The *SyncbaseTiming* module allows the syncbase of an element to be explicitly defined relative to the beginning or the end of some other element in the document. (The basic time containers provide short-hand notations for syncbase timing; this module defines explicit control from elements that are not necessarily co-located.) The SMIL Language profile and XHTML+SMIL support this module
- *RepeatTiming*: The *RepeatTiming* module defines attributes that control the way in which elements are repeated. Where SMIL 1.0 had a unified — and often misunderstood — attribute for repeating elements, SMIL 2.0 introduces two attributes: `repeatCount` and `repeatDur`. This module is supported by the SMIL Language profile, SMIL Basic and XHTML+SMIL.

Given the high degree of interaction among the SMIL timing modules in defining and assigning attributes and attribute values, this chapter will not take a module-based approach for discussing SMIL timing. Instead, we look at

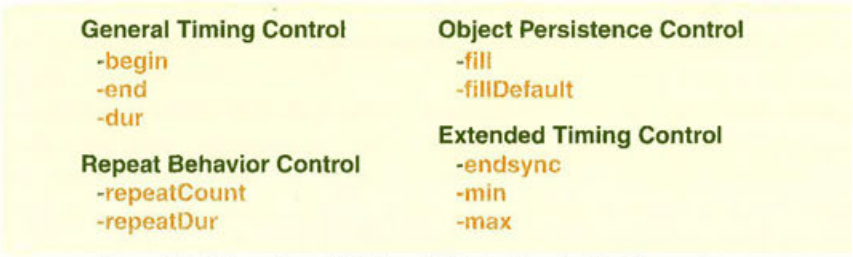


Figure 7-8. Overview of timing attributes treated in this section.

timing attributes from a user perspective. Figure 7-8 gives an overview of the attributes to be discussed in this section and the classification of attributes that this chapter uses. Within each group, we discuss basic and advanced attribute use. The more esoteric uses of SMIL's basic timing attributes are discussed in Chapter 14: *Advanced SMIL Timing Attributes*.

7.3.1 General Timing Control Attributes

The general timing control attributes are: `begin`, `dur` and `end`. The `begin` attribute controls when a particular element starts, the `dur` attribute controls the element's simple duration and the `end` attribute controls the end of the active duration. (In general, `begin+dur=end`, but this is not always true.) The collection of general timing control attributes and values for the basic timing elements are shown in Table 7-2.

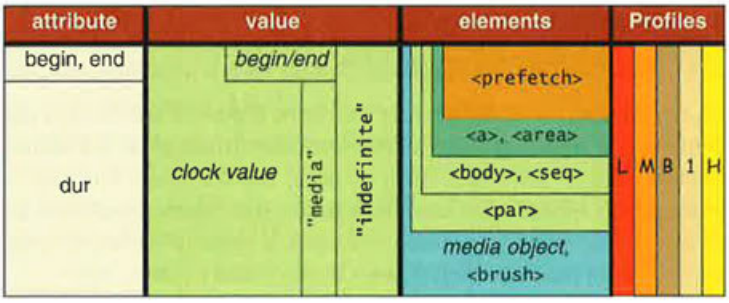


Table 7-2. General timing control attributes.

All three of these attributes retain their behavior from SMIL 1.0. The principal SMIL 2.0 extension to `begin` and `end` is that these attributes now support a list of begin/end times instead of a single begin/end value. This is not particularly useful for simple SMIL applications, but it is very useful if a combination of SMIL's scheduled and event-based timing is used. (More on this in Chapter 13: *Advanced SMIL Timing Behavior and Control*.) Multiple begin/end times can also be useful if objects are to be restarted, as in animations.

Attribute: begin

Each of SMIL's time containers defines a default begin time for its children. The default for <par> is the start of that <par>, while the default for a <seq> depends on the lexical order of its children: the first child starts at the start of the <seq> and each successive child starts after the preceding child has ended. SMIL allows this default behavior to be overridden by the begin attribute. For simple uses, the begin attribute will contain a clock value. Usually, this clock value contains a non-zero positive temporal offset, such as:

```
begin="10"
begin="10s"
begin="0:10.0"
begin="0:00:10.0"
```

All of these values represent a begin delay of 10 seconds. Given a choice, you should probably use the second version, as it gives a reasonable balance between clarity and brevity.

Technically, a specification of:

```
begin="10s"
```



defines a 10 second delay from the implied syncbase of the element containing the begin attribute. (This is usually either the start of a parent <par> or the end of a predecessor child in a <seq>.)

Defining explicit syncbases. SMIL allows the syncbase used in the presentation to be explicitly defined for <par> elements by specifying an ID of the syncbase, and specifying a timing qualification of that syncbase (either begin or end). The following element specifies that an audio object starts 4 seconds after the end of an element named *snarf*:

```
<audio id="bazinga" begin="snarf.end+10s" ... />
```

In SMIL 1.0, the element with ID of *snarf* had to be a peer of the *bazinga* element. SMIL 2.0 relaxes this requirement: the element *snarf* must be in the same document as *bazinga*, and it can't be a child of *snarf*, but there are no other restrictions. (The situation where *snarf* and *bazinga* are not peers is referred to as an *out-of-scope* syncbase.) Note that SMIL 1.0 used a slightly different syntax for specifying syncbases; this format has been depreciated in SMIL 2.0.

In the SMIL Language profile (including SMIL Basic), explicit syncbases are rarely used in simple documents, but for other host languages (like those supporting the XHTML+SMIL profile), the use of explicit syncbases can be handy. This is because the host document provides no common temporal basis among its objects, so that you often need to define a common syncbase explicitly when coordinating content specified in different parts of the same document.

Specifying multiple begin times. In a SMIL 1.0 document, each element could specify at most one begin time. SMIL 2.0 allows multiple begin times to be defined. In general, an object will start as soon as any of its begin conditions are satisfied. It will also restart if a new begin condition is satisfied! For example, the following object will begin after a delay of 10 seconds from its default sync-

base *and* 3 seconds after the start of *snarf*, unless it has had restrictions placed on its repeat behavior — it will not just start when the first begin condition in the list gets resolved:

```
<video begin="10s; snarf.begin+3s" ... />
```

The use of multiple begin times has a number of features and restrictions that are discussed in Chapter 14: *Advanced SMIL Timing Attributes*; these include ways of defining interactive starts of an object and restricting an object's restart behavior. For most simple SMIL documents containing only a single begin time, these restrictions are not important.

The* `begin` *attribute and the* `<seq>` *element. There is one restriction on the use of the `begin` attribute that *is* important for all SMIL documents: a `<seq>` time container may only have a single begin attribute value, and that value must contain a non-negative clock value. Multiple begin time lists are not allowed in a `<seq>`, and neither is explicit syncbase timing. In the following set of statements, line 5 is not valid SMIL, but lines 6 through 8 are valid and provide the same functionality as was intended in line 5:

```
1  <par>
2    <audio id="snarf" ... />
3    <seq>
4      <image begin="2s" dur="5s" ... />
5      <img begin="snarf.end+2s" ... />    <!-- this is invalid SMIL! -->
6      <par>                               <!-- this is valid SMIL -->
7        <img begin="snarf.end+2s" ... />
8      </par>
9    </seq>
10 </par>
```

Line 5 is not valid because a child of a `<seq>` may not contain a syncbase offset. The `<par>` on line 6 begins a new temporal content: the `<par>` starts when the preceding image ends; it then delays the start of its child image until 2 seconds after the end of the *snarf* object.

Invalid begin times. An interesting final point is: what does a SMIL player do if you give it an improper begin time specification? Some of the rules for processing bad values are profile dependent, and others depend on the kind of begin attribute values being used. If you were to give this example to a generic SMIL player, it should refuse to play the entire document because of an error with the value of a `begin` attribute in a `<seq>`.

Attribute: dur

The `dur` attribute establishes the simple duration of an element. (Recall from Figure 7-4 on page 132 that the simple duration is the duration without repeat behavior.) When used with a media object, it defines how much of that media object will be used. When used with a time container, it defines a limit on the simple duration of that container. If the defined duration is longer than the time container or media object, temporal and rendering padding will result.

For all media object references, the default value for `dur` is media. For time containers, the default value for `dur` is the simple duration of the children of

that container. When a value is explicitly specified, a non-zero, positive clock value is usually used.

***The* dur *attribute and* fill.** Once the simple duration specified by dur ends, the associated element will either repeat (if repeat behavior has been specified) or become a candidate for removal from the player's list of rendered objects. Whether it is actually removed depends on the value of the fill attribute, as discussed on page 147.

The* dur *attribute and indefinite timing. It is also possible to assign the value of *indefinite* to dur; this means that the simple duration is explicitly defined to be indefinite, and that the active duration and the element's end moment will be taken from the temporal context of the presentation.

Examples. The following two SMIL statements result in an element with a simple duration of 0 seconds:

```
<img src="image.png" ... />
<img src="image.png" dur="media" ... />
```

The following statement is actually invalid SMIL, but many SMIL players also accept it as a means to specify an element with a simple duration of 0 seconds:

```
<img src="image.png" dur="0" ... />
```

(This is invalid because the specification explicitly says that the clock value of the dur attribute must be greater than 0.)

The following (short) presentation illustrates the behavior of the *indefinite* duration value:

```
<smil ...>
  <body>
    <img src="image.png" dur="indefinite" ... />
  </body>
</smil>
```

The image will be rendered and the simple duration will be set to indefinite. This will result in the active duration being set to the further timing context of the presentation. Since no other timing context is explicitly provided, the image will be rendered "forever".

If we extend the presentation above a bit, we can see some additional behavior of the SMIL timing model:

```
1 <smil ...>
2   <body>
3     <par>
4       <img src="image.png" dur="indefinite" ... />
5       <video id="V1" src="ten-sec-video.mpg" ... />
6     </par>
7   </body>
8 </smil>
```

In this example, the <par> on lines 3 through 6 renders an image and a 10 second video. The simple duration of the image is indefinite and the simple duration of the video is 10 seconds. After 10 seconds, the simple duration of the

video ends, but the indefinite duration of the image means that unless something else in the presentation explicitly limits the duration, the presentation will not end. (The default behavior of the video is that the last frame will be shown during the entire period after 10 seconds.)

We now replace line 5 with the following:

```
5    <video id="V1" src="ten-sec-video.mpg" dur="10s" ... />
```

The result is that since the video now has an explicit duration, it no longer has default fill semantics, and the last frame is not shown for the remainder of the presentation. (Only the image is shown.)

We next replace line 3 with the following:

```
3   <par endsync="V1"/>
```

The result is that even though the image on line 4 has an indefinite duration, the <par> will end when the video on line 5 ends. The timing context for the indefinite is determined by the context of the parent.

This section has illustrated the behavior of dur on media objects, but all of the examples also apply when dur is defined for time containers such as <par> and <seq>. The only real difference is that an assignment of dur="media" is not allowed for time containers; in this case, the duration assignment will be ignored.



Attribute: end

A SMIL element ends at the end of its active duration. This is determined by either the element's content or the parent time container. An element can also specify a non-default end time via the end attribute. The end attribute is very similar to the begin attribute. Rather than repeating all of the begin's properties here, please make sure you've read the section on begin above before continuing with this section.

The end attribute provides explicit control over the end of the active duration. As with begin, the end can specify a single clock value or an explicit syncbase (with optional clock value offset), or a list of values. (There are also other values allowed; these are covered in Chapter 14: *Advanced SMIL Timing Attributes*.) Unlike the begin attribute, there is no restriction on the use of end with <seq> elements. Note that even if multiple end times are specified, the associated element will end only once — this is when the first end condition is met.

