



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 9,319,720
Amazon EX-1012

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@cw.nl
Lloyd.Rutledge@cw.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: KunkelLopka, 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 />
    <AttributeDefinitions />
    <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.

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

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:

Functional Group		
Modules	Structure	Meta-Information
	Timing	Time Manipulation
	Linking	Content Control
	Media Objects	Layout
	Animation	Transitions

An example of an attribute table is shown below.

attribute	value		elements	Profiles	
min	clock value	"0"	<prefetch>	L M B	
max		"indefinite"	<a>, <area> animation		
endsync	"last", "first", "all"	"media"	<body>, <seq>		
		IDREF to child	<par>, <excl> media object, <brush>		

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
GRiNS	The Oratrix GRiNS 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 <i>North-American and Asian Television Standard</i>
PAL	Phase Alternation Line <i>World-Wide Television Standard</i>
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 *Rogério 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 *Katelijnn 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.

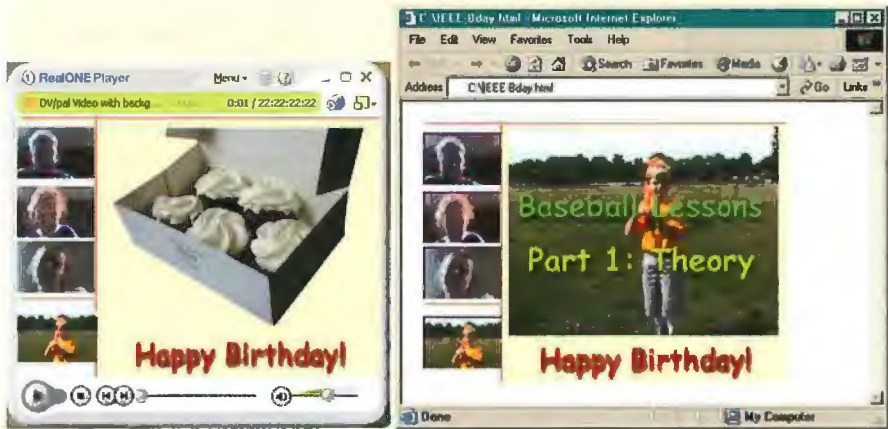


Figure 1-3. *Happy Birthday!*, as shown in native SMIL and XHTML+SMIL players.



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.

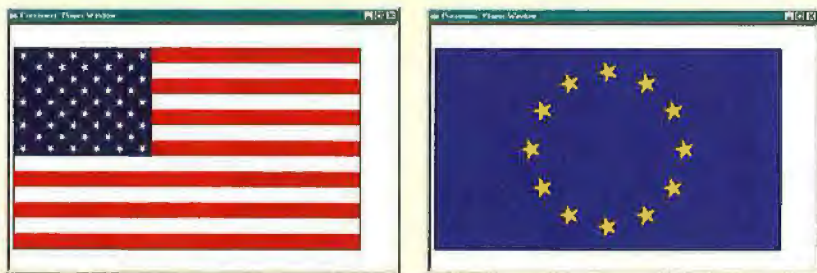


Figure 1-6. *Flags*, an example of SMIL-based animation in the SVG language.

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.

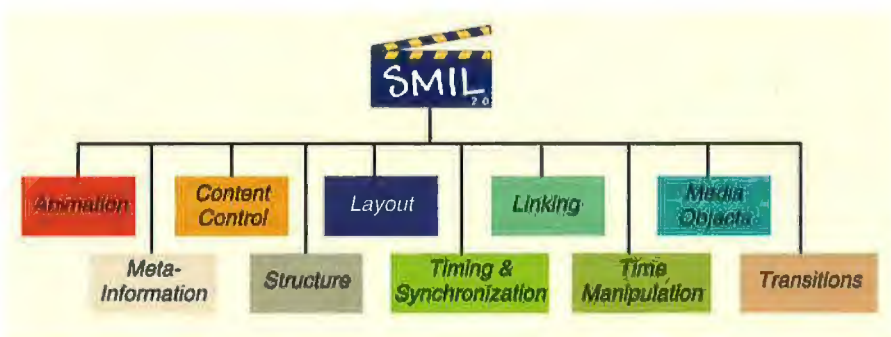


Figure 1-7. SMIL module architecture.



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 *SMILGen*, 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 *GRiNS* 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 *GRiNS* pre-viewer.

1.5.6 GRiNS

The industrial-strength power tool for SMIL 2.0 authoring is *GRiNS*, by Oratrix. *GRiNS* 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. *GRiNS* 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, *GRiNS* 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, *GRiNS* is useful for the more expert authors of larger presentations, but it can have a higher learning curve than simple text editors. The *GRiNS* 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/> .

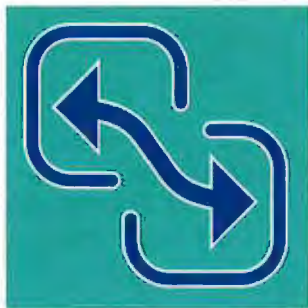
Vendor SMIL Pages

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.

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.)

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.

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
	GRNS	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, misc 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.

http://www.example.org/dir0/dir1/dir2/name.smil/#here

↑ ↑ ↑ ↑

scheme server path fragment identifier

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.

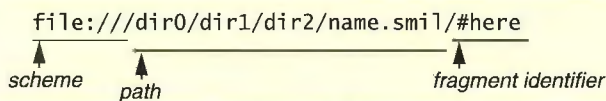


Figure 3-2. The Local URI

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.

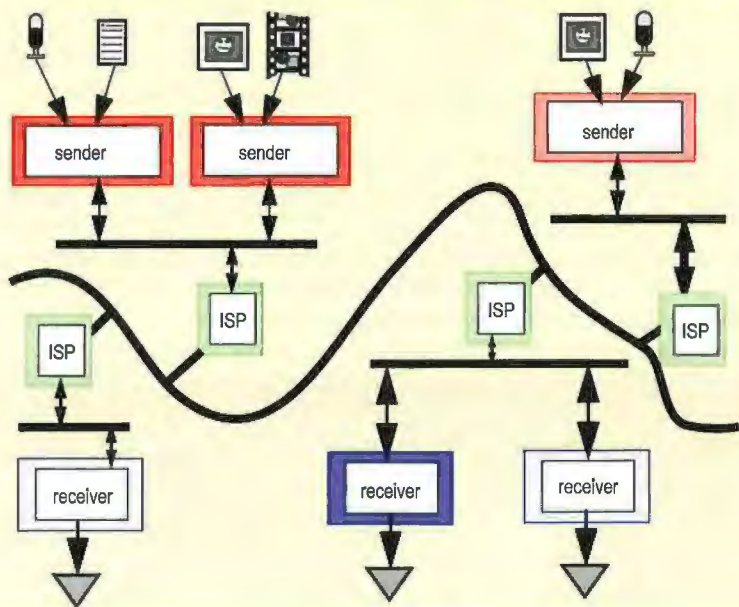


Figure 3-3. An overview of the network environment.
(The [Internet] backbone is not as homogeneous as shown.)

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-

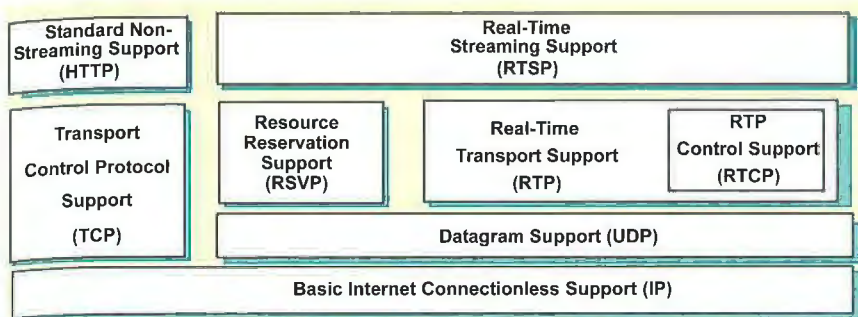


Figure 3-4. Types and relationships among network protocols for multimedia.



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/>.



RealNetworks 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

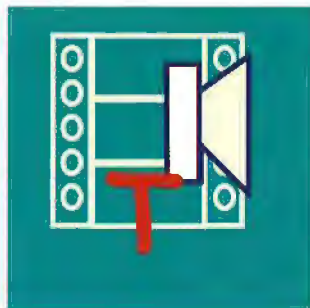
By the time you complete this part, you should be able to author presentations that make use of all of SMIL's basic multimedia presentation structuring facilities. You will also be able to understand and maintain SMIL presentations written by others for the various SMIL 2.0 profiles. The key to this 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 through 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.

```

1 <smil>
2 <head>
3 ...
4 </head>
5 <body>
6 ...
7 <text src="file:///hello.txt" type="text/plain" ... />
8 ...
9 </body>
10 </smil>

```

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

```

1 <smil>
2 <head>
3 ...
4 </head>
5 <body>
6 ...
7 <ref src="http://www.example.net/hello.txt" type="text/plain" ... />
8 ...
9 </body>
10 </smil>

```

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

```

1 <smil>
2 <head>
3 ...
4 </head>
5 <body>
6 ...
7 <text src="data:,Greetings!" type="text/plain" ... />
8 ...
9 </body>
10 </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.)

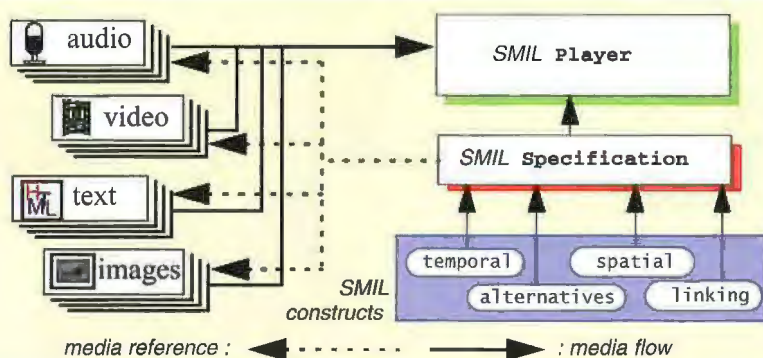


Figure 5-1. SMIL architecture.

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.

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>*: **, *<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

media object elements <ref>, , <video>, <audio>, <animation>, <text>, <textstream>		<brush>		Profiles			
attribute	media	src	color	M	B		
		type				1	H
		media clipping					
		media description					
		mediaRepeat					
		erase					
		readIndex					
	layout	region		M	B	1	
		sub-region					
		regalign, regPoint					
	timing	general timing		M	B	1	
		endsync					H
		timeContainer					
	Misc	transIn, transOut		M			
		tabindex					
		test		M	B	1	H
	parents	core					
		<body>		M	B	1	
		<a>					
		<seq>, <par>					H
		<exc1>					
		<priorityClass>					
	parents & children	<switch>		L	M	B	1 H
	children	<area>			M		H
		<anchor>					1
		animation					H
		<param>		M			

Table 5-1. Structure of the media object elements

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 roleName="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 mime-type 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.

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 mime-type of the media object, perhaps overriding the default processing that is associated with a particular name; or

Media Class	MIME Type
audio	audio/basic audio/mp3 audio/mpeg audio/vnd.m-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.m-realvideo video/x-msvideo video/x-ms-wmv
image	image/bmp image/gif image/jpeg image/png image/svg+xml image/tiff image/vnd.m-realflash image/vnd.m-realpix image/x-cmu-raster image/x-emf image/x-rgb image/x-wmf
text	text/html text/plain text/vnd.m-realtxt
composite	application/vnd.m-realmmedia application/x-shockwave-flash application/smil application/xhtml+xml

Table 5-2. Common *mimetype* definitions.

- *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 mime-types. 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="data:image/gif;base64,0x0aef7edb1dcab2abcd...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.

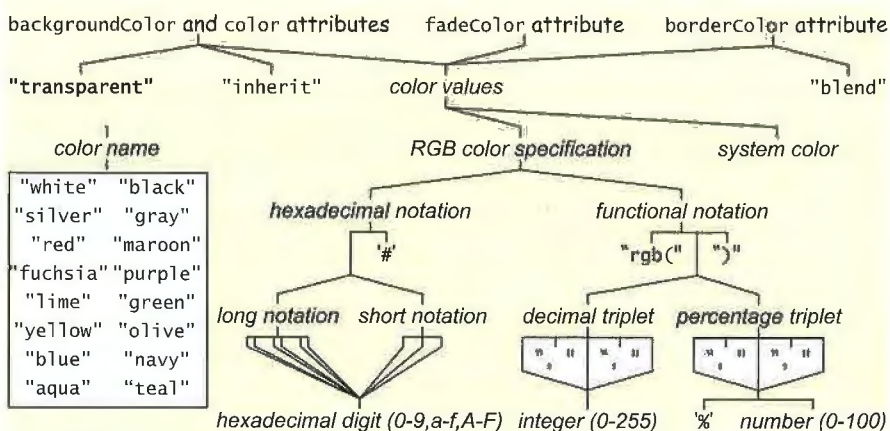


Figure 5-2. Syntax for values of attributes using color

- **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>.



6

SMIL Basic Layout



Layout states *where* on the screen each piece of visual media is shown and *how* audio objects are rendered when activated. A presentation's layout defines the visual "style" or "look and feel" of the composite combination of all of its media objects. Layout is important because it adds the semantic organization that helps a viewer absorb the multiple content streams inherent in a multimedia presentation quickly and efficiently.

SMIL layout is concerned with the relative placement of (multiple) media objects, but not the internal formatting of any of the individual objects. Explicit layout facilities exist in SMIL because the temporal and lexical structure of a SMIL document typically has no direct relationship with the spatial structure of the presentation. The placement of each object in a presentation is usually driven more by media centric concerns (such as having video and images appear in predictable places) than with any inherited properties based on the lexical place within the document of the media object. This makes SMIL fundamentally different than, say, HTML or most text documents.

This chapter presents the core facilities of SMIL layout. These include the elements and attributes that control presentation layout, and the way that SMIL separates layout specification from layout use. We will also consider alternatives to SMIL layout in a presentation, including SMIL's default layout rules and the ability to use alternative layout languages. If you don't care about the models, jump to page 104 for the basic concepts of SMIL 2.0 layout.

6.1 An Overview of Layout Concepts

There are several different strategies appropriate for multimedia layout. In order to understand some of the alternatives, consider the multimedia presentation shown in Figure 6-1. Here we see a hypothetical news broadcast, which is made up of several elements: there are several static icons and labels, there is a place for video and graphics, there is room for (optional) captions, and there is a ticker tape area that is intended to keep viewers with shorter attention spans engaged throughout the presentation. The contents of any of the objects may change during the newscast — the studio video may be replaced with a remote report or the labels and captions may change over time — but the general visual structure will remain the same.



Figure 6-1. A SMIL presentation of a Web newscast.

In organizing the presentation's media objects, a designer can choose one of three general approaches:

- *Embedded Layout*: The first is to assume all layout concerns are defined by the underlying media items. With this strategy, all of the layout decisions are resolved at media creation time and then 'baked' into the presentation. This approach gives ultimate control to the presentation designer and no control (other than composite rendering) to the media player.
- *Dynamic Layout*: At the opposite end of the layout spectrum is the approach in which all layout is determined dynamically by the presentation environment, based on the structure of the document or the timeline of the presentation. This philosophy gives a controlling role in how a presentation is displayed to the viewer's media player at the expense of the presentation's designer.
- *Compositing Layout*: A compromise between dynamic and embedded layout is to consider a presentation as a composite of relatively autonomous objects (each using internal embedded or dynamic layout models) that are visually positioned into a composite arrangement by a presentation designer. This approach de-couples media contents from media placement, but it leaves the presentation designer in control.

Each of these approaches has their advantages and disadvantages.

The key advantage of using embedded layout is that a certain look-and-feel is guaranteed at production time. In terms of Figure 6-1, the components can be carefully formatted, and complex content models can be used to provide often stunning visual effects. A disadvantage of the approach is that it limits individual component reuse or easy content substitution: if you want to reuse the video or the ticker tape news in an other context, you are out of luck. (While this is seen as a disadvantage in Web terms, it is often viewed as a prime advantage in other contexts.)

tage by content owners who want final-form formatting and strict distribution control.)

The key advantage of dynamic layout is that all placement decisions can be made based on knowledge of the final rendering environment. Figure 6-2 shows two dynamic restylings of the presentation for a 2::3 aspect ratio instead of the original 4::3. Figure 6-2(a) uses a lexical algorithm, where a scaled background and the ticker tape are placed as peers, and the video, graphics, images and text are nested within these objects in a top-to-bottom order based on their order of appearance in the document's source file. Figure 6-2(b) illustrates a restyling based on a temporal algorithm: each successive media object is nested in order of temporal appearance in the presentation, with each new object presented with slight offset in origin and at the next-higher stacking level. Both approaches are very predictable, but both are pretty useless because neither the structure of the source document or the coarse timing of the presentation have a direct relationship with the spatial needs of the presentation. Dynamic styling has no idea of the semantics of the information presented, which means that the composite message conveyed by a particular placement of objects is lost or altered. The result is usually undesirable for presentations with more than one simultaneous media object.

The advantage of using compositing layout is that it provides the ability to combine autonomous objects into a 'new' whole, but it still provides a designer with total control over the coordinated (visual) composition of the objects. Since multimedia presentations have multiple, simultaneous objects, this coor-

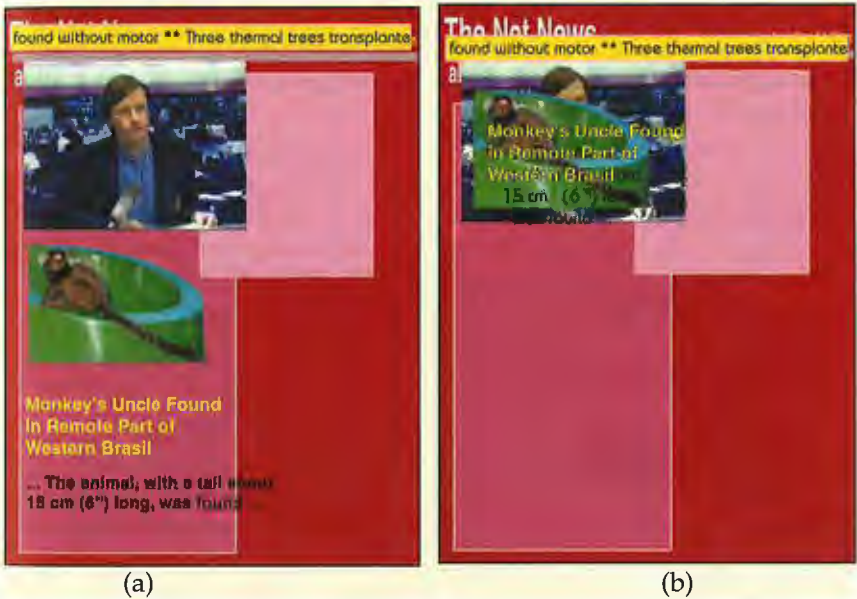


Figure 6-2. A dynamically transformed Web newscast.

- (a): layout based on lexical structure, as shown in Example 6-1 on page 121;
- (b): layout based on temporal order, as used by SMIL Empty Layout.

dination is essential to eliminating confusion in a presentation. At the same time, composition layout still provides the player with an ability to influence selection and — when appropriate — the rendering of individual objects. However, considering the processing expense of dynamically reformatting media such as audio, video and images, the restyling of media types other than text will be the exception, not the rule.

6.2 SMIL Layout Models

The initial proposals for SMIL layout were based on strict compositing model. From a graphic designer's perspective, the ability to directly influence layout within the presentation specification was seen as a great good. In terms of the W3C standardization process for SMIL 1.0, however, it was seen as a great problem. This debate, which centered on the definition of a special-purpose layout model within the SMIL specification rather than having SMIL apply CSS, resulted in a rather complex overall layout solution in SMIL when it was released in 1996. This section describes the basic layout facilities defined for SMIL 1.0 and then considers SMIL 2.0's extensions.

6.2.1 SMIL 1.0's Approach to Layout

SMIL 1.0 supported three forms of layout control:

- **SMIL *Default Layout*:** an implicit layout architecture based on a fully dynamic model that allowed the media player to make placement decisions. To obtain this model, a presentation excluded all layout definitions from the <head> element. SMIL 1.0 players were free to integrate their own layout constraint mechanisms — such as style sheets — but no major player offered this functionality.
- **SMIL *Empty Layout*:** an implicit set of dynamic layout rules based on a model in which objects are offset and stacked based on their temporal appearance in the document. (Figure 6-2(b) used the algorithm of SMIL Empty Layout.) To obtain this model, a presentation integrated the following empty layout declaration in the <head> section:

```
<head>
  <layout>
</layout>
</head>
```

(See *SMIL Empty Layout Semantics* on page 117 for more details.)

- **SMIL *Basic Layout*:** an explicit layout architecture based on the compositing model that provided logical containers (regions) that could be used to define where and how related groups of media are rendered. To obtain this model, the following specification was included in the <head>:

```
<head>
  <layout>
    ... <!-- explicit layout definition -->
  </layout>
</head>
```

A design constraint for SMIL 1.0 Basic Layout was that it had to maintain compatibility with CSS-2 absolute positioning functionality. SMIL 1.0 layout was strongly-typed and enforced a one-to-one association of media with layout regions: each region could only be used to render a single type of media, and only one media object could be active in a region at a time. (A region could be used for several media items of the same type, but only one at a time.)

For presentations that consisted of a single media object — such as one video or a single audio object — SMIL Empty Layout provided a baseline layout solution. For more complex presentations, SMIL Basic Layout provided a reasonable way of providing the basic kinds of control necessary for multimedia. (Since SMIL Default Layout was totally player dependent, its use is discouraged.)

6.2.2 SMIL 2.0's Approach to Layout

The essence of SMIL 1.0 layout has been carried forward and expanded in SMIL 2.0. As with SMIL 1.0, profiles based on SMIL 2.0 do not need to provide explicit support for layout control. On the other hand, in languages where layout plays an important part of the presentation's specification, extensive new facilities have been provided to allow fine-grain control of how objects are placed on the rendering surface. Most of the restrictions on SMIL 1.0 layout were also removed.

SMIL 1.0's *Default* and *Empty* models are preserved in SMIL 2.0. As illustrated in Figure 6-3, SMIL 1.0's *Basic* model has been extended and modularized into four (optional) layout modules. These are:

- *BasicLayout*: this module defines the `<layout>`, `<root-layout>` and `<region>` elements, the `region` attribute, and various control attributes for SMIL regions. It provides core layout functionality that is backwards compatible with SMIL 1.0 layout.
- *AudioLayout*: this module defines the `soundLevel` attribute as an extension to the `<region>` element. In contrast to the module and attribute names,

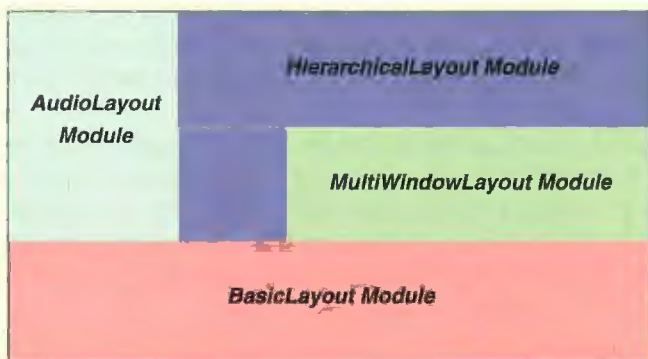


Figure 6-3. SMIL 2.0 Layout Modules and Dependencies

this module does not really define where audio is placed (for example, on which channel) or the absolute output level of the audio. Instead, it defines the relative value of the audio output.

- *MultiWindowLayout*: this module extends *BasicLayout* and allows multiple top-level windows to be defined and coordinated within one presentation context. It defines the `<topLayout>` element and the `open` and `close` attributes for managing the multiple windows.
- *HierarchicalLayout*: this module extends *BasicLayout* and *MultiWindowLayout*. It provides the ability to define layout overrides on the media definition element, it allows regions to be created dynamically within the presentation, and it allows content to be aligned relative to a set of registration points within the region.

All profiles that support SMIL Basic Layout will include the *BasicLayout* module. Only players that are based on the full SMIL 2.0 Language profile should be expected to offer complete support for all layout modules.

6.2.3 Summary of SMIL Layout Concepts

Before looking at the layout model in detail, we summarize some of the important concepts about SMIL's layout models:

- *Layout specification is optional in SMIL*: you don't have to use SMIL Basic (or Advanced) layout, but if you do use it in a player that conforms to the SMIL Basic or SMIL Language profiles, you are guaranteed to get SMIL's layout semantics;
- *Individual profiles may choose to extend or replace SMIL Basic Layout with other models*: the proposed XHTML+SMIL profile uses CSS layout, while some new profile may choose to define new layout models.
- *SMIL Layout is not constraint-based*: you don't define in terms of relationships among objects, you define layout in terms of one or more layout canvases. (There are some constraint based relationships defined in the *HierarchicalLayout* module.)
- *SMIL 2.0's implementation of Basic Layout is backwards compatible*: it provides extra functionality above that in SMIL 1.0, but all players that support SMIL Basic Layout must be backward compatible and render SMIL 1.0 documents correctly.
- *SMIL Basic Layout and CSS*: all of SMIL 1.0 Basic Layout can be transformed into a CSS-2 equivalent notation, albeit in a less concise manner than is convenient for multimedia specifications. In SMIL 2.0, the mapping between SMIL layout and CSS is not complete, but equivalent forms exist for many standard operations. We look into the relationship between SMIL layout and CSS in Chapter 12: *Advanced Layout Topics*.

This chapter discusses the Basic Layout and Audio Layout support. Multi-Window Layout and Hierarchical Layout are discussed in Chapter 12.

6.3 SMIL Basic Layout Elements and Attributes

Some layout features are so fundamental that all types of multimedia use them. Most multimedia applications place visual media at various rectangular locations on a presentation space consisting of a single window. SMIL 2.0 Basic Layout defines this behavior. It acts as the foundation for all SMIL-defined layout, being required for the use of any of the other layout modules.

SMIL Basic Layout constructs are sufficient for the display of multimedia on small-screen, portable devices; SMIL Basic and 3GPP SMIL devices use only these constructs for their layout. XHTML+SMIL, on the other hand, does not use Basic Layout, but provides support for positioning constructs based on CSS's absolute positioning.

The elements that are used to define layout positioning support for SMIL 2.0 Basic Layout are:

- `<layout>`, which defines a collection of regions for the presentation;
- `<root-layout>`, which defines the overall window for the presentation;¹ and
- `<region>`, each of which defines an area in the display for placement of visual media or a container for the rendering of audio media.

A presentation may have zero, one or — if `<switch>` is supported — multiple `<layout>` elements. Within each `<layout>` element, zero or one `<root-layout>` elements and zero or many `<region>` elements may appear. Neither `<root-layout>` nor `<region>` have children in Basic Layout. Within a non-empty `<layout>` section, a presentation may define at most one `<root-layout>` element and it may define multiple `<region>` elements.

Figure 6-4 revisits the example of Figure 6-1 to show how the components in the news presentation can be mapped to the various elements in SMIL 2.0 Basic Layout. The news presentation defines a set of interior regions and an outer player window.

The layout definition begins with the specification of an over-all canvas container (the `<root-layout>`), which defines the boundaries of that presentation's rendering space. This container tells the SMIL player how large the composite presentation window will be. The player may enlarge this window with extra control interfaces or provides scroll bars, if these are required for the final rendering. Such extra space is outside of SMIL Basic Layout's control.

Inside the `<root-layout>` canvas are six rendering regions, labelled *B*, *V*, *G*, *H*, *C* and *T*. Each of the regions reserves a range of pixels that can be assigned to various media objects during the presentation. A region's stack level (called its *z-index*) will determine which bits are actually visible in case of overlapping use of region pixels. Each region has attributes for identification and control;

¹ Note that while most SMIL 2.0 constructs use camel-case naming, SMIL layout uses the name `<root-layout>` instead of `<rootLayout>`. You must use the hyphenated form.



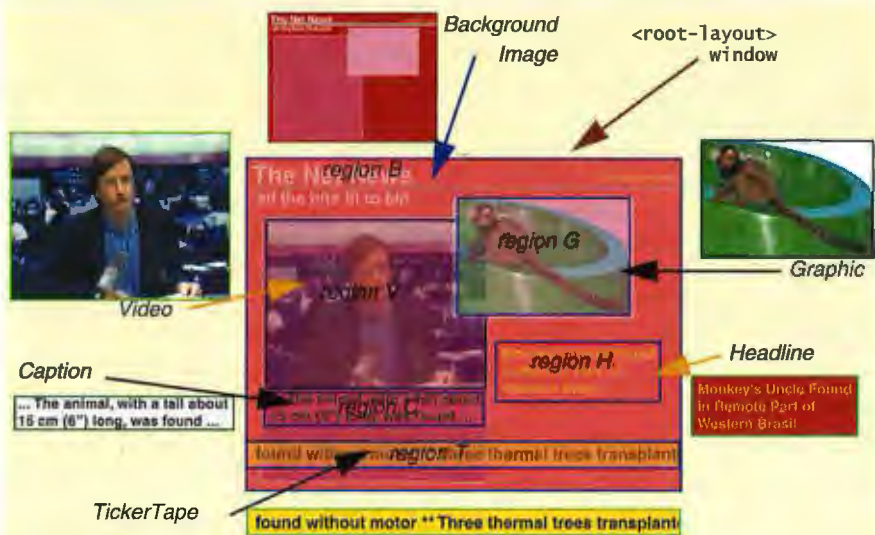


Figure 6-4. Layout Design for the Newscast Presentation

three sets of attributes are shown: the name, the size, and the z-index. (Other attributes are considered later.) By default, when a region is used to display a media object, the object is placed in the upper-left corner of that region.

Quick Tip

Not all mobile devices support scrolling of media within layout regions. Be careful not to rely on the mobile player to fit your media to the device's screen.

6.3.1 SMIL Basic Layout Elements

This section describes `<layout>`, `<root-layout>` and `<region>`.

Element: `<layout>`

The `<layout>` element specifies the placement rules for visual objects in a SMIL presentation. Most of this specification happens through its children, particularly through its child `<region>` elements. These elements define all placement of visual objects within the presentation window. A `<layout>` element can also have, optionally, one `<root-layout>` element child, which specifies the presentation window itself. All elements from the four SMIL 2.0 layout modules appear as descendants of `<layout>` elements, and most layout-related attributes appear in `<layout>` elements or their descendants. The `<layout>` element structure is shown in Table 6-1.

If a presentation has no `<layout>` element or if none of the type attributes are recognized, the player's internal default layout rules are used. If it has one empty `<layout>` section, then SMIL's empty layout rules are used. (They are described in *SMIL Empty Layout Semantics* on page 117.) If it has one non-empty `<layout>` section, then SMIL Basic Layout rules are applied, as discussed in this section. If it has multiple `<layout>` sections, then the first section that is recognized by the player based on a `<switch>` element evaluation is used. (See also Chapter 9.)

<layout>		Profiles			
attributes	type			B	1
	core	L	M		
	system test				
	customTest				
parents	<head>	L	M	B	1
	<switch>				
children	<region>	L	M	B	1
	<root-layout>				

Table 6-1. Structure of the `<layout>` element

Quick Tip

For documents that use players conforming to the SMIL Basic or Language profiles, the use of either explicit or empty SMIL layout is recommended. Having at least one `<layout>` element in a SMIL document prevents inconsistent presentation across different implementations.

Element: `<root-layout>`

The `<root-layout>` element defines the characteristics of the top level window in the presentation. There can be only one `<root-layout>` element within a layout section. This window is open during the entire presentation. The `<root-layout>` element is empty: it contains no `<region>` elements as children. (The `<region>` elements defining the regions to be placed in the `<root-layout>` window are siblings.) The `<root-layout>` structure is shown in Table 6-2.

<root-layout>		Profiles			
attributes	backgroundColor				
	width, height				
	core	L	M	B	1
	system test, skip-content				
	customTest				
parents	<layout>	L	M	B	1
children	none			M	B 1

Table 6-2. Structure of the `<root-layout>` element

As its hyphenated name suggests, `<root-layout>` is a holdover from SMIL 1.0, which has only single-window presentations. It is maintained for compatibility use in SMIL 2.0 and for use in SMIL Basic, which uses only one window. The `<root-layout>` element is still allowed in multi-window presentations,

but usually should be replaced by `<topLayout>` in common use. (See Chapter 12: *Advanced Layout Topics* for details.) It may be useful as an authoring convenience indicating one window as the primary window, such as a central menu window controlling the rest of the presentation.

Element: `<region>`

The `<region>` element is the primary connection between media object elements and SMIL's layout structure. For most applications, each `<region>` element defines a container that is used to render one or more media objects. The regions may hold arbitrary content: audio, video, text, animations, images or arbitrary programs that render to a visual or audio region. SMIL does not restrict the types of content, although various SMIL players may provide support for only limited data types. The structure of the `<region>` element is laid out in Table 6-3.

A major change between SMIL 1.0 and SMIL 2.0 is that SMIL 2.0 regions can support the simultaneous rendering of multiple media objects (of arbitrary types). If only the *BasicLayout* module is supported, this facility is of limited utility because there is no way of selectively positioning objects within the region, and thus information will be obscured. Selective positioning is provided via the sub-region positioning attributes in the *HierarchicalLayout* module. (See Chapter 12: *Advanced Layout Topics* for details.)

There are several types of attributes that apply to regions. These include the following groups:

- *referencing attributes*: these give the region a reference name;
- *positioning attributes*: these allow the region to be placed within the `<root-layout>`;
- *background attributes*: these define how parts of the region not containing media objects are rendered;
- *stacking and scaling attributes*: these attributes determine either how regions are placed relative to other regions, or how over-/under-sized media is placed within a region.

Each group of `<region>` element attributes is considered in the following sections.

<region>		Profiles			
attributes	regionName				
	top, left, width, height, bottom, right	M	B		1
	z-index, fit				
	showBackground				
	backgroundColor				
	core	M	B		1
	system test, skip-content				
	customTest				
parents	<layout>		M	B	1
children	none		M	B	1

Table 6-3. The `<region>` element's place in SMIL structure

6.3.2 Basic Layout Attributes Architecture

This section considers three sets of layout attributes, one for each of the layout elements defined in *SMIL Basic Layout Elements and Attributes* on page 105.

The attributes taken by each element are limited, except for the `<region>` element. The values that each `<region>` attribute can take are summarized in Table 6-4.

Class	attribute	value
Referencing	regionName	name
	id	id
Scaling	z-index	integer auto inherit
	fit	hidden scroll fill meet slice
Positioning	width, height, top, left, bottom, right	length-values auto
Background	backgroundColor	Color value
	showBackground	always whenActive
Audio	soundLevel	number %

Table 6-4. Attribute values of the `<region>` element.

6.3.3 `<layout>` Attributes

The `<layout>` element accepts the core SMIL attributes and — if supported by the profile — it also accepts attributes from the *SystemTest* module. It also accepts the `type` attribute, which is used to specify the layout mechanism to be used by the SMIL player.

If the `type` specification defined in a `<layout>` element is:

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

then SMIL's Basic Layout semantic are used. It is also possible to specify other layout models if the player supports these. If an invalid or unrecognized type definition is given, then the entire layout specification is skipped. (This will usually result in a presentation not being rendered at all, but this facility can be used as a brute-force method of selecting among alternative layout models.) If no `type` attribute is defined, then *text/smil-basic-layout* is assumed.

6.3.4 <root-layout> Attributes

The layout attributes available on the <root-layout> element control the size of the outer window and the color of the background.