Examples. The following fragment gives examples of the use of the end attribute:

```
1  <video src="one-hour-video.mpg" end="25s" ... />
2  <video src="one-hour-video.mpg" end="1:00:15" ... />
3  <video src="one-hour-video.mpg" end="snarf.end" ... />
4  <video src="one-hour-video.mpg"end="35s; snarf.end+5s" ... />
```

In line 1, the one-hour video is cut short after 25 seconds. In line 2, the active duration of the one-hour video is extended by 15 seconds. In line 3, the video element ends when the element *snarf* ends (even if this is after 3 hours!). In line 4, the video element ends either after 35 seconds or 5 seconds after *snarf* ends, whichever comes first.

The examples above do not show a time context for each video reference. It is important to note that, unless an explicit syncbase is defined, the end attribute modifies the end time relative to the default syncbase of the element, not the duration of the media object.

Consider the SMIL fragment in Example 7-4. The default syncbase of the children of the <seq> in lines 3 through 7 is the end of the predecessor child. This means that:

- the video element on line 4 ends 25 seconds after the start of its syncbase (which is the beginning of the <seq>)
- the video element on line 5 ends one hour and 15 seconds after the start of its syncbase (which is the end of the element on line 4)
- the video element on line 6 ends one hour and 15 seconds after the start of its syncbase (which is the end of the element on line 5) — this is 5 seconds before the end of the video media object, because that object started being rendered 20 seconds after the start of its syncbase.

Make sure you understand why the end of the video element on line 6 occurs before the end of the video object: the end attribute does not modify the duration of the media object, but it modifies the end of the element containing the video object in relation to that element's syncbase.

Now, look at the <par> in lines 9 through 13 of Example 7-4. The start of the <par> on line 9 defines the default syncbase for the children of this <par>. This means that:

- the video element on lines 10 ends 25 seconds after the start of its syncbase (which is the beginning of the <par>)
- the video element on line 11 ends one hour and 15 seconds after the start of its syncbase (which is also the beginning of the <par>)

```
1  <smil ...>
2    ...
3    <seq>
4      <video src="one-hour-video1.mpg" end="25s" ... />
5      <video src="one-hour-video2.mpg" end="1:00:15" ... />
6      <video src="one-hour-video3.mpg" begin="20s" end="1:00:15" ... />
7    <seq>
8    ...
9    <par>
10     <video src="one-hour-video1.mpg" end="25s" ... />
11     <video src="one-hour-video2.mpg" end="60.25m" ... />
12     <video src="one-hour-video3.mpg" begin="20s" end="1.004167h" ... />
13   <par>
14   ...
15 </smil>
```

Example 7-4. The end attribute in relation to its syncbase.

- the video element on line 12 ends one hour and 15 seconds after the start of its syncbase (which, once again, is the beginning of the <par> on line 9) — as with the case on line 6, this is 5 seconds before the end of the video media object, because that object started being rendered 20 seconds after the start of its syncbase.

These examples all relate to a single video object, but the same syncbase-based evaluation of end times also holds true for any time container containing an explicit end attribute.

Combined Use of begin, dur *and* end

It is possible to combine the use of the begin, end and dur attributes on an element. Combining begin and dur usually results in obvious behavior, since they both relate to the simple duration of an object. Since the end attribute overrides the default end of the active (not simple) duration, the resulting behavior is not always obvious. The less obvious details are discussed when we look at advanced timing properties.

7.3.2 Object Persistence Attributes

Two attributes control the persistence of objects after they have completed their active duration: fill and fillDefault. In this chapter, we consider simple uses of fill behavior. More complex uses of these attributes are discussed in Chapter 14: *Advanced SMIL Timing Attributes*. The allowed values and related elements for fill are shown in Table 7-3.

attribute	value	Elements	Profiles
fill	"freeze", "remove"	<par>, <seq>	L M B 1 H
		media object, <brush>	

Table 7-3. Basic object persistence attribute fill.

Attribute: fill

SMIL allows an object to remain visible after the end of its active duration by specifying a fill attribute. If fill is set to *freeze*, the object will remain visible until the end of its parent time container. If it is set to *remove*, the object is removed as soon as its active duration ends. The default for visual media is effectively *freeze*; fill has no meaning for audio media. There are some special cases for fill that allow the visibility of an object to extend past the active duration of its parent (such as when transitions exist). The value of fill also determines how long an object remains 'clickable' for linking and interaction. These cases are considered in Chapter 14: *Advanced SMIL Timing Attributes*.

If the fill attribute is applied to a time container, then the rendering state of all of the children (and descendants of the children) are set to their respective fill behavior at the moment the active duration of the container ends.

```
1 <smil ... >
2 <...>
3  <par id="outer">
4   <img id="label" src="label.png" dur="20s" />
5   <par id="inner" dur="15s">
6    <img   id="picture" src="image.png" dur="5s" fill="freeze" ... />
7    <video id="lady"    src="ten-sec-video.mpg" fill="remove" ... />
8   </par>
9  </par>
10 ...
11 </smil>
```

Example 7-5. The `fill` attribute.

The default `fill` behavior is *auto*. Determining the effective `fill` behavior can be complex, so we defer a complete discussion to Chapter 14: *Advanced SMIL Timing Attributes*. For most simple cases, we can use the rule that if an element explicitly defines a simple duration using `dur`, the effective default behavior of `fill` is *remove*. Otherwise, the simple duration is determined by the implicit duration of the element and the effective default `fill` behavior is *remove*.

Figure 7-5 illustrates `fill` behavior. The *outer* <par> element has an effective default `fill` behavior of *freeze*. Thus, when its children end, they will remain visible. The image *label* on line 4 has the default `fill` behavior of *remove* because of the `dur`. The *inner* <par> element has an effective default `fill` behavior of *remove*. The image *picture* has an explicit `fill` behavior of *freeze*, and the video *lady* has explicit `fill` behavior of *remove*. When the presentation starts, all media is active. After 5 seconds, *picture* ends its active duration but it stays visible because of the `fill="freeze"`. The *lady* video ends at 10 seconds and the last frame is immediately removed. The *inner* <par> ends at 15 seconds, and removes all of its children (even the image *picture*). After 20 seconds the image *label* ends, as does the <par> *outer*, and the screen is empty.[2]

Now, assume we change line 5 to add an explicit `fill="freeze"`:

```
<par id="inner" dur="15s" fill="freeze" >
```

In this case, the video *lady* is still removed after 10 seconds, but the image *picture* is displayed for the full 20 seconds (until the end of *outer*'s active duration.)

The `fill` attribute and transitions. When used with transitions, the `fill` attribute can also accept advanced values that can extend the simple duration (and the active duration) beyond the temporal scope of the parent time container. See Chapter 10: *Transition Effects* for details.

7.3.3 Extended Timing Control Attributes

SMIL 2.0 has two new attributes that provide extra duration control: `min` and `max`. These attributes can be used to define a lower or upper bound on the active duration of the containing element, regardless of that element's other

2 Note that some SMIL players keep the final state of the presentation rendered — this is a player implementation issue and has no relation to the presentation's composite fill behavior.

timing characteristics. As in SMIL 1.0, SMIL 2.0 also supports the ending of a time container via the endsync attribute. These three attributes, shown in Table 7-4, are discussed in this section.

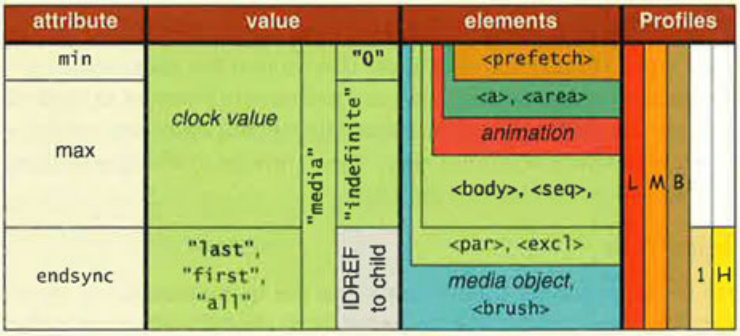


Table 7-4. Extended timing attributes

Attribute: min

The min attribute can be used to specify an explicit minimum active duration on an element. It accepts either a clock value or "media" as values. The default value of min is *"0"*, which is the same as saying that the value is unconstrained. The value *media* may only be used on media objects; its use says: the minimum duration is the implicit value of the media object. If used together with the max attribute, min must be less than or equal to the value specified for max.

The following statements give examples of the min attribute:

```
1 <video src="one-hour-video.mpg" min="25s" ... />
2 <video src="one-hour-video.mpg" min="0" ... />
3 <video src="one-hour-video.mpg" min="media" ... />
```

Statement 1 constrains the minimum active duration of the video element to 25 seconds. Line 2 illustrates default min behavior. Line 3 says that the minimum duration of the video element is the one-hour duration of the media object.

While syncbase timing cannot be used as a value for the min attribute, min can be used in conjunction with dur and end to produce interesting and occasionally useful results. Consider the following statement:

```
<video id="snarf" src="one-hour-video.mpg" min="25s"
      end="bazinga.end" ... />
```

If *bazinga* ends 15 seconds after the start of *snarf*, the duration of *snarf* will be extended for 10 seconds (to 25 seconds).

The following fragment shows the impact of min on a <par> with an explicit duration:

```
<par dur="10s" min="20s" >
  <video src="one-hour-video.mpg" ... />
  <img src="label.png" ... />
</par>
```

Without timing control, the video element would be active for an hour, and the label would be active for 0 seconds, and then be rendered for the next 60 minutes because of its default `fill="freeze"` behavior. The `min` attribute on the <par> says that its active duration is 20 seconds. The `dur` attribute on the <par> constrains the simple duration of the <par> to be 10 seconds. So, what really happens? After 10 seconds, the simple duration of the <par> ends, so the video and image have their timing context terminated. The `min` extends the active duration of the <par>, but since the simple duration has ended, only the default fill behavior is rendered: since this is *remove*, nothing is rendered for the 10 additional seconds of active duration.

Attribute: max

The `max` attribute can be used to constrain the maximum active duration of an element. It is similar to the min, except that an explicit value of indefinite can be specified. (This is also the default.)

The behavior of the max attribute is predictable, although finding relevant use cases can be a challenge. For example, consider the following statements:

```
<video id="a" src="one-hour-video.mpg" max="30s" end="bazinga.end" ... />
<video id="b" src="one-hour-video.mpg" end="30s; bazinga.end" ... />
```



The video element *a* will play for a maximum of 30 seconds or until the element *bazinga* ends (if this is earlier). The same behavior applies to video element *b*.

Attribute: endsync

Where `min` and `max` provide clock-value based constraints on element timing, SMIL provides the `endsync` attribute to provide logical control on timed objects. As in SMIL 1.0, if a <par> has multiple children, it can specify that the entire <par> ends when the first child ends, when the last child ends or when a named child ends. This attribute can be assigned to the <par>, <excl>, <brush>, and media object elements. (The use with <brush> and media objects is typically limited to animation.)

The following fragment illustrates `endsync` behavior:

```
<par endsync="snarf" >
  <video id="bazinga" src="video.mpg" ... />
  <audio id="snarf"   src="audio.mp3" ... />
</par>
```

The <par> ends when the *snarf* ends; if this is later than the end of *bazinga*, the last frame of *bazinga* will be frozen. The default value of `endsync` is *last*.

An element with the assignment `endsync="all"` waits for all of its children to play to completion at least once, regardless of whether they have scheduled or interactive begin times.

The behavior of the `endsync` attribute is intuitive, except when combined with the `dur` and/or `end` attributes. This is because having both an `endsync` and a `dur` or `end` will cause conflicting definitions of element duration. In order to resolve these conflicts, the following rules are applied by a SMIL player:

1) If dur and endsync are specified on the same element, the endsync is ignored. Thus, in the following example, the duration of *snarf* is 20 seconds:

```
<par id="snarf" dur="20s" endsync="b" >
   <video id="a" src="video.mpg" ... />
   <audio id="b" src="audio.mp3" ... />
</par>
```

2) If end and endsync are specified on the same element, the endsync is ignored. Thus, in the following fragment, the duration of *bazinga* ends when *snarf* ends:

```
<par id="bazinga" end="snarf.end" endsync="b" >
   <video id="a" src="video.mpg" ... />
   <audio id="b" src="audio.mp3" ... />
</par>
```

There are also some special cases of endsync behavior when used with repeated elements. We consider these below.