```
***
<head>
  <layout type="text/smil-basic-layout">
    <root-layout id="Player-window" width="640" height="480"
      backgroundColor="red"/>
  </layout>
</head>
***
```

Attribute: height and width

The size of a window is determined by its element's width and height attributes. The value of each attribute is always a number stating the given dimension in pixels, with an optional suffix of "px". This is an absolute specification of the window's size. If either of these attributes is not explicitly set, then the browser determines this dimension of the root window in any manner it sees fit. Thus, for predictable behavior across browsers, both of these attributes should always be assigned.

Attribute: backgroundColor

If the images you put on the screen do not cover all of the space or if a region is left empty for part of a presentation, the `backgroundColor` attribute determines how the unused space in the region is filled. When applied to the <root-layout> element, it sets the background for an entire presentation.

The `backgroundColor` attribute takes the same values as the `color` attribute. Its default value is *transparent*. Unless you are sure of the resulting behavior in the SMIL player, it is typically best to override this for the <root-layout> element (otherwise parts of the system background may appear). For reasons of compatibility with CSS-2, the `backgroundColor` attribute may also be given the value of *inherit*. While the intention of this facility is admirable, it is useless in practice, since none of the parent elements of <root-layout> accept a color-setting attribute.

Quick Tip

The `background-color` attribute is included in SMIL 2.0 for compatibility with SMIL 1.0. The attribute and its value are treated exactly as if it were a `backgroundColor` attribute. There is no reason to use this attribute in a new SMIL 2.0 presentation — `backgroundColor` should be used instead.

Other Attributes

The <root-layout> also accepts the core SMIL attributes.

6.3.5 <region> Referencing Attributes

There are two ways of naming a region: using `id` or with `regionName`. The choice depends on whether multiple regions need to be referenced by a single name.

Attribute: `id`

Like all SMIL elements, the `<region>` element can have an `id` attribute. For `<region>`, the `id` attribute has the special meaning that it can establish a name for that region. In SMIL 1.0, the `id` attribute was the only way of naming a region. Because of XML restrictions, each `id` must be unique. This means that presentations that have alternative layouts, or in which a single media object is to be rendered in multiple regions, need another mechanism to identify the target region. SMIL 2.0 introduced the `regionName` attribute for this purpose. As a result, the `id` attribute is only treated as a region name if no other region is assigned that same name via a `regionName` attribute.

Attribute: `regionName`

The `regionName` attribute takes an XML name string as a value, and makes this value the name of the region. Like the `id` attribute, it establishes its value as a means of referring to this element from another. Unlike the `id` attribute, this attribute does not have to be unique, and thus it enables multiple regions to share the same name. This is useful if you intend to have multiple layouts in a single presentation.

When resolving a region reference, a SMIL player will first see if one or more `<region>` elements exist that define a `regionName` corresponding to the reference name. If no `regionName` exists, the player will see if there is a `<region>` with an `id` attribute matching the reference name. If neither a `regionName` or an `id` is found, the reference is treated as an orphan. See *SMIL Empty Layout Semantics* on page 117 for more details on orphan processing.

Quick Tip

Since `regionName` is more flexible than `id`, you should use it when constructing presentations.

6.3.6 <region> Positioning Attributes

The primary use of the `<region>` element is *region positioning*. The region positioning attributes are `top`, `left`, `width`, `height`, `bottom` and `right`. They define rectangular areas in which visual media are displayed. The possible values of these attributes and their use is illustrated in Figure 6-5.

The positioning attributes are relative to the region's *containing block*, which is determined by the `<region>` element's parent. For the *BasicLayout* module (and for SMIL 1.0) the containing block is the root window — the region is



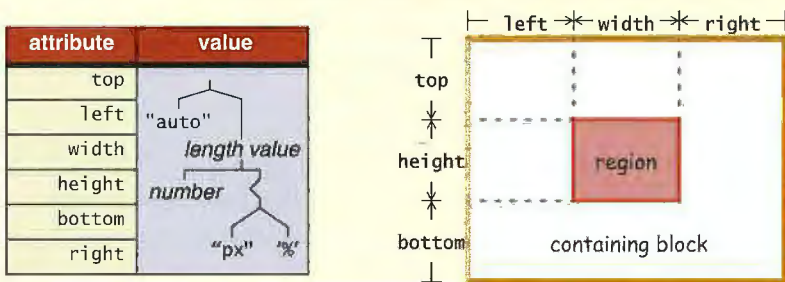


Figure 6-5. Region positioning attributes

placed in the root window, and the region's measurements are in terms of the root window. If you do not define region positioning attributes, the region fills up its containing block by default.

The values of the `<region>` element can contain percentages or pixel offsets. For Basic Layout and SMIL 1.0, these attributes do not directly specify the size of the window actually shown on the screen. Instead, they specify the size of the display area within this window. The window itself may also consist of the display area plus additional borders and scrollbars, increasing the total size of the window beyond the stated width and height dimensions. If a SMIL presentation is being written to cover the entire screen of a windowed environment, then this increased size should be taken into account. This additional size will vary between SMIL browsers and windowing environments.

The positioning attributes use *length values* to assign dimensions along the screen. To state a distance in terms of display screen pixels, make the value either a number by itself or a number followed by "px". To state the distance as a percentage of the maximum available distance, assign a number (from 0 to 100) followed by "%". This syntax is illustrated in Figure 6-5. Length values can be assigned to the region positioning attributes. The length values used in SMIL have the same syntax as length values in HTML.

Attribute: top and left

These are the primary region positioning attributes; they place the top-left corner of an image the specified distance away from the top and left edge of the containing block using either pixel or percentage values. The region will extend to the containing block's right or bottom edge unless constrained by other region positioning attributes.

Attribute: width and height

These are the primary region sizing attributes. The width and height attributes take pixel and percentage values. If `left` is set, then a width pixel value puts the region's right that many pixels over from its left. A percentage value for width sets the region's width at that percentage of the containing

block's width, making the region's right that far from the region's left. The same rules apply for using the height attribute with top to determine the region's bottom.

Attribute: bottom and right

Sometimes explicitly setting the bottom and right attributes is helpful, such as when these edges of the region relate strongly to other positions on the display. Their values state the distance between the region's lower-right corner and the bottom and right sides of the containing block, either as absolute pixels or as percentages of the containing block's dimensions. When used with top and left, the width and height attributes are unnecessary. In the absence of top and left attributes, the width and height attributes are applied to the bottom-left corner positioned by the bottom and right attributes. The value of a bottom or right attribute is ignored if both other region position attribute along the same dimension are assigned.

Value: auto

When specifying values for region positioning attributes, you'll never need to explicitly assign the value "auto". It states that the given dimension is calculated from the region's related dimensions and the containing block, as described above for each attribute. This construct comes from CSS, in which its more complex inheritance issues make the construct more of a concern to authors than in SMIL region positioning.

6.3.7 Background Attributes

You can set a color for the background of a region and you can control what happens to the region when it is not actively displaying content.

Attribute: backgroundColor

The <region> element has a backgroundColor attribute, which takes the same values as the backgroundColor attribute for <root-layout> elements. While we recommend against transparent window background colors for the <root-layout>, a backgroundColor="transparent" assignment is quite useful for regions since it prevents accidental blocking out of other images.

Quick Tip

For many presentations, it is usually advisable to assign a non-transparent background color for the <root-layout> and have all region backgrounds be transparent, allowing either other images or the window background to show through. Since this is the default assignment for the attribute, the most useful region background color is also the easiest to specify!

Attribute: showBackground

If the `<region>` element has a `backgroundColor` attribute, the `showBackground` attribute will determine when this background is shown. If the value is *always*, the background is always shown behind any content in the region. (This is even true when the region may not be referenced by any active media objects.) If the value is *whenActive*, the background will only be painted on pixels not in use with a region actually being referenced by a media object (including any hierarchical child regions.) The default value is *always*, but if you have lots of regions that are used only in certain parts of the presentation, you may want to use *whenActive*.

6.3.8 Stacking and Scaling Attributes

SMIL Layout provides two facilities for controlling the relative placement of media: the `fit` attribute controls how a media object is scaled to its window and the `z-index` attribute controls how windows are stacked relative to each other.

Attribute: fit

The `fit` attribute determines how media is scaled to the size of the region. This is only an issue if the media object is not the same size as the region into which it is being rendered. Figure 6-6 gives a summary of the options available in SMIL 2.0. In all cases, the top left corner of the media object is placed at the top left corner of the region. The value of `fit` determines the method with which images are resized for tailored integration into your presentations.

- `fit="hidden"`: The default value for the `fit` attribute is *hidden*. In regions with `fit="hidden"`, media objects are shown at their original size and shape, pixel by pixel. When an object is too short or narrow to reach both ends of its region, the background color fills the gaps. If the object extends beyond a region border, this extension is *hidden*, or cut off — the media object is cropped. The user cannot see the hidden portions of the object and has no means of accessing any link anchors associated with them.
- `fit="scroll"`: The *scroll* attribute value of `fit` acts just like *hidden*, with one exception: the user is given a scrolling mechanism, such as a scroll bar, to access any part of the media object hidden beyond the region edge. A scroll bar will appear along either the right side or bottom of the region, or both, when the corresponding dimension of the display is too large. Some players may support other mechanisms for scrolling, such as mouse or voice interaction.
- `fit="fill"`: While *hidden* and *scroll* preserve the original shape and size of the media object, the remaining values of `fit` can change the object's size for a tailored fit in the region. The value *fill* changes the size to make the media object fit exactly in the region, with no region background showing.

Original Image:
width="160px"
height="120px"



```
<region  
  backgroundColor="red"  
  ... />
```

	width="100" height="100"	width="200" height="100"	width="160" height="160"
fit="hidden"			
fit="scroll"			
fit="fill"			
fit="meet"			
fit="slice"			

Figure 6-6. The various values of the fit attribute.

As a result, it can also change the rendered shape of the media — if the height/width ratio, known as the *aspect ratio*, of the media object and the region differ, the object will be squeezed and stretched to *fill* the region completely.

- `fit="meet"`: To render the whole media object as large as possible without stretching it, use the *meet* value. This value starts the object at the upper left of the region and increases its size proportionally until it *meets* three of the region boundaries. If the object and region differ in aspect ratio, a bar of background will show along either the region bottom or right. This can be used to generate a *letterbox* effect.
- `fit="slice"`: To fill the entire region without stretching the media object, use the *slice* value. This value starts the object at the upper left of the region and increases its size proportionally until it meets all four of the region boundaries. If the object and region differ in aspect ratio, a piece of either the image's bottom or right will be *sliced* off and hidden from the user.

Note that the manner in which the difference between the bottom and right sides of the media and region is handled varies with the `fit` attribute setting.

Attribute: z-index

SMIL lets you display some visual media in front of others, blocking parts or all of the media further back from view. This effect can be used, for example, to create a montage appearance of photos scattered on top of each other, or to have a background image with other smaller images on top of it. This feature is applied when two or more regions have overlapping positions defined. SMIL handles overlaps by giving all regions *z-index* stacking levels. When two regions overlap, the region with the higher *z-index* “covers” the other with its overlap. The *z-index* attribute's value is either a positive or negative integer, or zero. The higher a region's *z-index* attribute, the higher its stack level, and thus the further forward it appears on the screen. The *z-index* attribute in SMIL behaves as the *z-index* property in CSS, with a few exceptions that relate to timing. (The name *z-index* is used instead of *zIndex* for compatibility with both CSS and SMIL 1.0.)

When two regions overlap, everything shown in the region with the higher *z-index* covers this overlap. This includes any still or moving image displayed in the region, plus any region background color. Transparent areas of the covering region will not block underlying region content. Such transparent areas include transparent region background color as well as transparent portions of images displayed².

² Some SMIL 1.0 players and many SMIL 2.0 Basic players may not be able to support *z-index* processing of high-performance media such as video. The direct use of low-level pixel mapping support may preclude the player's ability to process overlap information efficiently. Always test applications combining video with layered regions on target devices for compatibility.



When overlapping regions have the same z-index value, the region showing the most recently started media item will appear in front of the other. That is, any time a media object is presented, its region obscures any overlap with regions at the same z-index. A object started later in one of these other regions will then cover its overlap area with the first region. If displays begin simultaneously in overlapping regions with the same z-index, then the region defined lexically later in the SMIL code will appear foremost on the screen.

6.3.9 Referencing Regions from Media Objects

Media objects use a single attribute to reference layout regions.

Attribute: region

SMIL Basic Layout defines the `region` attribute as the mechanism used by media objects to reference one or more layout regions. The value of the `region` attribute typically refers to a `<region>` element in SMIL-defined layout based on the `regionName` or `id` (in that order), though it can also refer to the name of a rendering device in a `<layout>` element's alternative format, such as CSS.



6.3.10 SMIL Empty Layout Semantics

Media object elements without a resolved region name are *orphans*. Orphaned media are displayed according to the empty layout rules. This results in rendering orphaned object in the upper-left corner of the browser's main presentation window. Such media is not resized and can potentially fill the entire window, although some browsers may limit the space given to orphaned media to a fixed size (such as 100x100). When you see a media object unexpectedly appear outside the context of a defined region, it may be because it is orphaned.

6.4 SMIL Audio Layout Elements and Attributes

This section discusses SMIL's (minimal) support for audio.

6.4.1 Elements for Audio Layout

The *AudioLayout* module extends the functionality of the *BasicLayout* module's `<region>` element (see page 108) by adding a single new attribute: `soundLevel`. This module is specified separately from other Basic Layout support because some SMIL renderers may not be able to implement sound level manipulation.

6.4.2 Attributes for Audio Layout

The audio layout module defines one additional attribute for the `<region>` element: `soundLevel`.

Attribute: soundLevel

The `soundLevel` attribute allows the author to manipulate the relative output of an audio object. A typical use case is when the audio of a spoken-text object is reduced because a second spoken-text object (such as when audio captions are used) is presented in parallel.

Contrary to what you may expect, this attribute does not allow you to define an explicit output sound level as a decibel (dB) value. Instead, the `soundLevel` attribute takes a percentage value, which tells the player to play the sound at the defined percentage of the volume that would otherwise be used.

Setting the sound level to 100% tells the SMIL player to reproduce the sound at the same level it was saved in the media object. A value of 0% says to play the audio silently — but to still play it! A value of 200% will effectively double the recorded level of the audio, subject to hardware limitations. In all cases, the requested sound level is not the actual level of audio that will be produced via the output device on the player, since the user may independently set the volume control on the computer or on the physical speakers.

The default value of `soundLevel` is 100%. It is strongly recommended that you experiment with several values when explicitly setting the `soundLevel` attribute.

<region>		Profiles			
attributes	regionName				
	top, left, width, height, bottom, right	M	B	1	
	z-index, fit				
	soundLevel				
	showBackground				
	backgroundColor				
	core	M	B	1	
	system test, skip-content				
	customTest				
parents	<layout>	L	M	B	1
children	none	L	M	B	1

Table 6-5. The complete `<region>` element's place in SMIL structure

6.5 Examples Using SMIL Basic Layout

This section provides a number of examples of the use of SMIL Basic Layout. We start with the specification of a typical layout section, then consider how layout elements are referenced from the presentation body and conclude with an example of multi-target layout from within a single presentation.

6.5.1 Defining the `<layout>` Section

There are two cases of interest when defining a layout section in a presentation: layout for presentations with only a single media object active at one time and layout for presentations in which multiple media objects are active simultaneously.

Single Media Layout

Many simple SMIL presentations consist of a single streamed media object. It is tempting to ignore layout issues for these presentations, but this is not a good idea. In order to guarantee that a media object is shown in its (visual) entirety, a basic layout definition should be used. For a 240x180 video, the following layout section could be appropriate:

```
...
<head>
<layout type="text/smil-basic-layout">
  <root-layout id="Example 1" width="240" height="200"
    backgroundColor="black"/>
  <region id="v" top="10" width="100%" height="180" z-index="1"/>
</layout>
</head>
...
```

This will result in letterbox video layout: the layout will fill the width of the root layout and 10-pixel black bands will be shown on the top and the bottom of the player's window. While the root layout window and the video window could be made the same size, the use of the letterbox gives the view the impression that a widescreen video is being shown rather than a postage stamp.

Note that nothing in the layout definition restricts the media type of the contents. Note also that the values for various attributes can be mixed as percentages and pixel offsets. Finally, note that SMIL makes a distinction between the overall presentation window (the <root-layout>) and the rendering regions for a media objects.

Multimedia Layout

The following example shows a layout section that defines the six regions used in Figure 6-4 on page 106:

```
...
<head>
<layout type="text/smil-basic-layout">
  <root-layout id="Example 2" width="640" height="480" />
  <region id="B" left="0" width="640px" top="0" height="480px" z-index="0"
    backgroundColor="red" showBackground="always"/>
  <region id="V" left="30" width="50%" top="90" height="50%" z-index="1"/>
  <region id="C" regionName="G" left="308px" width="257" top="57"
    height="160" z-index="2" backgroundColor="white" />
  <region id="H" left="370" width="55%" top="270" height="80" z-index="1"/>
  <region id="G" regionName="C" left="5%" width="300" top="335"
    height="60" z-index="2" backgroundColor="#cceece"
    showBackground="whenActive"/>
  <region id="T" left="0" width="640" top="400" height="40" z-index="2"/>
</layout>
</head>
...
```

In this definition, the root layout does not have a background color defined, but a separate background region is used (region B). While the body of the presentation will probably have a background image that is shown in this region, a separate background color is also defined — this makes sure that a predictable background is available if the background media object is not available. Since



we always want to see the background, the `showBackground` attribute is defined to be *always*. (This is the default value; we use it for clarity.) The region labelled *G* may be used to hold an overlay graphic. We explicitly define the background to be white. Since the default functionality of *always* is explicitly defined, this white background will be available during the entire presentation, even when the region is not active — this is probably a bad idea. A better approach is the one shown for the region with the `regionName` of *C*: here, a background color is defined which will only be visible when the region is actually used to render the media content. If any part of the content is transparent, then the region's background color will show through rather than the background in the presentation. This usually gives a more controlled result.

Note that the author of this fragment is doing his best to make things interesting for the SMIL player: the region with the `regionName` of *G* has an `id` of *C*; *C* is also the `regionName` of the region with the `id` of *G*. A good SMIL player won't be fooled: a `regionName` will be associated with a region even if it has a different `id`. (Whatever the construction, all `id`'s need to be unique.)

6.5.2 Supporting Multiple Layouts in One Presentation

In most of our examples, one layout section is defined that describes the layout geometry of the presentation. For some applications, supporting multiple layouts within the presentation is appropriate. You may want to have one layout that describes how a mobile client presents its information when it is docked and one for when the same presentation is accessed when the client is in its mobile mode. Another example is the layout approach used by the MMS mobile clients: two layouts are defined, one for landscape devices and one for portrait devices.

SMIL allows multiple layout specifications to be given and to be wrapped in a SMIL `<switch>`. We give an example of this in Chapter 9: *Content Selection and Control*.

6.5.3 Referencing Media in the Presentation

Once a layout section has been defined, the media content in the presentation can be directed to the appropriate regions using the `region` attribute:

```
***
***
<par id="Story4" >
  <video id="Studio4" region="v" src="rtsp://.../studio4.mpg"/>
  
  
  <ref id="Caps4" region="C" src="MkyCaps.txt" systemCaptions="true"/>
</par>
***
```

This fragment shows how media elements can send multiple sets of information to the layout objects based on the timing structure of the presentation. The layout definition knows nothing about the presentation timing and the presen-

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
    "http://www.w3.org/TR/REC-smil/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
<head>
  <layout type="text/smil-basic-layout">
    <root-layout id="what Cheer" width="640" height="480" />
    <region id="B" left="0" width="640px" top="0" height="480px" z-index="0"
      backgroundColor="red"/>
    <region id="V" left="30" width="50%" top="90" height="50%" z-index="1"/>
    <region id="C" regionName="G" left="308px" width="257" top="57"
      height="160" z-index="2" backgroundColor="white"
      showBackground="whenActive"/>
    <region id="H" left="370" width="55%" top="270" height="80" z-index="1"/>
    <region regionName="C" left="5%" width="300" top="335" height="60"
      z-index="2" backgroundColor="#cceeee" showBackground="whenActive"/>
    <region id="T" left="0" width="640" top="400" height="40" z-index="2"/>
  </layout>
</head>
<body>
  <par id="NewsMaster">
    
    <ref id="ticker" region="T" src="feed.txt" />
    ...
    <par id="Story4" >
      <video id="Studio4" region="V" src="rtsp://.../studio4.mpg"/>
      
      
      <ref id="Caps4" region="C" src="MkyCap.txt" systemCaptions="true"/>
    </par>
    ...
  </par>
</body>
</smil>

```

Example 6-1. SMIL code for Figure 6-4's layout.

tation timing does not know anything about the presentation layout: the two sections are orthogonal in terms of functions and features.

We can combine the layout and the referencing sections together into one presentation. Example 6-1 shows how the news presentation developed in Figure 6-4 could be implemented using SMIL Basic Layout.

6.5.4 Adjusting Rendering Properties

The following fragment shows how layout attributes can be used on SMIL media elements to control presentation rendering behavior:

The layout region *V* is given a size of 640 wide by 480 high. The default scaling algorithm specified with the *fit* attribute is *fill*, which means that the content in the presentation will be stretched in both directions to fit the display space.

In the body section, three media objects are presented in sequence. The first is a picture of a famous movie actor. While famous actors often like the idea of a face lift, they typically don't like to have their cheeks spread wide or their noses pulled down, so the default *fit="fill"* behavior will probably get you into trouble. Instead, set the *fit* value to *meet*: this scales the image to fill the space,

```

***
<head>
  <layout type="text/smil-basic-layout">
    <root-layout id="Example 3" width="640" height="480"
      backgroundColor="red"/>
    <region regionName="v" width="640px" height="480px" fit="fill"/>
  </layout>
</head>
***
<seq id="FilmReviewsAndNews">
  
  <video id="Second" region="v" src="movie.mpg" fit="hidden" />
  <text id="Third" region="v" src="review.html" dur="45s"/>
</seq>
***

```

Example 6-2. Adjusting layout from the presentation <body>.

but it preserves the original aspect ratio of the media. Note that this actor's contract apparently specifies a particularly flattering background color, so the media object also explicitly sets the `backgroundColor` attribute.

The second media object is the film itself. Because video is expensive to reshape in real time, it is advisable to keep the object at its original size. You can do this with a `fit` override value of *hidden*. If the video is smaller than 640x480, the background will show. In Chapter 12: *Advanced Layout Topics*, we show how advanced layout can be used to center the media in a window; this is not supported in basic layout.

The final media object is an HTML review. Here, a scaling option to fill the space is appropriate.

6.5.5 Rendering Multiple Copies of a Single Media Object

In addition to manipulating the size of a media object, SMIL also lets you target multiple regions from a single media object. The `regionName` attribute is the key to this functionality. Consider the following fragment:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
  "http://www.w3.org/TR/REC-smil/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <layout type="text/smil-basic-layout">
      <root-layout id="Player-Window" width="1280" height="960"
        backgroundColor="red"/>
      <region regionName="v" width="640px" height="480px" fit="hidden"
        top="0" left="0" />
      <region regionName="v" width="640px" height="480px" fit="hidden"
        top="50%" left="0" />
      <region regionName="v" width="640px" height="480px" fit="hidden"
        top="0" left="50%" />
      <region regionName="v" width="640px" height="480px" fit="hidden"
        top="50%" left="50%" />
    </layout>
  </head>
  <body id="wallOfTVs">
    <video id="RobsFavorite" region="v" src="movie.mpg" fit="hidden"/>
  </body>
</smil>

```

In this infamous example, multiple regions have the same name (because they use the `regionName` attribute), so the media object is displayed in all the regions simultaneously. If, for whatever reasons, the player is unable to present the media object in all these regions, then the object will be shown in the region that was defined first in the SMIL code. For this reason, make sure the most important region for a multiply displayed media item is defined first in the code.

6.6 Summary and Conclusion

For the most part, SMIL layout defines the spatial arrangement of visual media on the screen. It establishes the windows used in a presentation, and what media gets shown in them. Within these windows, SMIL layout sets up regions for the different spaces media gets shown in. Layout constructs can adapt the size and shape of images to fit best in these allocated spaces. While elements from the SMIL Layout Modules typically hold the attributes for this positioning and adjustment, media object elements can assign these as well for further fine-tuning of individual image displays. In addition to this placement of visual media, SMIL layout also controls the relative volume of audio media played in a presentation.

The facilities discussed in this chapter form a base for layout functionality. SMIL 2.0 also supports a number of advanced layout features that make media positioning easier and more predictable. They also allow layout regions to be structured as a hierarchy of regions. (This is useful if you want to animate the position of multiple objects in a coordinated manner.) Finally, advanced layout also allows you to selectively create new regions dynamically on the media object reference line for increased positioning control. All of these features are discussed in Chapter 12: *Advanced Layout Topics*.

6.7 Further Resources

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>.

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>.



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.

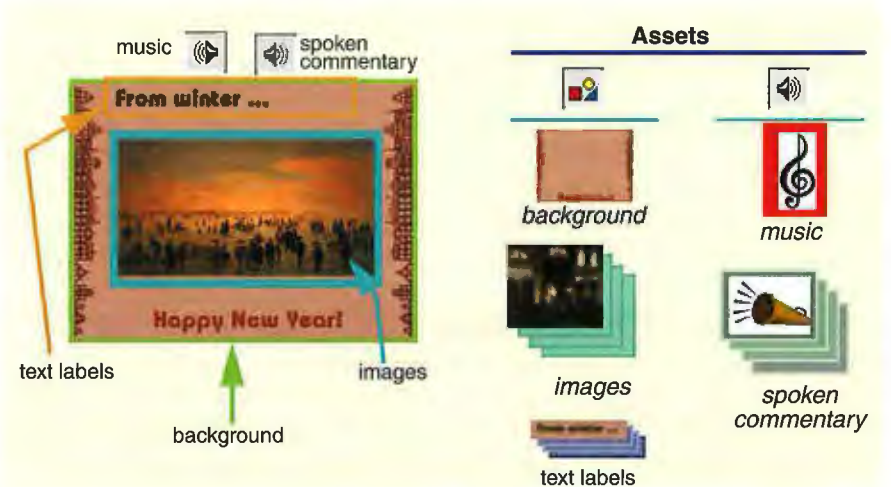


Figure 7-1. Happy New Year

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*.

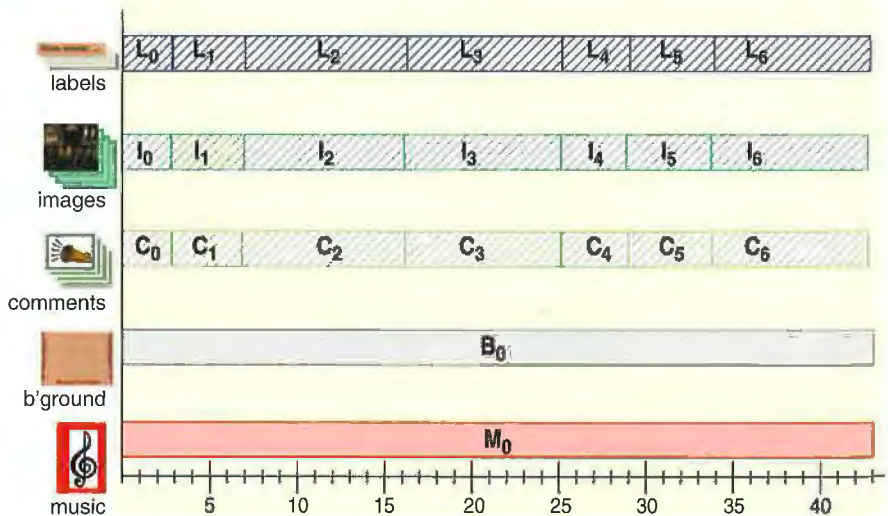


Figure 7-2. A timeline representation of the presentation in Figure 7-1.

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.

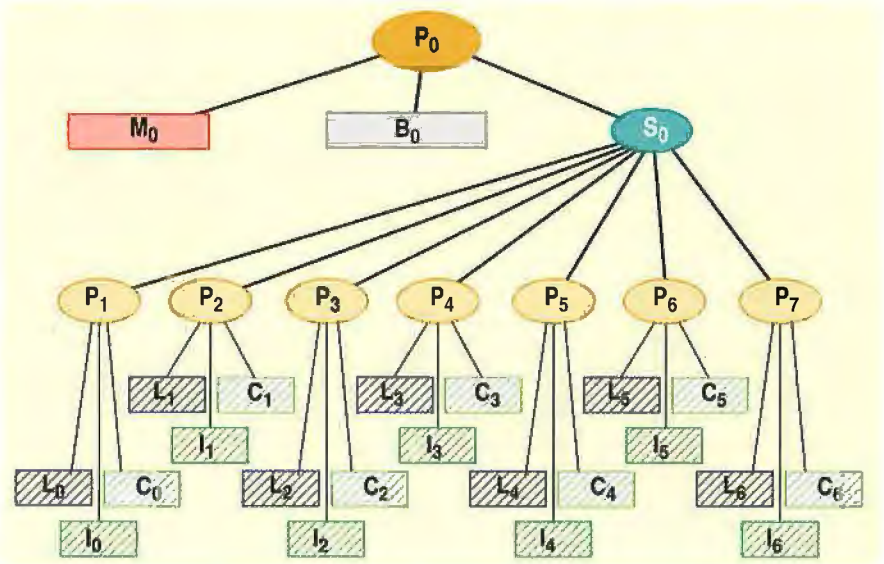


Figure 7-3. The SMIL hierarchy for Figure 7-1.