7.3.4 Repeating Objects and Sub-Structures

The default behavior of all elements is that they play once, for either an implicit or explicit duration. The SMIL 2.0 repeatCount attribute allows an iteration factor to be defined that acts as a multiplier of the object's simple duration. (The resulting duration is the active duration.) A special value *indefinite* is used to specify that an element is to be repeated continually until its (parent) timing context ends. The repeatDur attribute defines a duration for all of the repeated iterations. The repeat timing control attributes and values are shown in Table 7-5.

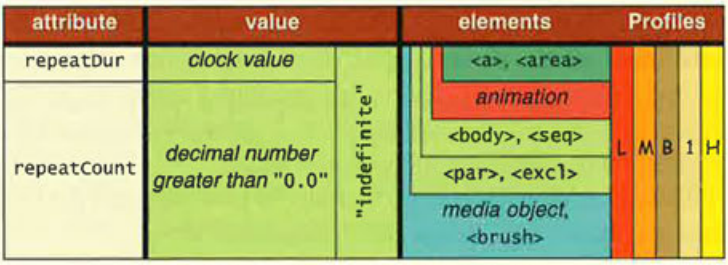


Table 7-5. Repeat control attributes.

Attribute: repeatCount

The repeatCount attribute makes the element's content sub-presentation or media object to repeat the stated number of times. Its value can be either a number or *indefinite*. A numeric value indicates the media object plays that number of times. Fractional values, specified with decimal points, can be used as well.

The following statement specifies that *snarf* is to be repeated 4.5 times:

```
<audio id="snarf"   src="audio.mp3"  repeatCount="4.5" ... />
```

If the associated audio file was 4 seconds long (its implicit duration), then the active duration of snarf is 18 seconds.

A repeat count may also be applied to time containers. As is shown in the following fragment, the simple duration of the container is repeated for the number of iterations specified by the `repeatCount`.

```
<par repeatCount="3" >
   <video id="bazinga" src="video.mpg" ... />
   <audio id="snarf"   src="audio.mp3" ... />
</par>
```

In this example, the active duration will depend on the implicit durations of the media items; the simple duration will be the maximum of *bazinga* and *snarf*. The active duration will be the simple duration times three.

A `repeatCount` is typically only useful on media with implicit timing, or on media with an explicit duration. Consider the following three statements:

```
1 <img id="bazinga" src="picture.png" repeatCount="3" ... />
2 <img id="snarf"   src="picture.png" dur="3s" repeatCount="3" ... />
3 <img id="rex"     src="picture.png" dur="indefinite" repeatCount="3".../>
```



The element *bazinga* on line 1 has an implicit duration of 0 seconds; repeating the element doesn't change this. The element *snarf* on line 2 has a defined simple duration of 3 seconds; the active duration after the repeat is 9 seconds. The element *rex* on line 3 has an indefinite duration; as with a 0 duration, a repeat count doesn't alter the active duration.

If the repeat count is set to be *indefinite*, then the playback repeats until the element is stopped by its temporal context.

Attribute: repeatDur

The `repeatDur` attribute is similar in most respects to `repeatCount`, except that a duration is given instead of a multiplier. It states that the element is to repeat its playback until the specified duration has passed.

The following statement specifies that *snarf* is to be repeated for 15 seconds:

```
<audio id="snarf"   src="audio.mp3"  repeatDur="15" ... />
```

The actual number of time that the audio object is repeated depends on its duration. If the audio object's implicit duration is 3 seconds, it will repeat 5 times. If its implicit duration is 15 seconds, it will be played once. If its implicit duration is 25 seconds, only the first 15 seconds will be played.

As with all elements, a timing constraint on a parent container will limit the active duration of the children. As a result, in the following fragment, the audio object will be rendered for 25 seconds:

```
<par dur="25s">
  <audio id="snarf"   src="audio.mp3"  repeatDur="100s" ... />
</par>
```

Combing Repeat Behavior with Other Timing Attributes

It is possible to combine either `repeatCount` or `repeatDur` with other timing attributes. The resulting timing usually results in predictable behavior for simple cases, but sometimes the differences in manipulating the simple, active and rendered durations within one element can lead to SMIL statements that are not always easy to understand.

The best advice that we can give is to make sure that you understand the differences between the various types of SMIL durations. This becomes vital if you work with interactive behavior in SMIL, or if you use multiple begin times on objects. If you plan to use SMIL Animation — or if you plan to make use of the SMIL repeat attribute — then a complete understanding of the SMIL timing model is essential. We suggest that you read the model section of this chapter and Chapter 13: *Advanced SMIL Timing Behavior and Control* very carefully. As a last resort, the SMIL specification can be consulted: there you will find 156 pages of information on the timing model alone.

7.3.5 Advanced Timing and Synchronization Attributes

SMIL 2.0 provides a collection of advanced attributes to control synchronization behavior and to facilitate integration of SMIL into other XML languages. While these attributes are discussed more fully in Chapter 14: *Advanced SMIL Timing Attributes*, we introduce them briefly here.



SMIL synchronization control attributes

In a perfect world, all of the defined timing in a specification would be implemented perfectly by a SMIL 2.0 player. Unfortunately, the world is not only imperfect — it is also unpredictable. In order to provide some measure of control in the face of unpredictably, SMIL 2.0 provides three high-level synchronization control attributes: `syncBehavior`, which allows a presentation to define whether there can be slippage in implementing the composite timeline of the presentation; `syncTolerance`, which defines how much slip is allowed; and `syncMaster`, which allows a particular element to become the "master" timebase against which all others are measured.

XML integration support

When used in a native SMIL 2.0 document (one in which the outer XML tag is `<smil>`), the nature and meaning of various timing elements is clear. When integrating SMIL timing into other XML languages, a mechanism is required to identify timing containers. The SMIL 2.0 specification does this using the `timeContainer` and `timeAction` attributes.

7.3.6 The General Timing Attributes

Some of the attributes from the timing and synchronization modules apply to many SMIL element types. These are grouped into the *general timing attributes*. These attributes are:

- `begin`, `end`, `dur`,
- `repeat`, `repeatCount`, `repeatDur`,
- `restart`, `restartDefault`,
- `fill`, `fillDefault`,
- `min`, `max`,
- `syncBehavior`, `syncBehaviorDefault`, `syncTolerance`, and `syncToleranceDefault`.

Perhaps the most important thing to remember about these attributes is that they may be applied to nearly all of SMIL's elements — not just to media objects.

7.4 Summary and Conclusion



Timing is the most important contribution of SMIL. This chapter introduces the basic constructs of SMIL timing. The basic time container elements `<seq>` and `<par>` specify SMIL's simplest temporal relationships: those of playing in sequence and playing in parallel. The basic in-line timing values of the *inline* timing attributes `begin`, `end` and `dur` set an element's timing as temporal offsets from the begin time it gets from its time container parent. Syncbase values for the `begin` and `end` attributes synchronize an element with the beginning or ending of other elements. The attributes `repeatCount` and `repeatDur` cause an element's media object to repeat its playback. If, after its start time, duration and repeated duration, the element ends before the end of its parent, the `fill` attribute defines the persistence of the element in SMIL's timing tree.

This chapter has considered the basic timing and synchronization attributes. While these cover a broad range of SMIL presentations, there is more to timing control than is presented here. The SMIL timing model also supports interactive and event based timing. These constructs are considered in Chapter 13: *Advanced SMIL Timing Behavior and Control*. (They include the `<excl>` element and various refinements to the `begin` and `end` timing attributes discussed in this section.) The attributes discussed here also have more esoteric values and uses. These are considered in Chapter 14: *Advanced SMIL Timing Attributes*. After experimented with the basic timing support in SMIL, you are encouraged to make more extensive use of SMIL timing by using the advanced features.

A summary chart of the attributes that provide basic timing control in SMIL 2.0 are shown in Table 7-6.

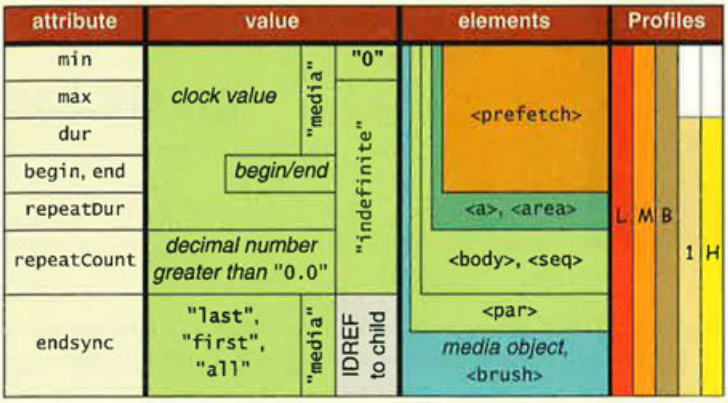
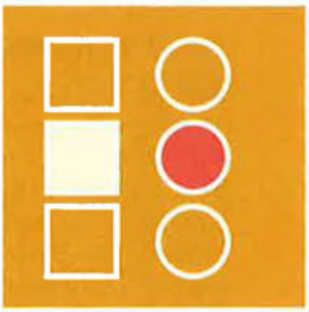


Table 7-6. Structure of the basic timing and synchronization attributes.

16

Extended Content Control



The SMIL *BasicContentControl* module defines a set of elements and attributes that allow the presentation of parts of the document to be determined based on the setting of one or more system test attributes. The facilities provided by the basic content control primitives are essentially the same as those supported in SMIL 1.0, with some minor naming differences and the extension that test attributes can also be placed on SMIL elements without being wrapped in a `<switch>`.

The SMIL 2.0 content control mechanisms also provide a number of advanced features. These include the ability to load media objects before they are actually needed, so that presentation can occur as smoothly as possible, and the ability to extend the set of test attributes to include special-purpose test variables that are defined by the document author. SMIL 2.0 also supports the `skip-content` attribute from SMIL 1.0.

This chapter looks at the three general areas of advanced content control. We begin with a discussion of the `skip-content` attribute, since this attribute is supported by all SMIL 2.0 Language and Basic profile players. We then look at support for pre-fetching media and close with a discussion of the custom test attribute facilities in SMIL 2.0.

The nature of the material in this chapter is such that we segment each topic area into separate sections, each with their own examples. We build on the material presented in Chapter 9: *Content Selection and Control*; readers are encouraged to read that chapter before continuing with the discussions below.

16.1 Skip Content Control

If a prize were to be given for the worst attribute description in the SMIL 2.0 standard, `skip-content` would stand an excellent chance of winning. While nothing in the description is incorrect, there is very little text to help language users and language designers understand the intent and use of skip-content control. In order to understand the operational aspects of skip-content, it is handy to consult the SMIL 1.0 specification. The section *Extending SMIL 1.0* provides a useful summary of skip-content behavior. (The section below also provides all the information you need to know to effectively use the skip-content attribute.)

16.1.1 Motivation: Future-Proofing SMIL

SMIL has strict rules that govern how a SMIL parser must react to errors in a document. In general, if an error is found, the entire document should be rejected. While the reason for this is clear — the temporal structure of the document depends on the total evaluation of the SMIL file, which means that a syntax error in one part can radically impact the timing of other parts — it has the unhandy side-effect that if older parsers attempt to process a newer version of SMIL (in which new element structures have been defined), then the entire document must be rejected when this is not really necessary.

In order to future-proof SMIL, an attribute was defined that can control the parsing and evaluation of elements and element content. The `skip-content` attribute covers two cases:

- If a new element is introduced in some future version of the SMIL language, the `skip-content` attribute can be used to instruct older parsers either to ignore the new element but to still process the children of that element as normal SMIL statements, or to ignore the element and its contents.
- If an existing empty element — that is, it is defined not to have any child content — is changed in a future version of the language to include child content, the `skip-content` attribute can instruct the older parser either to ignore the contents of that element (but to still process the element) or to generate a syntax error.

The content addressed by the `skip-content` directive has nothing to do with any object pointed at by a `src` attribute: `skip-content` only considers the element and the children of that element. This is a key concept in understanding how `skip-content` works.

16.1.2 Skip Content Control Attributes

The *SkipContentControl* module defines one attribute: `skip-content`. Note that since this attribute is a hold-over from SMIL 1.0, it uses old-style hyphenated naming instead of camel cased naming.



Attribute: skip-content

The `skip-content` attribute is a Boolean variable that accepts two values:

- *true*: This value covers two cases.
  (1) If the element containing this attribute is not recognized, ignore the element and ignore its contents.
  (2) If the element containting this attribute is recognized but if its content model changes from empty to non-empty, then process the element but ignore the child content.
- *false*: This value covers two cases.
  (1) If the element containing this attribute is not recognized, skip the elment but process the content (its child elements) as if the unknown element did not exist.

(2) If the element containting this attribute is recognized but if its content model — that is, the fact that it has child elements — is not recognized, generate a syntax error.

The values of the `skip-content` attribute determine the actions of the document parser on unknown elements and content. If the elements and content are recognized by the parser, then the `skip-content` attribute has no effect (it is ignored). The `skip-content` attribute is only evaluated at the point that a parser would otherwise generate an error when processing the document.

The default value for `skip-content` is *true*; this means that for all elements that accept the attribute, any unknown content should be skipped by the parser and not generate syntax errors. Unknown elements are ignored and processed as if `skip-content` is set to *true* (unless `skip-content` is explicitly declared *false*).

16.1.3 Using the skip-content Attribute

We will use the code fragment in Example 16-1 to illustrate the use of the `skip-content` attribute.

Assume that a SMIL 2.0 parser were to process Example 16-1. The following steps will be taken when the document is parsed:

1) This example carries no XML namespace definition. The SMIL 2.0 parser will continue processing the document and will treat it as SMIL 1.0.
2) When SMIL 1.0 was designed, the <region> element on line 4 was not allowed to accept content (in this case, the specification of a hierarchical region as a child on line 5). Since `skip-content` is set to *true* on the parent, the region *F* is processed and the hierarchical region *C* is ignored.
3) When SMIL 1.0 was designed, the <a> element was not part of the language. As a result, the definition on line 11 would ordinarily cause a syntax error during parse. However, since the `skip-content` attribute is defined and is set to *false*, the parser will ignore the <a> element but its contents (in

```
1  <smil >
2   <head>
3    ...
4    <region id="F" height="240" width="280" skip-content="true" >
5     <region id="C" top="220" height="20" width="280" />
6    </region
7   </head>
8   <body>
9   ...
10   <par>
11    <a id="anchor1" href="http://www.example.org/doc.smil"
          target="destination" skip-content="false">
12       <video id="F" src="snarf.mpg" region="b"/>
13    </a>
14    <img id="caption" src="caption.txt" dur="15s" region="C" />
15   </par>
16  ...
17  </body>
18 </smil>
```

Example 16-1. Using the `skip-content` attribute in a SMIL 1.0 file.

this case, the video object) will be processed as if the <a> element wrapper did not exist. If skip-content has been set to *true*, both the <a> element and the <video> element would be ignored. If there had been no skip-content attribute, the statement on line 11 would be processed as if skip-content was declared and set to *true*.

4) The <img> element on line 14 now references a region that does not exist — the definition of the *C* region was ignored. As a result, the content of the media object will be rendered using SMIL default layout semantics, but it will be rendered.

Now, let's see what happens if a SMIL 2.0 parser processes the same file, but this time with a SMIL 2.0 namespace declaration. (This is illustrated in Example 16-2.) The following aspects will occur during parsing:

1) The SMIL 2.0 parser will recognize the SMIL 2.0 namespace definition. It will process this file as SMIL 2.0 content.
2) The region definition on line 4 contains content that is understood by the parser. As a result, the skip-content attribute is ignored.
3) In SMIL 2.0, the <a> element is defined and allowed to have <video> object content. Thus, the parser will again ignore the skip-content attribute and process the statements normally.
4) The <img> element will render to the requested sub-region.

If a future SMIL 3.0 parser were to process this file, it should recognize the SMIL 2.0 namespace and process the file as described above.

The discussion up to this point represents the theory of how SMIL 2.0 (and SMIL 1.0) parsers should work. In practice, the ability to skip document content is of marginal utility. It is really only useful if the namespace declaration is changed to a recognized but unsupported type. In this case, unknown elements will be ignored, but it is doubtful that the result will be meaningful.

```
1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
2  <head>
3   ...
4   <region id="F" height="240" width="280" skip-content="true" >
5    <region id="C" top="220" height="20" width="280" />
6   </region
7  </head>
8  <body>
9  ...
10   <par>
11    <a id="anchor1" href="http://www.example.org/doc.smil"
          target="destination" skip-content="false">
12      <video id="F" src="snarf.mpg" region="b"/>
13    </a>
14    <img id="caption" src="caption.txt" dur="15s" region="C" />
15   </par>
16 ...
17  </body>
18 </smil>
```

Example 16-2. Using the skip-content attribute in a SMIL 2.0 file.

16.2 Custom Test Attributes

One of the important contributions of SMIL 1.0 was the introduction of system test attributes which, together with the <switch> element, allowed a choice to be made among various content paths in a presentation. The SMIL specification defines a set of system test attributes that cover many of the basic selection needs with SMIL documents. These attributes and the <switch> statement are discussed in Chapter 9: *Content Selection and Control*.

The disadvantage of the system test attribute architecture is that all test variables need to be defined as part of the language specification. There is no mechanism to extend the choices available to users on a per-presentation basis. Although it is possible to define custom namespaces with new sets of test attributes, the author would have no control over the recognition of these extensions by a particular SMIL player.

The SMIL *CustomTestAttributes* module defines an architecture for extending the set of test variable available for use in <switch> statements (or as stand-alone test variables). It allows authors to create new test variables for adaptivity that still work on existing SMIL 2.0 players. The module defines two elements and four attributes: the <customAttributes> and <customTest> elements and the customTest, uid, defaultState and override attributes.

16.2.1 Understanding Custom Test Attributes

The custom test attribute architecture is similar in scope and intent to the system test attribute architecture defined in Chapter 9: *Content Selection and Control*: a test variable is defined that can be used to select an element for inclusion in a presentation. The (custom) test variable is defined as an attribute that may be attached to any element: if the attribute evaluates to *true*, the element is included in the presentation; if it evaluates to *false*, it is ignored.

Unlike the system test attribute architecture, there is no pre-defined set of custom test variables. As a result, a mechanism is required to define custom test variables within the SMIL source document.



Custom test variables are defined as part of the <head> section of a presentation by using the <customTest> element. All of the <customTest> elements are defined as children of the <customAttributes> element. At the time of definition, each custom test variable can be given a default value (*true* or *false*). This value may be overridden by either the user (via the user interface of the player) or by consulting an external settings file that is provided via the uid attribute.

Within the user interface community, the issue of providing overrides to system-defined values has always been a hot topic. There are many who feel that, philosophically, the user should always have the final say over the content that is presented. Document authors often feel that they are in the best position to decide which content is appropriate for which set of users. SMIL takes a neutral

position in this debate. The override attribute is made available to define author intentions: the author can state whether the value in the custom test may be overridden by the user (or user agent). User agents are free to enforce or ignore this setting. It is expected that some higher-level protocol will ultimately manage the access to the custom test variable definitions.

We give an example of using custom test variables on page 393 after a more precise definition of the available elements and attributes.

16.2.2 Custom Test Elements

SMIL defines two elements to manage the definition and processing of custom test variables. All of these are considered in this section.

Element: <customAttributes>

Table 16-1 defines the structure of the <customAttributes> element. This element delineates a section in the SMIL <head> that is used to define a set of custom test variables. It contains specifications for all the custom tests used in a presentation.

<customAttributes>		Profiles
attributes	*system test*	L
	skip-content	
	core	
parent	<head>	L
child	<customTest>	L

Table 16-1. Structure of the <customAttibutes> element.

Element: <customTest>

The <customTest> element defines a custom test variable for use in the presentation. The element's id attribute lets the test be referred to from elements in the <body> section that are to be tested. Its structure is laid out in Table 16-2.

<customTest>		Profiles
attributes	defaultState	L
	override	
	uid	
	skip-content	
	core	
parent	<customAttributes>	L

Table 16-2. Structure of the <customTest> element

A <customTest> element's initial value may be overridden by values found at a location defined by the uid attribute or by user interaction though the player interface. The author may specify a preference for the persistence of a test variable value via the override attribute.

16.2.3 Custom Test Attributes

SMIL defines four attributes to manage the definition and processing of custom test variables. Each of these are considered in this section.

Attribute: customTest

The `customTest` attribute takes a list of custom test names defined in the `<head>` section as an argument. If all of the names refer to custom test variables that resolve to *true*, the associated element is included in the presentation. If any of the custom test attributes listed resolve to *false*, the element is not included in the presentation.

All of the rules pertaining to the evaluation of system test variables apply to custom test variables.

Attribute: defaultState

The `defaultState` attribute defines an initial value for the custom test variable. It is used in the `<customTest>` element only. Permissible values are:

- *true:* the associated test variable is true and should allow rendering.
- *false:* the associated test is false and should block rendering.

The default value is *false*.

Attribute: override

The `override` attribute states whether or not the custom test's state should be readily changeable by the user. It accepts the following values:

- *visible*: this value lets the user alter the test's state through the browser's interface for changing custom test states.
- *hidden*: do not list this variable as alterable in this interface.

The default value is *hidden*.

Note that the browser may still provide the user with other, less obvious means to change a *hidden* test state.

Attribute: uid

The `uid` attribute provides a URI to a resource determining the test's state. The URI may simply be a registration of a fixed state for the test. On the other hand, it could be a process that communicates with the browser to set the test's state. This attribute takes priority over the `defaultState` attribute in assigning the state. In turn, the user has highest priority, being able to change the state given by the `uid` attribute.



16.2.4 Examples Using Custom Test Attributes

Use of the custom test attribute facility is illustrated in Figure 16-1. This application illustrates part of a medical dossier. The dossier (of which this fragment is a small part) contains information on a problem with a patient's right back leg. It also contains media objects that describe the operation on the right leg.

The application of custom test attributes is partitioned across two sections of the document. The <head> section contains the definitions of two custom test attributes (*Annotations* and *medicalStaff*), and the <body> section contains references to the state of the attribute values.

In the <head> section, the <customAttributes> element on line 3 contains the definition of the two custom test attributes. Each test variable is defined in a separate <customTest> element. The first definition, of *Annotations* on line 4, creates a test variable with a default state of *true* and with an override value of *visible*. This means that the UI of the player may provide an obvious means of selectively changing the attribute value from *true* to *false* (or vice versa). The second definition, of *medicalStaff* on line 5, creates a test variable with a default state of *false* and with an override value of *hidden*. The intention is that, as part of the user's profile, there is some indication on the capabilities of this viewer. If he or she is a member of the medical staff, they will get access to more detailed information in the presentation.

The <body> section contains several examples of the application of the custom test variables, as well as the use of other content control mechanisms. Line 11 contains a <switch> element. The <switch> evaluates its children in document order, activating the first child that has a test variable (*system* or *custom*) that resolves to true.

If the bandwidth of the network link has been determined by the player to be 56Kb or greater, the <par> element (and its contents) on line 12 is played. If the bandwidth is determined to be lower, then the <seq> on line 20 is selected. Both the <par> on line 12 and the <seq> on line 20 contain a collection of media objects. One of the two elements in the collection is a video object or an image (depending on the bitrate) and the other is a conditional piece of content that is only displayed if the *Annotations* variable resolves to `true`. At the top of Figure 16-1, we see three images. The left two images represent the output of this section of the presentation: if the *Annotations* custom test variable resolves to *true*, the base media object is displayed with an (animated) annotation overlay. If it resolves to *false*, only the base media object is presented.