```

<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.

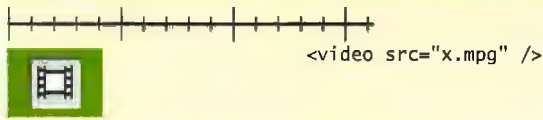
7.1.3 Durations, Time and Timebases

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

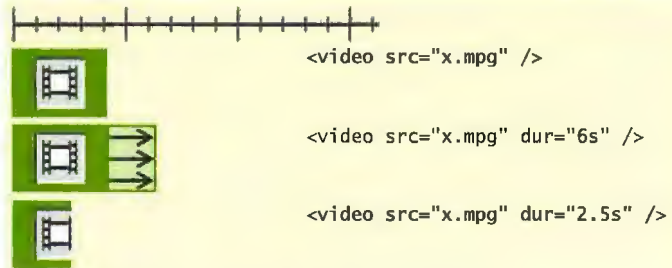
Most media formats require that the duration of all of the component media objects are explicitly defined. SMIL has several attributes that support direct duration definition, but it also provides attributes that allow you to specify or limit several layers of logical object durations. These layered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

(a)
**Implicit
Duration**



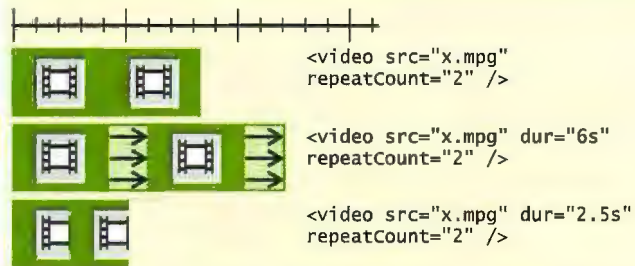
In this example, the *implicit duration* is 4 seconds.

(b)
**Simple
Duration**



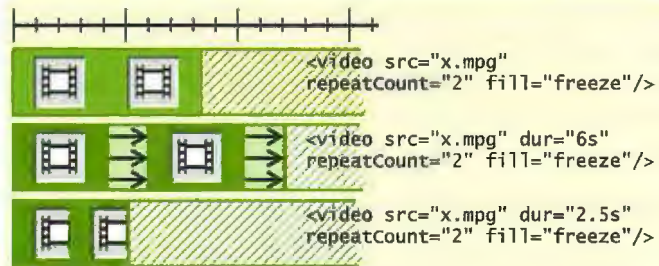
Implicit Duration modified by *dur*.

(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 *synchbase*. (Synchbases 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     
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 `<exc1>` 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*.

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.

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

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

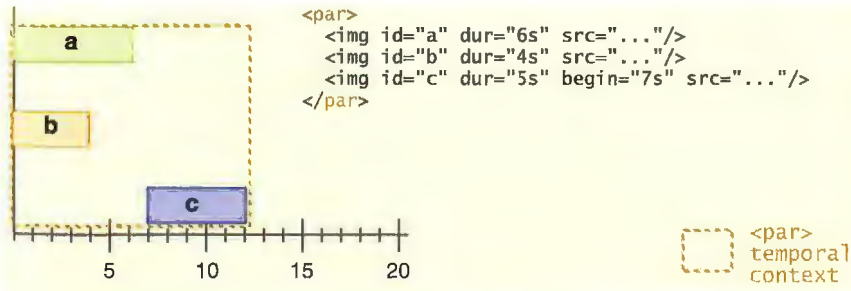


Figure 7-5. SMIL's `<par>` timing element.

The basic timing structure of the <par> is illustrated in Figure 7-5. The default synchbase 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 synchbase 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.

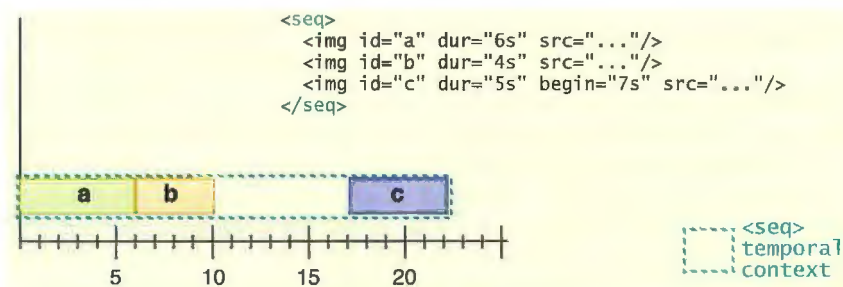


Figure 7-6. SMIL's <seq> timing element.

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

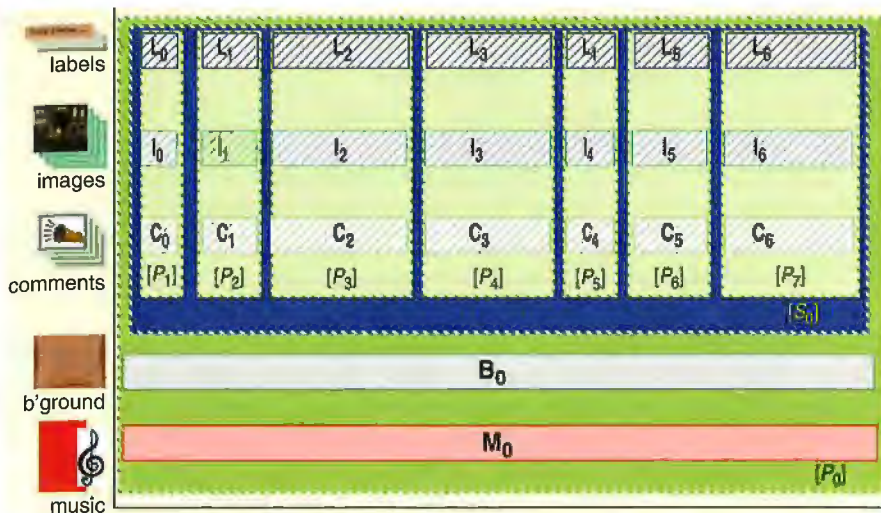


Figure 7-7. A structured timeline representation of the presentation in Example 7-2.

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 synchbase 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.
- *SynchbaseTiming*: The *SynchbaseTiming* module allows the synchbase 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 synchbase 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



General Timing Control

- begin
- end
- dur

Repeat Behavior Control

- repeatCount
- repeatDur

Object Persistence Control

- fill
- fillDefault

Extended Timing Control

- endsync
- min
- max

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, $\text{begin} + \text{dur} = \text{end}$, 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.

attribute	value	elements	Profiles
begin, end	begin/end	<prefetch>	
		<a>, <area>	
		<body>, <seq>	L M B 1 H
		<par>	
dur	clock value	media object, <brush>	
	"media"		
	"indefinite"		

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 synchbase 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 synchbases. SMIL allows the synchbase used in the presentation to be explicitly defined for `<par>` elements by specifying an ID of the synchbase, and specifying a timing qualification of that synchbase (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* synchbase.) Note that SMIL 1.0 used a slightly different syntax for specifying synchbases; this format has been depreciated in SMIL 2.0.

In the SMIL Language profile (including SMIL Basic), explicit synchbases are rarely used in simple documents, but for other host languages (like those supporting the XHTML+SMIL profile), the use of explicit synchbases 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 synchbase 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 synch-

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:

```


```

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:

```

```

(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>
    
  </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       
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 synbase is defined, the end attribute modifies the end time relative to the default synbase of the element, not the duration of the media object.

Consider the SMIL fragment in Example 7-4. The default synbase 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 synbase (which is the beginning of the <seq>)
- the video element on line 5 ends one hour and 15 seconds after the start of its synbase (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 synbase (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 synbase.

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 synbase.

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 synbase for the children of this <par>. This means that:

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

- the video element on line 12 ends one hour and 15 seconds after the start of its synchbase (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 synchbase.

These examples all relate to a single video object, but the same synchbase-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 
5 <par id="inner" dur="15s">
6 
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.

attribute	value	elements	Profiles
min	clock value	<prefetch>	L M B
max		<a>, <area> animation <body>, <seq>	
endsync	"last", "first", "all"	<par>, <excl> media object, <brush>	1 H

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" ... />
  
</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.



attribute	value		elements	Profiles
repeatDur	clock value		<a>, <area> animation <body>, <seq> <par>, <excl> media object, <brush>	L M B 1 H
repeatCount	decimal number greater than "0.0"	"indefinite"		

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 
2 
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 times 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>
```

Combining 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 unpredictability, 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” time-base 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 `<exc1>` 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 experimenting 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.

attribute	value		elements	Profiles									
min	clock value	"media"	"0"										
max		"indefinite"											
dur													
begin, end													
repeatDur	begin/end												
repeatCount	decimal number greater than "0.0"												
endsync	"last", "first", "all"											"media"	IDREF to child

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



8

Basic Linking



Of all of the features and facilities available across the Web, probably the single most important is *linking* (or, as it is sometimes known, *hyperlinking*). Linking is the ability to define and activate a non-linear navigation structure within and across electronic documents. By following a link, you select the ordering of content that best meets your needs.

The popularity of HTML and the World Wide Web have given the concepts of linking broad acceptance and a place in technological history that Vannevar Bush probably never could have imagined when he first published the basic concepts of linking more than 50 years ago¹. SMIL provides support for all of the basic HTML linking functionality that most Web users expect, meaning that SMIL authors can quickly integrate linking into their presentations. SMIL also provides a set of extensions to HTML linking that provide support for links in a temporal environment. In addition, SMIL's linking facilities provide a way to better separate the definition of the link's components from the actual media contents; this can make linking more manageable in a dynamic Web environment. Finally, SMIL also provides features that differentiate the two main uses of linking constructs: links for navigation and links for conditional activation of content.

This chapter explores the basic linking facilities in SMIL. We start with a review of linking concepts. We then briefly relate these concepts to how linking is supported in HTML and extend the discussion to cover the concepts available in SMIL 2.0.

8.1 An Overview of Linking Concepts

In its simplest form, a link is a pointer. It defines an address of a document and, optionally, an offset within that document. Of course, to be useful, the user needs to know that the link exists. This means that a mechanism is required in the originating document to indicate that a link is present. This is called a (source) *anchor*. If the link points to anything other than the beginning of the target document, then a (destination) anchor into the target is also required.

¹ V. Bush, "As We May Think," The Atlantic Monthly, July 1945.

There are many ways to implement links and anchors. In a conventional book, you can create an anchor by circling some text or an illustration and then create a link by writing an address — such as the name of another book and a relevant page number — in the margin. When you follow this cross-reference type of link, you will replace one document with another. (You may want to turn down the corner of the page you are currently reading so that you can return to the source place easily; doing so creates a destination anchor for the way back.) Another model for a link is the footnote²: you can point to additional material that is not considered mainstream, with the expectation that the reader returns to the main flow after reading the note. A glossary is an example of a multi-source link, while an index is an example of a multi-destination link.

When circling text and scribbling addresses, the anchors and links were placed directly in the source material: the book. This is handy if you are the only user of the book, but it can also be a problem if the book has many users — or if the book isn't yours. In this case, it is sometimes useful to construct a separate list where you can define source and destination anchors and the associated links without corrupting the original document.

All of these issues and concepts also relate to electronic links, including those used in HTML. SMIL extends these ideas with *timed links*: links in which the source anchor may only be active for a limited period of time (say the first eight seconds of a presentation), or in which the target of the link may change during the rendering of the media object. Adding the concept of time to links provides many new possibilities — including auto-firing of a link after a fixed period of time, or limiting the amount of data that is rendered when a link is followed.

Before looking at the intricacies of timed links, we will start with a brief review of non-temporal links as implemented in HTML. We then consider non-temporal and temporal links in SMIL.

8.1.1 Links in HTML

The use of links with text-based documents is usually called *hypertext*. Text provides a relatively simple linking environment: you can easily insert the link address and the source and destination anchors in the text document using a text editor. The browser then needs to be able to identify those places where a source anchor exists, and bingo: you have links!

The most prominent uses of links in hypertext documents are:

- *links for navigation*: by following a link, you are taken from one content 'focus' to another. The link is typically uni-directional, but a two-way link can also be defined. The source focus is usually replaced by the link, but the destination can also augment the source.

2 In a footnote, the source anchor is the footnote symbol or number. The address in the link is typically implied: either at the bottom of the page, at the end of a chapter or at the end of the document. A second link, taking you back to the original text, is implied at the end of the footnote.

- *links for activation*: by following a link, a follow-on part of a presentation is started. This can be used when structuring the presentation of lengthy, complex information. Activation links also provide a means of presenting alternative content with one local presentation flow.
- *links for message passing*: when following certain links, it is sometimes useful to present state information to the destination document. Both navigation and activation links can be supplemented with parameters provided by the user (or the user's browser).

Of these, the use of links for navigation (with or without message passing) is the most common form of HTML's hypertext links.

In the early days of the Web, "following a link" always meant replacing one page with another. It was only later that a new window could open on link traversal, and even later that the notion of a *frame* was introduced that allowed a destination to be placed within a relatively constant common context.

While the concepts of linking are pretty straight-forward, there are many approaches to hyperlink implementation that have been presented and (passionately) defended over the last fifty years. Since SMIL linking is nearly the same — but not identical! — to HTML linking, we will concentrate on their similarities and differences in the following sections.

Embedded Anchors and Links in HTML

One of the things that made HTML linking a huge success was its simplicity. An HTML `<a>` element is used to define both anchors (source and destination) and link addresses. Example 8-1 shows a simple HTML document that illustrates the basic concepts of HTML links. An HTML source anchor defines a span of content that is placed between the `<a>...</a>` elements (see lines 3, 4 and 5). The source anchor can span one character or an entire document, but it cannot overlap other source anchors.

An HTML destination anchor is defined by the 'name' attribute in the `<a>` element (line 6). The destination anchor may span a range of content (line 6), or it may identify a single point in the document (line 7).