On line 29, we see another use of the custom test variable. Here, the 29 second <par> is only played is the *medicalStaff* attribute value is *true*. This segment — which illustrates how a scalpel breaks during the operation — consists of a base media object and a separate annotation object; this latter object is only shown if the *Annotations* custom test variable resolves to true. The top-right portion of Figure 16-1 shows the video of the operation with the annotation overlay.



Annotations="true"



Annotations="false"



medicalStaff="true"

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
 <head>
  <customAttributes>
   <customTest id="Annotations" title="View Annotations"
    defaultState="true" override="visible"/>
   <customTest id="medicalStaff" title="Medical Staff Use"
    uid="http://www.example.org/Doctors/list.txt"
    defaultState="false" override="hidden"/>
  </customAttributes>
  ...
 </head>
 <body>
  ...
  <switch>
   <par systemBitrate="56000">
    <video region="Image" dur="11s" src="Video/Symptoms.mpg"/>
    <img id="Annotation1" region="Overlay" begin="1.55" dur="8.7s"
     src="Images/Overlay.gif" customTest="Annotations">
      <animateMotion ... />
      <animate attributeName="width" ... />
      <animate attributeName="height" ... />
    </img>
   </par>
   <seq>
    <par dur="5s">
     <img region="Image" fill="freeze" src="Images/Symp-F20.jpg"/>
     <img region="Overlay" begin="1" dur="3s" src="Images/Overlay.gif"
      customTest="Annotations"/>
    </par>
    ...
   </seq>
  </switch>
  ...
  <par dur="29s" customTest="medicalStaff">
   <video region="Image" dur="29s" src="Video/ScalpelBreaks.mpg"/>
   <img region="Overlay" begin="15" dur="4s" src="Images/Overlay.gif"
    customTest="Annotations"/>
  </par>
  ...
 </body>
</smil>
```

Figure 16-1. Using custom test attributes.

The custom test facility provides an interesting basis for experimentation with user-centered presentation design. Unfortunately, the attributes — although they are part of the SMIL 2.0 Language profile — are not universally implemented. Only the *GRiNS* and Ambulant players provide full support for custom test attribute implementation and integration.

16.3 Prefetch Control

The *PrefetchControl* module allows an author to give hints to the player on when various media items should be fetched from the media server in advance of that media's use. This can allow a presentation to execute more efficiently, since media objects that are needed later in a presentation can be fetched during times when not much transfer activity takes place in the player. This can help spread any peaks in network transfer use across the duration of an entire presentation

16.3.1 Understanding Pre-Fetching of Media

Media objects tend to be relatively large items that require substantial time to transfer across a network infrastructure. Often, a player can optimize presentation performance by pre-loading parts of currently active media items or pre-fetching the next media element in its display queue. Such pre-load and pre-fetch behavior is nearly always local in scope.

In contrast to the player, a presentation author has a global view of which objects will play an important role in the presentation. For this reason, the author is sometimes in a better position to decide which elements should be fetched early. Unfortunately, most authors have little information available about the characteristics of the network connection state during the presentation or the amount of buffer space available to the player. As a result, author pre-fetch directives are taken as informed hints by the player, but the player remains in final control over which objects get fetched when.

In the SMIL pre-fetch model, the loading of future media is not a transparent activity that occurs in the background: an explicit request for a pre-fetch will result in the creation of a pseudo-media reference with a simple duration that gets factored into the timing of the document. (We give an example of this on page 396.)

Pre-fetching media works best when the media object is static — that is, it is an image or video that is not expected to change during the presentation. If you were to pre-fetch stock ticker information, the data that you pre-fetch may actually be stale by the time you present it: since the SMIL player has no means to reliably detecting that the stored information has changed other than information in the original transport header, it cannot invalidate the version in its buffer. Authors should be aware of this problem. For example, if the *http* protocol header indicates that a media object is valid but if the media object changes at the server, this will not be detected.

Another problem of which authors should be aware is the potential for incurring media charges in a pay-per-view (PpV) environment. If a media item is pre-fetched from a PpV server but never rendered because of early termination of the presentation (or because of the settings on a SMIL `<switch>`), users may be presented with charges that they may not realize they had made. SMIL has no mechanisms for guarding against the (mis)use of this situation.

16.3.2 Pre-Fetch Elements

The SMIL *PrefetchControl* module defines <prefetch> to selectively pre-load media objects. Table 16-3 shows the structure of the <prefetch> element.

Element: <prefetch>

The <prefetch> element is used to identify a media object that is to be pre-loaded into the the player. The element contains the name of the media object and zero or more control attributes. The intrinsic duration of the element is the duration of the media fetching activity. This duration can be qualified as a simple duration by providing a dur attribute.

The player may decide not to process the pre-fetch hint. If no pre-fetch takes place, the player must still respect any explicit simple duration on the <prefetch> element. If a pre-fetch is repeated, a fresh copy of the media object must be loaded each iteration.

The author of a presentation may only supply the pre-fetch command as a hint to the player; no control over actual pre-fetch behavior is provided. SMIL 2.0 players should support parsing of the <prefetch> element and attributes, even if no implementation support for pre-fetching media is provided.

<prefetch>		Profiles	
attributes	mediaSize	L	M
	mediaTime		
	bandwidth		
	general timing		
	test		
	core		
parents	<body>	L	M
	<priorityClass>		
	<seq>, <par>		M
	<excl>		
	<switch>		M
	<a>		

Table 16-3. Structure of the <prefetch> element

The amount of data that is pre-fetched is determined by the settings of the pre-fetch control attributes. It is also determined by any control information in the protocols used to transfer the media data: if, for example, media caching is prohibited by the transfer protocol, a pre-fetch may occur but the information will not be saved for reuse.

16.3.3 Pre-Fetch Attributes

The SMIL *PrefetchControl* module defines three attributes to specify pre-fetch control behavior: mediaSize, mediaTime and bandwidth.

Attribute: mediaSize

The mediaSize attribute defines how much of the media object will be pre-loaded in terms of size (number of bytes). It accepts either a byte count or a percentage value. A value of 100% is equal to a request to load the entire object. The default is 100%. A value of 0% is ignored, meaning that the default value of 100% will be applied. If specified together with the mediaTime attribute, the value of the mediaTime attribute will be ignored.

Attribute: mediaTime

The `mediaTime` attribute defines how much of the media object will be pre-loaded in terms of its duration. It accepts either a clock value or a percentage value. A value of 100% is equal to a request to load the entire object. The default is 100%. A value of 0% is ignored, meaning that the default value of 100% will be applied. Discrete media is always completely downloaded. If specified together with the `mediaSize` attribute, the value of the `mediaTime` attribute will be ignored.

Attribute: bandwidth

The `bandwidth` attribute defines how much network bandwidth is allocated to the pre-load activity. It accepts either a bitrate value or a percentage value. A value of 100% implies that all of the bandwidth should be used. An implementation is free to use less bandwidth than requested.

16.3.4 Using Pre-Fetch Control

Example 16-3 illustrates the use of the <prefetch> element. An opening logo is shown at the start of the presentation. The logo has an intrinsic duration of 0 seconds, but the `fill="freeze"` behavior will keep it displayed until the end of the parent <par> element.

During the display of the logo, two pre-fetch operations are started. The first is a request to pre-load the closing logo: it is an image, 100% of which will be fetched and stored (if possible). The simple duration is set to 10 seconds; this forms the minimum activation time for the <par> element. A pre-fetch request is also made for a video object. A fetch of the total media object is requested, using 80% of the available bandwidth (if possible). Since there is no simple duration defined for this pre-fetch, the operation will last as long as it takes to transfer the (large) object. This can consume a considerable amount of time. (This is clearly an unwise strategy, but it is allowed by the SMIL architecture. A better strategy is to limit the amount of information that will be transferred by specifying a `mediaSize` or `mediaTime` attribute) Note that if a simple duration

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
 ...
 <par>
  <img id="intro" src="openingLogo.png" fill="freeze" region="M"/>
  <prefetch id="case1" src="closingLogo.png" mediaSize="100%" dur="10s"/>
  <prefetch id="case2" src="bigVideo.mpg" bandwidth="80%" />
 </par>
 ...
 <seq>
  <video id="Feature" src="bigVideo.mpg" region="M" end="activateEvent"/>
  <img id="theEnd" src="closingLogo.png" region="M" dur="5s" />
 </seq>
 ...
</smil>
```

Example 16-3. Using the <prefetch> element and attributes.

was defined (for example, `dur="10s"`), this would bound the delay within this directive. If the entire video could not be fetched in this period, the implementation is free to cancel the pre-fetch activity.

At some later point in the presentation, the video is shown. The video will run until completion unless the user initiates a mouse click on the rendering region *M*. If so, the video ends and the closing logo is displayed. If the pre-fetch on the closing logo was successful, the image would pop up very quickly.

Note that even if only a few seconds of the video were viewed, it is possible that the entire object was pre-fetched. This can have pay-per-view consequences.

16.4 Summary and Conclusion

The purpose of the extended content control attributes is to provide additional document processing control for advanced users. Of the various features presented, all are supported only in the full SMIL 2.0 Language profile. The skip-content attribute is often recognized and parsed, but it is not always correctly supported by SMIL players. The pre-fetch control attributes are generally supported, but they represent only a high level of presentation performance control. The custom test attributes, although very useful in designing custom-tailored presentations, do not enjoy wide player support.

One would think that there is little reason to use all three sets of attributes. Still, gaining experience with advanced content control features that remain declarative rather than scripted exposes powerful features that will probably only be expanded upon in future versions of the SMIL language.

16.5 Further Resources

SMIL 1.0 Specification.

Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C Recommendation, 15 June 1998. `http://www.w3.org/TR/REC-smil` . See the appendix on *Extending SMIL 1.0* for the motivation for the `skip-content` attribute.



Index

'<' character 22
'#' attribute value character 25, 62–64, 249
  in color values **97**, 97
'%' attribute value character
  in color values **96**, 112, 166
  in region positioning **112**
  in `soundLevel` attribute **118**, 109
'.' attribute value character 412, 419, 423–424
'/' character 22
":smil" substring 78
';' attribute value character 369, 371, 375, 424
'=' attribute value character 424
'>' character 22
"0" attribute value xi, 149, 155
"0.0" attribute value 151, 155, 357
"1.0" attribute value 357
"100%" attribute value 166

A

<a> element **162**, x–xi, 36, 79, 88–89, 137, 141, 149, 151, 155, 162–163, 166, 182, 302–303, 395, 425
absolute length units 282
absolute URIs 64
`abstract` attribute **403**, 424
`accelerate` attribute **358**, 327, 357, 425
accessibility 16, 80
`accesskey` attribute **173**, 425
"`accesskey()`" attribute value substring, 412, 419, 423–424
AccessKeyTiming module 412, 423–424
`accumulate` attribute **374**, 369, 373–374, 424
activation 285
active 34
active duration 279
`actuate` attribute **172**, 425
adaptivity 7, 17–18, 28, 37, 75, 79
`additive` attribute **373**, 369, 373–374, 424
Adobe 14
"`aix`" attribute value **192**
"`all`" attribute value **150**, xi, 149, 155
"`alpha`" attribute value **192**
`alt` attribute **404**, 80, 172, 424
alternative layout 107
"`always`" attribute value **343**, 109
<anchor> element **166**, 88–89
angle brackets 22
<animate> element **368**, 242, 367–371, 373, 375, 424
<animateColor> element **368**, 367–371, 373, 375, 424
<animateMotion> element **368**, 367–371, 373, 375, 424
animation attributes **369**, 368–374
<animation> element **88**, 424
animation elements **367**, x–xi, 88–89, 137, 149, 151, 162, 302–303, 327, 333, 368
Animation Modules 10, 14
animations 6, 11
<area> element **163–164**, xi, 88–89, 141, 149, 151, 155, 166, 182, 242, 368, 425
"`arm`" attribute value **192**
"`arm32`" attribute value **192**
"`arrowHeadWipe`" attribute value **232**

aspect ratio 116
attributeName attribute **370**, 369–371, 424
attributes 369–371
attributeType attribute **371**, 369–370, 424
audio 123, 282
audio descriptions 16
<audio> element **88**, 424
AudioLayout module 412, 423–424
author attribute **403**, 424
authoring 10, 15–16, 18, 80, 92
"auto" attribute value **113**, 109, 112, 333–334, 369–371
autoReverse attribute **357**, 327, 357, 425

B

background color 110, 116
backgroundColor attribute **110**, 107–109, 113, 118, 272, 277, 282, 424
background-color attribute **108**
background-color property of CSS 282
bandwidth 189
bandwidth attribute **396**, 395, 424
"barnDoorWipe" attribute value **231**
"barnVeeWipe" attribute value **231**
"barnZigZagWipe" attribute value **231**
"barWipe" attribute value **230–231**
base URIs 64, 80
basic inline timing 154
basic linking attributes 242
basic time containers 154, 136–137
BasicAnimation elements 79
BasicAnimation module 419, 423–424
BasicContentControl module 412, 414, 416, 419, 423–424
BasicInlineTiming module 412, 414, 416, 419, 423–424
BasicLayout module, 412, 414, 416, 423–424
BasicLinking module 412, 414, 416, 419, 423, 425
BasicMedia module 85, 98, 412, 414, 416, 419, 422–424
BasicTimeContainers module **137**, 412, 414, 416, 419, 423–424
BasicTransitions module 412, 414, 423, 425
begin attribute **142**, 39–40, 154–155, 162–163, 285, 335–336, 338–339, 344, 346–347, 419, 424
"begin" attribute value substring, 424
begin event 39–40
begin time 22, 34, 90, 117, 138, 140, 150, 154, 278–279, 285
begin/end values 141, 155
"beos" attribute value **192**
binary encoding 18
body 98, 282, 390
<body> element **79**, x, 23–25, 77, 88–89, 121–122, 137, 162–163, 184, 302, 368, 395, 424
borderColor attribute **221**, 96, 425
borderWidth attribute **221**, 425
bottom attribute **113**, 33, 108, 111–112, 118, 263, 268, 272, 282, 424
"bottom" attribute value **233**
bottom property of CSS 281–282
"bottomCenter" attribute value **231**
"bottomLeft" attribute value 231, 262
"bottomLeftClockwise" attribute value **234**
"bottomLeftCounterClockwise" attribute value **234**
"bottomLeftDiagonal" attribute value **234**
"bottomMid" attribute value, 262
"bottomRight" attribute value, 231, 262
"bottomRightClockwise" attribute value **234**
"bottomRightCounterClockwise" attribute value **234**
"bottomRightDiagonal" attribute value **234**
"bowTieWipe" attribute value **231**
"boxSnakesWipe" attribute value **234**
"boxWipe" attribute value **231**
browsers 193, 391
<brush> element **89–90**, xi, 79, 85, 88, 95, 98, 141, 147, 149–151, 155, 162, 242, 252, 327, 333, 343, 350, 357, 368, 424
BrushMedia module 98, 412, 423–424
"bsdi" attribute value **192**

by attribute **372**
in animation **372**, 369, 371, 424
in transitions **219**

C

calcMode attribute **219**, **374**, 369, 373–374, 376, 424
calculation modes 374
Candidate Recommendation W3C 257
"canSlip" attribute value **351**, 350
captions 8, 16
CDATA (character data) attribute type 80
CD-ROMs 63
"center" attribute value, 262
"centerRight" attribute value **233**
"centerTop" attribute value **233**
centimeter 282
character attribute value type 412, 419, 423–424
child elements xi, 23, 35, 38–89, 138, 149–150, 155, 334
"circle" attribute value **232**
class attribute **281–283**, 80, 281, 424
class selectors, of CSS 281
classes of XML documents 76–77, 80–81
clipBegin attribute **244–245**, 94, 424
clipEnd attribute **245**, 94, 424
clock values **134**, 350, xi, 141–142, 149, 151, 155
"clockWipe" attribute value **230**, **233**
"clockwiseBottom" attribute value **233**
"clockwiseBottomRight" attribute value **233**
"clockwiseLeft" attribute value **233**
"clockwiseNine" attribute value **233**
"clockwiseRight" attribute value **233**
"clockwiseSix" attribute value **233**
"clockwiseThree" attribute value **233**
"clockwiseTop" attribute value **233**
"clockwiseTopLeft" attribute value **233**
"clockwiseTwelve" attribute value **230**, **233**
close attribute **278**, 278, 282, 424
closing windows 278–279, 283
color 95, 98, 110, 190
color attribute **95**, 88, 90, 96, 98, 110, 424
color names 95
color values **95**, 95–97
comma character 281
Compaq Research 14
containing block 33–113, 268
content attribute **401**, 425
content control 10, 79
Content Control Modules 10, 412, 414, 416, 419, 423–424
content, of elements 151
coords attribute **168**, 425, 242
copyright attribute **403**, 424
core attributes **80**, x, 77–80, 88, 107–108, 118, 137, 182, 185, 210, 216, 252, 263, 272, 276–277, 281, 302–303, 368, 390, 395
"cornersIn" attribute value **231**
"cornersOut" attribute value **231**
"counterClockwiseBottomLeft" attribute value **233**
"counterClockwiseTopRight" attribute value **233**
country codes 186
"crossfade" attribute value **235**
CSS (Cascading Style Sheets) 98–104, 123, 282–283
"CSS" attribute value, 369–370
current document 64
custom test attributes **394**, 182
<customAttributes> element **390**, 390, 424
customTest attribute **391**, 107–108, 118, 162–163, 252, 263, 272, 276–277, 424
<customTest> element **390**, 78, 184, 390, 424
CustomTestAttributes module **392**, 412, 419, 423–424
CWI (Centrum voor Wiskunde en Informatica) 15

D

"data" attribute value **253**
decelerate attribute **358**, 327, 357, 425

decimal number values xi, 141, 149, 151, 155, 357
default attribute assignments xi, 149, 155, 166, 333, 343, 350, 357, 369–371, 373–375, 391
"default" attribute value **334**, 333, 343, 350-352, 369, 373
default inheritence 327, 333, 343, 350, 334
default layout 102–107
defaultState attribute **391**, 390–391, 424
"defer" attribute value **304**, 304
deprecated constructs 89, 194, 424–425, 153
descendants of elements 106, 279, 334
descriptive markup 10
destination of link 39–40, 285
destinationLevel attribute **172**, 166, 425
destinationPlaystate attribute **170**, 166, 425
"dgux" attribute value **192**
D-HTML (Dynamic HTML) 17
"diagonalBottomLeft" attribute value **232**
"diagonalBottomLeftOpposite" attribute value **235**
"diagonalTopLeft" attribute value **232**
"diagonalTopLeftOpposite" attribute value **235**
"diagonalWipe" attribute value **232**
dialects 186–187
"diamond" attribute value **233**
direction attribute **212**, 425
directories 63–64
"disable" attribute value **305**
"discrete" attribute value **374**, 369, 373
display environments 166
"doubleBarnDoor" attribute value **232**
"doubleDiamond" attribute value **232**
"doubleFanWipe" attribute value **234**
"doubleSweepWipe" attribute value **234**
"down" attribute value **232–233**
DTDs (Document Type Definitions) 15–16, 75, 77
dubbing 188
dur attribute **143–145**, 23–25, 154–155, 162–163, 210–211, 216, 335, 344, 346–347, 419, 424–425

E

editing tools 39–40
"element" attribute value **375**, 369, 373
element type 22
elements 23
"ellipseWipe" attribute value **232**
empty layout 102
end attribute **145–147**, 39–40, 154–155, 162–163, 285, 419, 424
"end" attribute value substring, 424
end event 39–40
end time 35, 90, 140, 150, 152, 154, 279, 285, 333–334
endProgress attribute **211**, 425
endsync attribute **150**, x–xi, 88, 137, 149, 155, 302, 419, 424
end-tags 22
erase attribute **253**, 88, 424
events 39–40, 285, 424
EventTiming module 412, 414, 419, 423–424
<excl> element **301–302**, x–xi, 39–40, 79, 88–89, 137, 147, 149–151, 285, 302–303, 327, 368, 395, 424
ExclTimeContainers module 412, 419, 423–424
extensions to SMIL 81
external attribute **172**, 166, 425
"eyeWipe" attribute value **232**

F

"fade" attribute value **235**
fadeColor attribute **212**, 96, 219, 425
"fadeFromColor" attribute value **235**
"fadeToColor" attribute value **235**
"false" attribute value **166**, 350, 357
"fanInHorizontal" attribute value **233**
"fanInVertical" attribute value **233**
"fanOutHorizontal" attribute value **233**

"fanOutVertical" attribute value **233**
"fanWipe" attribute value **234**
"file://" URI scheme 63
filenames 64
files 8, 17, 25, 62–64, 249
fill attribute **147**, **333**, 79, 154, 327, 333, 335–336, 338–339, 344, 346–347, 424
"fill" attribute value **114–116**, 109, 282
fill behavior 334
fillDefault attribute **334**, 154, 327, 333, 343, 424
FillDefault module 412, 424
"first" attribute value xi, 149, 155
fit attribute **114**, 108–109, 114–116, 118, 272, 282, 424
"fivePoint" attribute value **232**
Flash 17
font size 282
"fourBlade" attribute value **233**
"fourBoxVertical" attribute value **234**
"fourBoxWipe" attribute value **231**
"fourPoint" attribute value **232**
fragment attribute, 425
fragment identifiers **249**, 62–250
fragments 64, 249
"freebsd" attribute value **192**
"freeze" attribute value **333**
freeze period 279
from attribute **371**, 218, 369, 371, 424
"fromBottom" attribute value **235**
"fromLeft" attribute value **235**
"fromRight" attribute value **235**
"fromTop" attribute value **235**

G

general timing attributes **154**, x, 79, 88, 90, 137, 302, 368, 395
generic identifier 22
GRiNS 6, 16, 28, 39–40

H

head 282
<head> element **78**, 77–79, 107, 182, 184, 210, 216, 276, 390, 424
"heart" attribute value **232**
height attribute **112–113**, 33, 107–108, 110–113, 118–119, 121–122, 268, 272, 277, 282, 424
height property of CSS 282
"hexagonWipe" attribute value **232**
"hidden" attribute value **391**, 109, 114, 282
hidden value of CSS 282
"hide" attribute value **304**
hierarchical regions 282
HierarchicalLayout module **271**, 412, 423–424
hierarchy, of elements **23**, 23–89, 283
higher attribute **304**, 303, 424
"hold" attribute value **333–334**, 333
"horizontal" attribute value **231–233**
"horizontalLeft" attribute value **234**
"horizontalLeftSame" attribute value **234**
"horizontalRight" attribute value **234**
"horizontalRightSame" attribute value **234**
"horizontalTopLeftOpposite" attribute value **234**
"horizontalTopRightOpposite" attribute value **234**
horzRepeat attribute **221**, 425
HPAS 14
"hppa1.1" attribute value **192**
"hpux" attribute value **192**
href attribute, 25, 36, 39–40, 64, 162–163, 166, 220, 424–425
HTML 10, 13, 17, 21, 36, 53, 62, 64, 82, 90, 98, 112, 123, 176, 281, 283, 424–425
<html> element of HTML 78, 82
"http://" URI scheme 62–63
"http://www.w3.org/1999/xhtml" attribute value, 78
"http://www.w3.org/2000/svg" attribute value, 78
"http://www.w3.org/2001/SMIL20" attribute value, 78
"http://www.w3.org/2001/SMIL20/Language" attribute value 28, 41, 75, 77, 82, 121–122, 193