```

1 <HTML>
2 <Body>
3 <P>Here is an <A href="#nearTheEnd"> anchor </A>.
   It points to a location within this document.</P>
4 <P>This is another
   <A href="http://www.w3.org/AudioVideo">anchor</A>.
   It points to an external document located at www.w3.org.</P>
5 <P>Here is a link containing a reference to a destination anchor
   <A "http://www.w3.org/TR/smil20/extended-linking.html
     #SMILLinking-Introduction">fragment</A>.
6 <P><A name="nearTheEnd">This</A> is the definition of a
   destination anchor.</P>
7 <A name="theEnd"/>
8 </Body>
9 </HTML>

```

Example 8-1. An HTML document showing link concepts.



An HTML link address can point to a destination anchor within the source document, a complete external document or a destination anchor within an external document. (Lines 3, 4 and 5 give an example of each type.) The link address is specified by the HREF attribute and consists of a URI and a fragment identifier, joined by the '#' connector. If the URI is not present, then the current source document is assumed. If the fragment identifier is not present, then the beginning of the document is assumed. If neither the URI or the fragment identifier are present, the beginning of the current document is also assumed.

External Anchors and Links in HTML

All of the anchors and links in Example 8-1 are specified within the source HTML document. If you want to add new links to the document (or replace old ones), you need to explicitly edit its content. While this has proven to be a simple and popular approach, it limits the reuse of the HTML text object. If the document being extended is not a shared resource, adding links means making a separate copy. This assumes, of course, that you are allowed to copy and modify the text. (For third-party audio and video objects, this is rarely the case.) It also makes it difficult to add links to media types that are not easily changed with a text editor.

Creating links in non-textual (or protected) media requires a facility for separating the anchor/link definition from the media object associated with the source anchors. (X)HTML supports external link specification via the `<map>` and `<area>` elements, as shown in Example 8-2. Line 1 references an image and uses the `<map>` named `mapNP` to define the various source anchors and link addresses that are active when the image is rendered. The browser will map the coordinates in each of the `<area>` specifications as overlays on top of the image and associate the link addresses with each area. Note that the shapes of the areas need not be rectangular. Note also that several images can reference the same map, but that areas within a map may not overlap.

```
1 <p></p>
2
3 <map name="mapNP">
4   <area href="overview.html#start"
5       alt="General overview"
6       shape="rect"
7       coords="0,0,123,45">
8   <area href="overview.html#quiz"
9       alt="Did you get it?"
10      shape="rect"
11      coords="111,0,675,45">
12   <area href="review.html"
13       alt="A Reneral Review"
14       shape="circle"
15       coords="111,99,30">
16   <area href="summary.html"
17       alt="Final words"
18       shape="rect"
19       coords="111,150,401,45">
20 </map>
```

Example 8-2. External links via `<map>` and `<area>`.

8.1.2 SMIL: Adding Time to Links

SMIL has taken many of the concepts of HTML linking, including the <a> and <area> elements. While these general concepts have been reused, SMIL provides its own specification of the <a> and the <area> elements within the SMIL namespace. This is because SMIL's model of separating content from document structure has required some changes to how the <a> and <area> elements have been defined.

SMIL defines a timing and synchronization model that can be used to influence many of the objects in a presentation. Links are no exception: the active period of an anchor and the temporal effects of following a link can be controlled directly or indirectly by the timing parameters associated with the object.

Time, Linking and Structure in SMIL

It is tempting to think about linking in a timed environment as simply a matter of jumping to a particular time moment in a presentation. For example, consider the newscast video shown in Figure 8-1. Let's suppose you want to add anchors and links so that you can jump from story to story in the video. If you add a source anchor to the video element at times t_{s1} through t_{s11} , and define a destination anchor at time t_d , following the link becomes a straight-forward process of moving the presentation's time pointer to t_d at the moment the link is activated.³

Upon closer inspection, however, the process is anything but simple. First, what we really want to do is to link *story 1* to *story 2*, and not necessarily the interval t_{s1} through t_{s11} to t_d . (Today's *story 1* may be 28 seconds long, but *story 1*

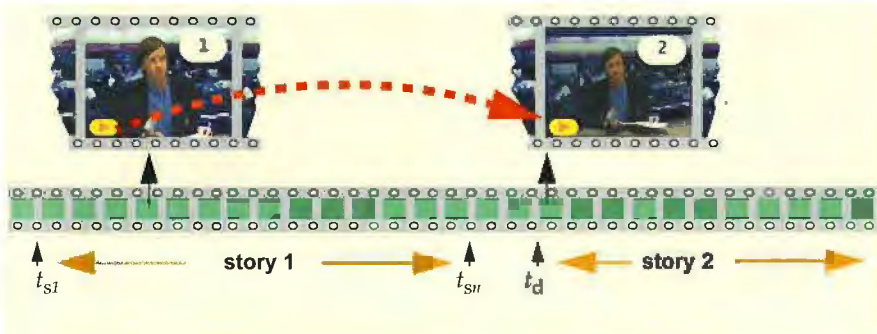


Figure 8-1. Links in a single media object.

³ In a streaming environment, this is not so simple: if the destination anchor has not been buffered at the player, there will be a delay in the presentation until enough of the video is sent from the server to allow streamed playback. This can happen for both forward and backward links. Linking backward in a 'live' video stream is theoretically possible, but jumping forward in a live stream presents some serious (meta)physical problems that are outside of SMIL's scope!

tomorrow may be 90 seconds long!) Also, when we navigate in the video, we also navigate along the sound track. Our video encoding has encapsulated sound, but if it had multiple sound tracks — or an animated ticker-tape running under the image — each would need to be repositioned independently.

SMIL provides the multimedia author with unprecedented flexibility in defining a reusable link architecture. It does this because SMIL links are based on the presentation's structure, not a single timeline. This flexibility means that the same SMIL document, with the same linking definitions, can be used for many different versions of the document. In our news example, the logical linking structure of navigating from *story 1* to *story 2* is totally independent of the duration of each of the stories. It also means that any other media object active will also be linked, so that a consistent presentation state is preserved.

8.2 SMIL Basic Linking Elements

This section describes the two elements defined in the *BasicLinking* module. They are: `<a>` and `<area>`.

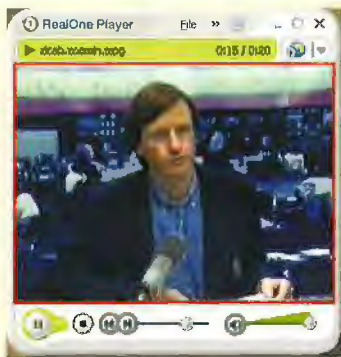
Element: `<a>`

The `<a>` element's structure is laid out in Table 8-1. Like HTML, SMIL uses the `<a>` element to define a source anchor and a link address. Unlike HTML, SMIL does not use the `<a>` element to define a destination anchor — instead, the XML ID of the target object is the specification for the destination. The link address is a URI with an optional fragment name connected with a '#'; it is essentially the same as in HTML.

SMIL uses the `<a>` element as a *whole-node anchor*: it acts as a wrapper around the associated object. An example of SMIL's use of the `<a>` element is shown in Figure 8-2. When the media object *Video1* is played, the entire object is wrapped with an anchor containing a link to the destination anchor named *part1* of another SMIL presentation named *Example.smil*. Note that when the link is followed, default SMIL behavior is that the entire source presentation is replaced with the destination presentation — not just the element that contains the source anchor.

<a>		Profiles					
attributes	LinkingAttributes						
	href						S
	begin, end, dur, repeat timing, min, max	L	B	M	1	H	
	system test, customTest						
parents	<body>,<priorityClass>	L	B	M	1	H	
parents & children	time container	L	B	M	1	H	
	<switch>						
children	media object, <brush>		B	M	1		
	<prefetch>	L					H
	animation						S

Table 8-1. Structure of the `<a>` element.



anchor:
entire media item

active period:
while media is active
(unless modified by a
begin/end attribute)

```

1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language" >
2   ***
3   <a href="http://www.example.org/Example.smil#part1">
4     <video src="rtsp://www.example.org/Video1.mpg" ... />
5   </a>
6   ***
7 </smil>

```

Figure 8-2. Use of the <a> element in SMIL.

Element: <area>

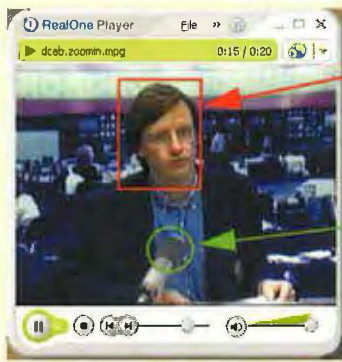
The <area> element defines a source anchor and a link address. Just as SMIL's <a> element (and unlike HTML), the destination anchor associated with the link is the XML ID of the target object; this may be an ID in the same document or an external URI with optional fragment name. The structure of the <area> element is shown in Table 8-2.

<area>		Profiles					
attributes	LinkingAttributes						
	href						S
	begin, end, dur, repeat timing, min, max	L	B	M	1	H	
	system test, customTest						
parents	media object	L	B	M	1	H	
children	none.	L	B	M	1	H	

Table 8-2. Structure of the <area> element

Where SMIL's <a> element is a whole-node anchor, the <area> element defines an anchor that applies to only a part of the media object. This 'part' can be spatial (like the HTML <area> element), but it can also be temporal: <area> can be used to define an anchor that is only active for a part of an object's duration. Unlike HTML, the <area> element is not defined as content of a <map> element, but as content of a media object. The <map> element does not exist in SMIL.

An example of SMIL's use of the <area> element is given in Figure 8-3. The <area> a1 provides conventional HTML functionality in a SMIL document; it is active during the entire active duration of the media object. The <area> a2 is



anchor a1: active period:
head while media is active

anchor a2: active period:
microphone 5 seconds,
starting at
10th second