I

IBM 14
id attribute **80**, 24–25, 110, 119, 121–122, 263, 390, 424
  in <region> elements **111**, 109, 119, 121–122, 282
  in temporal hierarchy 120–122
ID attribute type 24–25, 109
IDREF attribute type xi, 24, 80, 149, 155, 369–370, 412, 419, 423–424
IETF (Internet Engineering Task Force) 54, 83, 98, 257
<img> element **88**, 120–121, 281, 424
"indefinite" attribute value **135**, xi, 141, 149, 151, 155, 424
"independent" attribute value 350, 351
"inherit" attribute value **334**, 109, 344, 352
inheritance 282
inline timing attributes 154
InlineTransitions module 412, 419, 423, 425
integer attribute value substring type 109
integer values 190
integration 31
interaction 10, 79
internal URI references 64
Internet 62–63
Internet Explorer 13
interpolations 374
"irisWipe" attribute value **230**, 232
"irix" attribute value **192**
ISO/IEC 200–201, 406–407

K

"keyhole" attribute value **232**
keySplines attribute **376**, 375, 424
keyTimes attribute **376**, 375, 424

L

language of user 8
language sub-tags 186–187, 201, 407
language tags 80, 186, 200, 406
"last" attribute value, xi, 149, 155
layout **99**, 6, 10, 18, 28, 32, 75, 78, 88–326
<layout> element **106**, 78, 105, 107–108, 117–118, 182, 263, 272, 276–277, 281, 424
Layout Modules 10, 123, 412, 414, 416, 423–424
left attribute **112**, 33, 108, 111–113, 118, 263, 268, 272, 282, 424
"left" attribute value **231–233**
left property of CSS 281–282
"leftCenter" attribute value **231**
"leftToRight" attribute value **230–231**
length values **112**
"linear" attribute value **374**, 369, 373
linking 6, 21, 28, 36, 75, 80
linking in context 8
Linking Modules, 10, 412, 423, 425
LinkingAttributes attributes 162–163
LinkingAttributes module 412, 414, 423, 425
"linux" attribute value **192**
"locked" attribute value 351, 350
longdesc attribute **405**, 80, 424
lower attribute **304**, 303, 424

M

"m68k" attribute value **192**
"macos" attribute value **192**
Macromedia 17
MAGpie 16
"marker()" attribute value substring 412, 419, 423–424
"marker=" attribute value substring 424
max attribute **150**, 154–155, 162–163, 424
"media" attribute value **136**, xi, 141, 149, 155
media clipping attributes **244**, 88
media content **85**, 9, 16–18, 28, 31, 34, 75, 79–98, 103, 193, 253, 282, 357
media creation 56–59
media description attributes **402**, x, 79, 88, 137, 302–303
media object elements **88**, x–xi, 39–40, 79, 85, 88–90, 98, 123, 137, 141, 147, 149–151, 155, 162, 182, 242, 252, 268, 302–303, 327, 368

Media Object Modules 10, 412, 414, 416, 422–424
media objects 98, 151
media parameter attributes 252
media processing 253
media repositories 64
MediaAccessibility module 412, 414, 423–424
media-based SMIL 77, 81
MediaClipMarkers module 412, 423–424
MediaClipping module 412, 414, 419, 423–424
MediaDescription module 412, 414, 423–424
MediaMarkerTiming module 412, 423–424
MediaParam module 412, 423–424
`mediaRepeat` attribute **253**, 88, 424
`mediaSize` attribute **395**, 424
`mediaTime` attribute **396**, 395, 424
"`meet`" attribute value **116**, 109, 282
`<meta>` element **400**, 78, 184, 425
meta-data 78
`<metadata>` element **400**, 78, 184, 425
Metainformation module 10, 414, 425
Microsoft 13
"`midLeft`" attribute value, 262
"`midRight`" attribute value, 262
mimetypes **92**
`min` attribute **149–150**, xi, 149, 154–155, 162–163, 424
MinMaxTiming module 412, 414, 416, 419, 423–424
"`mips`" attribute value **192**
"`miscDiagonalWipe`" attribute value **231**
"`miscShapeWipe`" attribute value **232**
mobile devices 8, 17
`mode` attribute **217**, 425
modularization 10–11, 34, 193–194
MPEG-4 14, 17–18
MultiArcTiming module 412, 419, 423–424
MultiWindowLayout module 412, 423–424

N

`name` attribute
in `<param>` element **252**, 252
in `<meta>` element **400**, 424–425
name attribute type 109, 111, 249
names, of attributes 369–370
namespaces 10, 77, 81, 192–194, 371
native SMIL 76–78, 81–82, 318
NCAM (National Center for Accessible Media) 16
"`ncr`" attribute value **192**
"`nec`" attribute value **192**
nested elements 39–40
"`netbsd`" attribute value **192**
"`never`" attribute value **254**, 304, 343
"`new`" attribute value **171**, 166
newline characters 193
"`nextstep`" attribute value **192**
`nohref` attribute, 425
"`none`" attribute value 369, 373
non-linear time 5
"`nto`" attribute value **192**
number attribute value substring type 109, 112, 424

O

"`object`" attribute value **253**
ObjectLinking module 412, 419, 423, 425
"`off`" attribute value **186**, 188
offsets, temporal 154
"`on`" attribute value **186**, 188
"`onRequest`" attribute value **278**
"`onStart`" attribute value **278**
"`opaque`" attribute value **254**
`open` attribute **278**, 282, 424
open source 16
open standards 17–18
"`openbsd`" attribute value **192**
opening windows 278–279, 283
"`openvms`" attribute value **192**
"`oppositeHorizontal`" attribute value **233**
"`oppositeVertical`" attribute value **233**
Oratrix 16
`origin` attribute **374**, 369, 373, 424
"`os2`" attribute value **192**
"`osf`" attribute value **192**
"`overdub`" attribute value **188**, 186
`overflow` property of CSS 282

override attribute **391**, 390, 424

P

"paced" attribute value **374**, 369, 373
"palmos" attribute value **192**
<par> element **137**, x–xi, 39–40, 79, 88–89, 121, 137, 141, 147, 149–151, 154–155, 302–303, 327, 333, 335, 338, 340–341, 344–347, 368, 395, 424
parallel playback 154
"parallelDiagonal" attribute value **233**
"parallelDiagonalBottomLeft" attribute value **233**
"parallelDiagonalTopLeft" attribute value **233**
"parallelSnakesWipe" attribute value **234**
"parallelVertical" attribute value **233**
<param> element **252**, 88–89, 252, 424
"parent" attribute value **375**, 369, 373
parent elements 33, 375
parentheses 412, 419, 423–424
path attribute **376**, 375, 424
path, of URI **63**, 62–64
pathnames 63
paths, from SVG 375
"pause" attribute value **171**, 166, 303–304
pauseDisplay attribute **304**, 303, 424
peers attribute **303–304**, 303, 424
"pentagonWipe" attribute value **232**
percentage as spatial measure 282
percentages as attribute values 112, 166, 263, 282
Perl 16
Perly SMIL 16
"pinWheelWipe" attribute value **233**
pixel units 110, 112–113, 281–282
pixels 105, 190, 263
"play" attribute value **171**, 166
points as spatial measure 282
"ppc" attribute value **192**
pre-defined registration points 262–263
<prefetch> element **395**, xi, 79, 141, 149, 155, 162, 182, 327, 333, 395, 424
PrefetchControl module 412, 414, 423–424
"preserve" attribute value **253**
primary language tags 186–187, 200, 406
<priorityClass> element **302–303**, x, 88–89, 137, 162, 302–304, 368, 395, 424
privacy 192
profiles 7, 11, 18, 35, 98, 103–105
properties of CSS 282–283, 371
proprietary formats 17–18
"pushWipe" attribute value **234**
"px" attribute value substring **112**, 110, 112

Q

"qnx" attribute value **192**
QuickTime 9

R

readIndex attribute **405**, 88, 424
RealNetworks 9, 16
RealOne 18
RealPlayer 9
Recommendation, W3C 6, 14, 53–54, 82–83, 98, 123, 176, 257, 283, 326, 384, 406
"rectangle" attribute value **230**, 232
"ref" attribute value **253**
<ref> element **88**, 31, 88, 424
regAlign attribute **263**, 88, 261, 263, 424
region attribute **117**, x, 24–25, 88, 90, 117, 120–121, 137, 268, 281–282, 302, 424
<region> element **108**, 272, 24–25, 105–108, 112, 118–119, 121–123, 263, 267–268, 272, 276–277, 279, 282, 424
region positioning 33, 112–113, 268
region positioning attributes **111**, 112
regionName attribute **111**, 108–109, 118, 272, 282, 424
regions 33, 80, 102, 105, 123, 267
registration points 260
regPoint attribute **263**, 88, 261, 424
<regPoint> element **262**, 261, 263, 424

relative paths 64
relative URIs 62, 64
"remove" attribute value **333**
repeat attribute 154
repeat timing attributes 162–163
"repeat()" attribute value substring 424
repeatCount attribute **151**, 140, 154–155, 424
repeatDur attribute **152**, 140, 151, 154–155, 424
repeating 151, 154
RepeatTiming module 412, 414, 419, 423–424
RepeatValueTiming module 412, 423–424
"preplace" attribute value **171**
"replace" attribute value 166, 369, 373–374
restart attribute **343**, 154, 327–344, 424
restartDefault attribute **344**, 154, 327–344, 419, 424
RestartDefault module 412, 419, 423–424
RestartTiming module 412, 419, 423–424
RFC (Request for Comments) 200, 406
"rgb" attribute value substring 96
RGB color specification **96–97**, 95
RGB decimal triplet **96**
RGB functional notation 96
RGB hexadecimal notation **97**, 96
RGB percentage triplet **96**
"rhapsody" attribute value **192**
right attribute **113**, 33, 108, 111–112, 118, 263, 268, 272, 282, 424
"right" attribute value **231–233**
right property of CSS 281–282
"rightCenter" attribute value **231**
root element **22**, 77, 81
root window 111
<root-layout> element **107–108**, 105–107, 110, 113, 276, 282, 424
"roundRectWipe" attribute value **232**
"rs6000" attribute value **192**
RTSP 63
"rtsp://" URI scheme 63

S

's' attribute value character 23–25, 335, 424
S2M2 14
"saloonDoorWipe" attribute value **233**
scalability framework 194
schemas 16, 77
scheme, of URI **63**, 62–63
Schmunzel 14
"sco" attribute value **192**
screen resolution 8
"scroll" attribute value **114**, 109, 114, 282
scroll value of CSS 282
seeking 285
selection of content 38–40, 90
sensitivity attribute **254**, 424
<seq> element **138**, x–xi, 39–40, 79, 88–89, 137, 141, 149, 151, 154–155, 302–303, 327, 368, 395, 424
sequential playback 154
server, of URI **63**, 62–64
servers 63
<set> element **368**, 242, 367–370, 424
shape attribute **168**, 242, 425
show attribute **171**, 166, 425
"show" attribute value, 304
showBackground attribute **114**, 108–109, 118, 272, 278, 282, 424
simple duration 357
"singleSweepWipe" attribute value **233**
"sinix" attribute value **192**
"sixPoint" attribute value **232**
skip-content attribute **386**, 107–108, 118, 210, 216, 252, 263, 272, 277, 368, 390, 424
SkipContentControl module 412, 414, 416, 423–424
slash character 63
"slice" attribute value **116**, 109, 282
"slideWipe" attribute value **234**
SMIL 1.0 9–16, 88, 98, 102–103, 107–108, 110, 118, 150, 184, 194, 419, 424–425

SMIL Basic (SMIL 2.0 Basic Language profile) 105, 107, 140, 184, 194, 272, 425
SMIL basic layout 13, 102, 281–283
<smil> element **77**, 28, 75–79, 121–122, 424
SMIL Language (SMIL 2.0 Language profile) 11, 15–16, 18, 98, 107, 140, 263, 302–303, 334, 344, 352, 425
SMIL language family 76, 81
SMIL Syntax Validator 15
SMILGen 16
"snakeWipe" attribute value **230**, 234
Soja 14
"solaris" attribute value **192**
"sonly" attribute value **192**
soundLevel attribute **118**, 109, 118, 272, 282, 424
source of link 39–40, 285
sourceLevel attribute **172**, 166, 425
sourcePlaystate attribute **171**, 166, 425
space characters 193, 375
speed attribute **357**, 327, 357, 425
"spiralWipe" attribute value **234**
"spline" attribute value 424
SplineAnimation module 424
src attribute **90**, 25, 64, 85, 88, 90, 98, 120–121, 335–336, 338–340, 344, 346–347, 424
stack level 105
startProgress attribute **211**, 425
start-tags 22
"starWipe" attribute value **232**
state of presentation 285
"stop" attribute value **171**, 166, 303–304
streaming media 9, 63
"strip" attribute value 253
Structure module 10, 75, 412, 414, 416, 423–424
sub-presentations 251, 182, 193
sub-region attributes 88, 268
sub-regions **267**, 282
"subtitle" attribute value **188**, 186
subtitles 188
subtype attribute **211**, 216, 230–235, 425
"sum" attribute value **374**, 369, 373–374
Sun Microsystems 14
"sunos" attribute value **192**
SVG (Scalable Vector Graphics) 8, 10, 14, 18, 53, 62, 64, 76–78, 82, 98, 140, 162–163, 249, 375, 384, 424–425
<svg> element of SVG 78
SVG Viewer 14
<switch> element **182–184**, x, 38–40, 78–79, 88–89, 107, 137, 162, 182, 276, 302–303, 368, 395, 424
syncbase elements 39–40
syncbase values 154
syncbases **135**
SyncbaseTiming module 412, 419, 423–424
syncBehavior attribute **350**, 154, 327, 347, 350–351, 424
SyncBehavior module 412, 419, 424
syncBehaviorDefault attribute **351**, 154, 327, 350–351, 424
SyncBehaviorDefault module 412, 424
synchronization 9, 13, 16, 18, 35, 80, 82, 90, 139, 154, 279, 347
synchronization behavior **347**, 350–353
synchronization master 347
syncMaster attribute **353**, 327, 347, 424
SyncMaster module 424
syncTolerance attribute **352**, 154, 327, 347, 350, 352, 424
syncToleranceDefault attribute **352**, 154, 327, 350, 352, 424
synthesized speech 282
system colors 95, 97
system test attributes **185**, 107–108, 118, 162–163, 182, 184, 186, 190, 192, 194, 252, 263, 272, 276–277, 390
systemAudioDesc attribute **188**, 184, 186, 424
system-bitrate attribute **194**
systemBitrate attribute **189**, 184, 190, 194, 424
system-captions attribute **194**
systemCaptions attribute **187**, 184, 186, 194, 424

systemComponent attribute **193**, 184, 192, 424
systemCPU attribute **192**, 184, 424
system-language attribute **194**
systemLanguage attribute **186**, 184, 194, 400, 424
systemOperatingSystem attribute **192–193**, 184, 192, 424
system-overdub-or-caption attribute 194
systemOverdubOrSubtitle attribute **188**, 184, 186–188, 194, 424
system-required attribute **194**
systemRequired attribute **193–194**, 184, 192, 194, 424
system-screen-depth attribute **194**
systemScreenDepth attribute **190**, 184, 194, 424
system-screen-size attribute **194**
systemScreenSize attribute **190**, 184, 194, 424

T

tab characters 193
tab key 90
tabindex attribute **172**, 88, 90, 425
tags, of XML elements 23, 318
target attribute **172**, 166, 425
target elements 371
targetElement attribute **370**, 220, 369–370, 424–425
temporal composition **138–139**, 16, 33–35, 38–40, 139–140, 150, 285, 318, 347
temporal context 152
test attributes **184–185**, x, 38, 88, 90, 137, 184, 186, 190, 192, 210, 216, 302–303, 368, 395, 400
text content of presentations 25, 282–283
<text> element **88**, 22–25, 424
"text/css" attribute value **281**
textflow-based layout 13
<textstream> element **88**, 424
time container elements 154, 162, 182
timeAction attribute **320**, 318, 424
timeContainer attribute **319**, 88, 318, 424
TimeContainerAttributes module 424
timeline 347
TimeManipulations module 10, 425
timing 6, 79, 88, 154, 28, 34–35, 75
Timing and Synchronization Modules 10–11, 34, 278, 412, 414, 416, 419, 422–424
title attribute **403**, 80, 424
to attribute **372**, 218, 369, 371, 424
top attribute **112**, 33, 108, 111–113, 118–119, 121, 263, 268, 272, 282, 424
"top" attribute value **233**
top property of CSS 281–282
"topCenter" attribute value **231**
<topLayout> element **276**, 113, 272, 276–279, 282, 424
topLayoutCloseEvent event **279**
topLayoutOpenEvent event **279**
"topLeft" attribute value, 231, 262–263
"topLeftClockwise" attribute value **234**
"topLeftCounterClockwise" attribute value **234**
"topLeftDiagonal" attribute value **234**
"topLeftHorizontal" attribute value **230**, **234**
"topLeftVertical" attribute value **234**
"topMid" attribute value, 262
"topRight" attribute value, 231, 262
"topRightClockwise" attribute value **234**
"topRightCounterClockwise" attribute value **234**
"topRightDiagonal" attribute value **234**
"topToBottom" attribute value **231**
transfer protocols **94**, 62–63
transIn attribute **213**, 88, 90, 425
"transition" attribute value **333**
Transition Effects Module 10, 425
<transition> element **210**, 78, 216, 425
<transitionFilter> element **216**, 327, 357, 425
TransitionModifiers module 412, 419, 423, 425

transitions 6, 10, 78, 90, 333
transOut attribute **214**, 88, 90, 425
transparency 116
"transparent" attribute value **110**, **254**, 113
tree, of elements **23**, 22–23, 98
"triangleWipe" attribute value **232**
"true" attribute value **166**, 350, 357
"twoBladeHorizontal" attribute value **233**
"twoBladeVertical" attribute value **233**
"twoBoxBottom" attribute value **234**
"twoBoxLeft" attribute value **234**
"twoBoxRight" attribute value **234**
"twoBoxTop" attribute value **234**
type attribute
  in <layout> elements, 107, 276, 281, 424
  in media object elements **92–94**, 85, 88, 90, 98, 424
  in <param> elements **253**, 252, 210
  in <transition> elements, 230–235, 425
types of media 17, 90, 103, 249, 92

U

uid attribute **391**, 390, 424
UNINETT 200, 406
unique identifier references 24
unique identifiers 24, 80, 249
"unixware" attribute value **192**
"unknown" attribute value **192**
"up" attribute value **231–233**
URIs (Uniform Resource Identifiers) **62**, 54, 62–63, 80, 83, 98, 166, 192–193, 257
URL(Uniform Resource Locator) 81
user preferences 8, 187–188, 192
users 92, 391

V

value attribute **253**, 252, 424
values attribute **373**, 219, 369, 371, 424
values, of attributes 369, 371
valuetype attribute **253**, 252–253, 424
"vax" attribute value **192**
"veeWipe" attribute value **231**
"vertical" attribute value **231–233**
"verticalBottomLeftOpposite" attribute value **234**
"verticalBottomSame" attribute value **234**
"verticalLeft" attribute value **234**
"verticalRight" attribute value **234**
"verticalTopLeftOpposite" attribute value **234**
"verticalTopSame" attribute value **234**
vertRepeat attribute **221**, 425
<video> element **88**, 424
viewport 282
"visible" attribute value **391**
visible value of CSS 282
volume property of CSS 282

W

W3C (World Wide Web Consortium) 3, 6, 10, 14, 53–54, 81–83, 98, 123, 176, 257, 283, 326, 384, 406–407
wallclock value 424
"wallclock()" attribute value substring 424
WallclockTiming module 412, 419, 423–424
"waterfallWipe" attribute value **234**
"whenActive" attribute value **278**, 109
"whenDone" attribute value **254**
"whenNotActive" attribute value **278–279**, 343
white space characters 193, 192
width attribute **112–113**, 33, 107–108, 110–113, 118–119, 121–122, 268, 272, 277, 282, 424
width property of CSS 282
"win16" attribute value **192**
"win32" attribute value **192**
"win9x" attribute value **192**
"wince" attribute value **192**
windows 278 6, 33–112, 123, 281–282
"windshieldWipe" attribute value **233**
"winnt" attribute value **192**

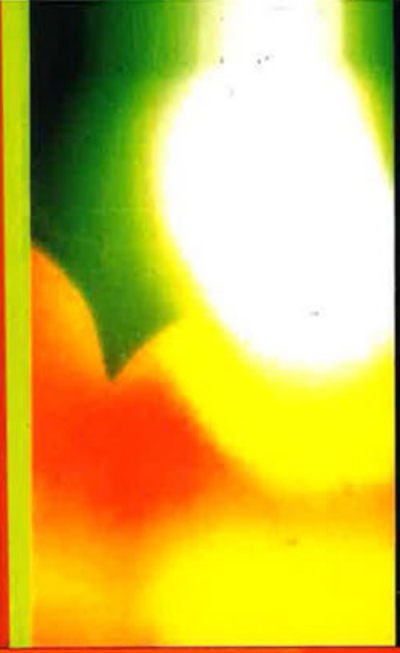
X

'x' attribute value character 190
"x86" attribute value **192**
XHTML 10
XHTML+SMIL 13, 16, 76, 105, 140, 375, 424–425
XLink (XML Linking Language) 425
XML viii–ix, xxviii, 1, 3, 9–11, 13–18, 21–26, 28, 31, 34, 45, 47, 53–54, 64, 73, 75–78, 80–83, 88, 98, 111, 121–122, 125, 136, 153, 162–164, 171, 176, 208, 210, 220, 248–250, 257, 281, 285, 318–319, 326, 361, 369–371, 381–382, 387, 399, 401, 404, 406, 409
"XML" attribute value **371**, 369–370
"xml:" namespace prefix **81**, 424
xml:base attribute **80**, 64, 80–81
xml:lang attribute **403**, 80–81, 424
xmlns attribute **81**, 28, 75, 77–78, 81–82, 121–122, 424
XMT (eXtensible MPEG-4 Textual Format) 11, 14, 18
XPointer **250**, 10, 176, 257
XSLT (XSL Transforms) 10

Z

"zigZagWipe" attribute value **231**
z-index attribute **116**, 105, 108–109, 118–119, 121–122, 272, 282, 424
z-index property of CSS 282

X.media.publishing



Dick C. A. Bulterman
Lloyd Rutledge

SMIL 2.0

The SMIL 2.0 Language is the Web's solution for multimedia. This book provides a comprehensive guide to all of SMIL 2.0's elements and attributes for both desktop and mobile multimedia presentations. Driven by dozens of examples and illustrated code fragments, every aspect of SMIL is treated.

Starting with essential background information on media encoding and transfer, this book considers the document structure, media integration, presentation layout and detailed timing constructs available in a wide range of SMIL compatible players and browsers. The book also considers advanced SMIL topics of linking, content control and SMIL animation, all from a practical, implementation-oriented perspective.

Key Topics

- Web-based multimedia
- Timing and synchronization
- Interactive activation
- Audio, video and text integration
- Presentation layout
- Conditional content
- User accessibility
- Links and hypermedia
- Multi-profile Web and mobile implementations

Dick Bulterman and **Lloyd Rutledge** are research staff members at CWI, the Dutch national center for mathematics and computer science in Amsterdam. Both were members of the SMIL 1.0 and SMIL 2.0 development committees of W3C, the World Wide Web Consortium. Together they have over 30 years of experience with multimedia and the Web.

ISSN 1612-1449
ISBN 3-540-20234-X

9 783540 202349

springeronline.com