```

1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language" >
2   ...
3   <video src="rtsp://www.example.org/video1.mpg" ... >
4     <area id="a1" href="http://www.example.org/Link1.smil"
5       shape="rect" coords="110,10,180,100" />
6     <area id="a2" href="http://www.microfoonz.org/Model1604.smil"
7       shape="circle" coords="160,177,20" begin="10s" dur="5s"/>
8   </video>
9   ...
10 </smil>

```

Figure 8-3. Use of the <area> element in SMIL.

only active for the five seconds starting at the 10th second of the video. If the video is shorter than 10 seconds, a2 is never active.

In order to appreciate the structured and temporal linking options that SMIL presents, consider the two methods of defining a linked-news structure. The first method uses a single video object in parallel with a sound track, while the second method uses multiple video/audio object combinations.

Figure 8-4 shows a video containing two timed anchors, each of which is defined in an <area> element. In both cases, the entire viewing area is used for the anchors. If the link is activated within the first 28 seconds, the video presentation is set so that rendering continues from the 36-second point. Not only the video's rendering time is adjusted, however: the accompanying audio track is also set so that it renders from the 36-second point onward. (The reason for this is explained below.) The activation times are hard-coded into the presentation. This approach is useful if the video object is static — that is, if the same video object is used every time the presentation plays.

Figure 8-5 shows another approach. Here, the link addresses are not set to the XML ID's of the video, but to structure containers containing logical combinations of the presentation. The advantage of this approach is that the timing of each anchor is independent of the document structure. The entire presentation's structure can be reused with new media objects without change. When combined with SMIL's content control facilities, this approach becomes even more powerful. By wrapping a <switch> element around the audio, you could support multi-lingual audio or a combination of audio and text captions, without having to adjust presentation timing.

(a) Navigation within a media object.

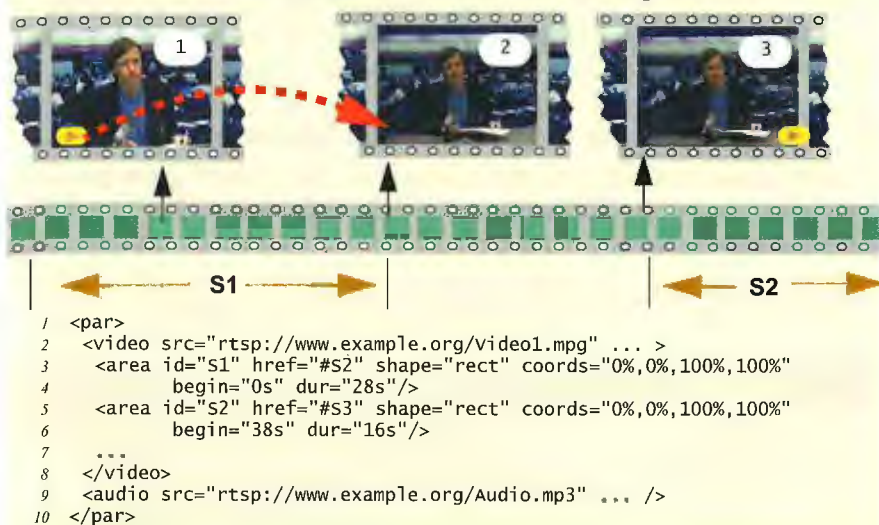


Figure 8-4. Linking strategies in SMIL: Linking within an object.

(b) Navigation across media objects.

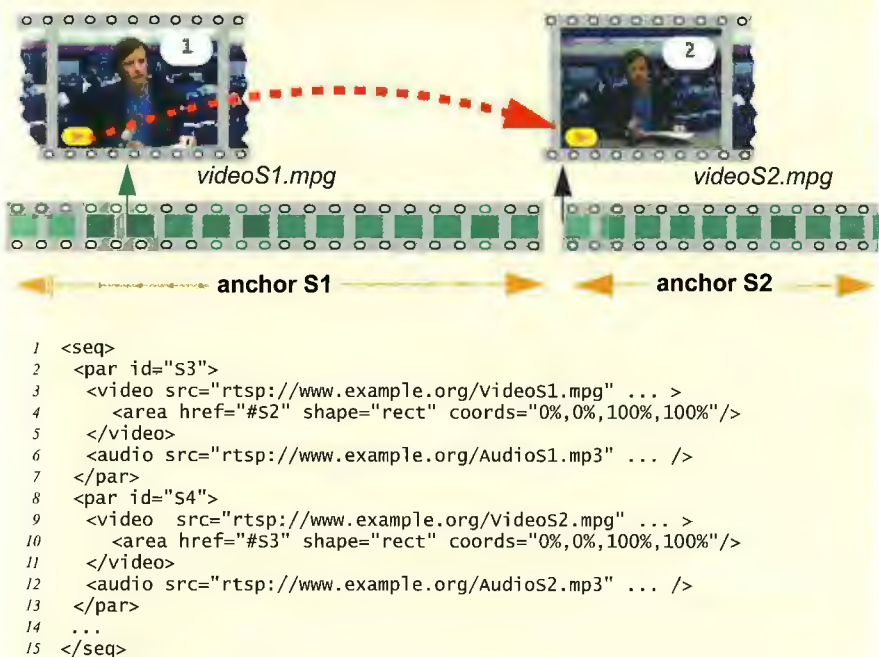


Figure 8-5. Linking strategies in SMIL: Linking across objects.

Element: <anchor>

SMIL 1.0 used the <anchor> element to specify partial node anchors. This element is still available in SMIL 2.0, but since the <area> element offers the same functionality, the <anchor> element has been depreciated.

8.2.1 SMIL Basic Linking Attributes Architecture

As enumerated in Table 8-3, SMIL defines a range of link-related attributes. These attributes define how a link is activated, when it is activated, where it is activated, and the impact of link activation on the source and destination presentations. Each of these aspects are addressed in this section. The attributes apply to both the <a> and <area> elements, unless otherwise noted.

attribute	value	elements	Profiles
href	URI	<a>, <area>	M B 1 H S
sourcePlaystate	"play", "pause" "stop"		
destinationPlaystate			
show	"new", "replace"		1
external	"true", "false"		
target	name of display environment		
sourceLevel, destinationLevel	non-negative percentage, "100%"		

Table 8-3. Linking attributes overview.

8.2.2 Specifying the Activation Period of Source Anchors

SMIL timing can be used to restrict the active period for a link construct. In Figure 8-4, the <area> element with ID S1 specifies that the associated anchor is active during the first 28 seconds of the video (line 4) and the <area> element with ID S2 specifies that its anchor is active for 16 seconds, starting 38 seconds into the video object (line 6). Outside this period, no anchors are active.

SMIL allows the definition of a link activation period on the <a> and <area> elements. Both types of anchors are timing-transparent: they do not influence the timing of the SMIL presentation.

No new linking attributes are required to support this behavior; the <area> elements accepts all of SMIL's basic timing attributes.

8.2.3 The Temporal Moment of the Destination Anchor

Where specifying the duration of a source anchor is usually straight-forward, specifying the behavior of following a link to a destination anchor can be very complex. This complexity arises out of the nested time structure of SMIL documents: when a destination object is activated, then all of the other media objects

that are also specified to be rendered must also be activated. The details of such activation are actually part of the chapter on SMIL timing, but the basic rule is:

When a link is specified to a point in a destination document that is not the (temporal) beginning of that document, then the link needs to be implemented as if it were a 'fast-forward' across the effective timeline of the target document up to the destination anchor. All parts of the presentation that would be active if the presentation had reached destination anchor without linking must also be present (and in the same state) if it reaches this point via linking.

This statement looks pretty simple, but it has tremendous temporal consequences for the SMIL presentation scheduler. Since not all of the timing attributes of each of the media objects and the SMIL `<par>/<seq>` structure are defined statically, the scheduler needs to compute the effective state of the presentation so that all of the media items (and the progress into those items) are displayed just as if the document had been allowed to be played from the beginning. If this computation is not done efficiently, it can severely impact the duration of following a link!

The rules for determining the temporal moment of the destination of the link can be summarized as follows:

- *Destination is already active:* If the destination of the link is already active, the element is effectively restarted. The temporal moments of all of its peer and ancestral elements are also effectively reset based on the begin time of the link destination.
- *Destination is inactive, begin time is resolved:* If the begin time of the destination is resolved (that is, if it has a defined begin time because of explicit or implied timing semantics or if the begin time was resolved based on an earlier traversal of the link), then the presentation seeks to that temporal moment. The temporal state of the peers and ancestors is also realigned to match the temporal moment of the link destination.
- *Destination is inactive, begin time is unresolved:* This is the most complex case. If the destination is not active and if it has an unresolved begin time (it is waiting on an event or it has `begin="indefinite"` defined), the scheduler has to bubble up the document tree to find the first resolved parent. This time is applied to each of the children until the link destination is reached. The resolved begin time is applied to the link destination and it is saved by the scheduler for potential reuse in resolving this link (or any link that targets a child of the destination) in the future. The pseudo-resolved begin time for the link may be reset if a parent restarts or if some other condition exists that causes a new effective begin time to be defined.

There are special case rules that are applied for interactive timing. Since these rules require more background on SMIL events, we return to this topic in *Linking, Events and SMIL Timing* on page 290.



8.2.4 Attributes for Link Anchor Geometries

The `shape` and `coords` attributes can be used with `<area>` to specify geometry.

Attribute: shape

The `<area>` element has — as in HTML — a `shape` attribute, to specify the shape of an anchor. Permissible values are: *rect*, *circle* and *poly*. The `shape` attribute is applied to the underlying media object after this object has been scaled (based on SMIL Layout constructs) but before the media is clipped onto the viewing region. This can result in part of the shape being set outside the viewable area of the media.

Attribute: coords

The size and position of the anchor defined by the `shape` attribute is determined by the `coords` attribute. They are similar to HTML's `shape` and `coords` attributes.

The mapping of the coordinates given in the `<area>` to the source image and the containing regions occurs as follows:

- 1) The origin of the coordinate space for the anchor (that is, the 0,0 point) is placed at the upper-left corner of the image, not of the region into which the image is rendered. If the media object has been offset within the region (by using SMIL Layout's sub-region positioning), care needs to be taken to make sure that the anchor is correctly specified relative to the object. If a 320x240 media object is placed at an offset of 15 pixels from the top and 30



```
1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language" >
2 ...
3 <region id="r1" width="360" height="270" fit="hidden" />
4 ...
5 <video src="rtsp://www.example.org/Video1.mpg" region="r1"
6     top="15" left="30">
7     <area id="a1" href="http://www.example.org/Link1.smil"
8         shape="rect" coords="110,10,180,100" />
9     </video>
10 ...
11 </smil>
```

Figure 8-6. Offset origin of the `coords` attribute.

pixels from the left, the center point of the anchor's coordinates (160x120) will actually be placed at a region offset of 190, 135 in region space. This will normally not present a problem unless you want to overlay another object within the same visual space or if the image is scaled by the player for rendering. (See Figure 8-6.)

- 2) In order to remain consistent with HTML, SMIL defines that the pixel values used in the <area> correspond to the resolution of the display device (usually the screen) rather than to the media object's dimensions. Thus, if a 320x240 video is scaled to a 300x200 region of screen pixels with *fit="meet"* (that is, the entire object is scaled proportionally so that the full media object is visible), the coordinates of the image's center should be (133, 100) and *not* (160,120). (See Figure 8-7.)

region r1:
region size: 300x200
fit: meet

video:
original size: 320x240
scaled size: 266x200



anchor a1:
'coords'
(in region space):
 92, 8, 180, 83

effective offset
(in media space):
 110, 10, 180, 100

```

1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language" >
2 ...
3 <region id="r1" width="300" height="200" fit="meet" />
4 ...
5 <video src="rtsp://www.example.org/Video1.mpg" region="r1" >
6   <area id="a1" href="http://www.example.org/Link1.smil"
7     shape="rect" coords="92,8,180,83" />
8 </video>
9 ...
10 </smil>
  
```

Figure 8-7. Display scaling and coords behavior.

- 3) If a portion of an image is not visible (because it is larger than the region size and the image is not scaled), it may be possible to specify a range of pixels that is outside the region. A typical case of this happening is when percentage values are used to specify the coordinates in the <area>. If this happens, the anchor is defined relative to the image (because of rule 1), but since it is outside of the display area, these pixels cannot be resolved (because of rule 2) and thus the anchor will not be selectable. If an anchor falls partially in the region and partially out of the region, only that area within the region is selectable. (See Figure 8-8.)

The rules for mapping coordinate values to visual display space represent a compromise between reusing existing HTML functionality and working with SMIL layout rules. For profiles that do not support SMIL layout, this is not a problem. Unfortunately, these profiles often have only limited SMIL linking

support. More information on spatial cropping is provided in Chapter 11: *Subsetting and Extending Media*.

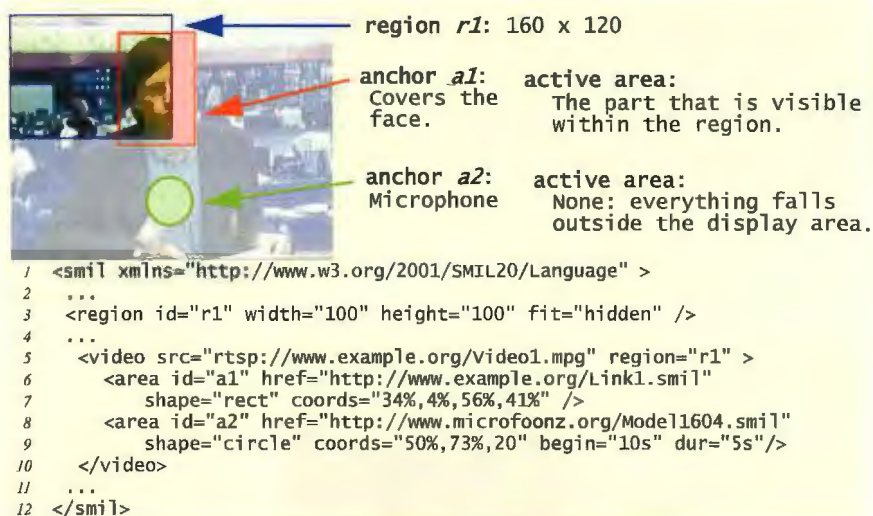


Figure 8-8. The shape and coords attributes within the SMIL `<area>` element.

8.2.5 Attributes to Control the Playstate After Activation

When a link is activated (followed) in a SMIL presentation, the SMIL timing and synchronization rules specify the temporal state of the link's destination. But what happens then? Does the new document start playing, or does it wait for you to request that it starts? And what happens to the state of the original presentation? Does it get replaced, does it get paused or does it simply keep on playing? SMIL Basic Linking provides a collection of attributes that control both the original (source) document and the new (destination) document.

SMIL 2.0 uses three attributes to control the state of the source and destination presentations when a link is followed: *destinationPlaystate*, *show* and *sourcePlaystate*. Note that in order to maintain compatibility with SMIL 1.0, both *show* and *sourcePlaystate* can influence the behavior of the source presentation after a link is activated. When both are specified, the value of the *show* attribute has precedence over *sourcePlaystate*.

The specification of behavior for the source and destination presentations is generally only relevant for linking to external documents. Special rules for source and destination states exist if the source and destination anchors are in the same SMIL file; in this case, the destination always replaces the source.

Attribute: *destinationPlaystate*

This attribute specifies the state of the presentation containing the destination anchor. It can be set to *play* or *pause*. (The default is *play*.) If paused, user inter-

action in the player controlling the destination anchor will need to be used to play the presentation.

Attribute: show

This attribute specifies how the source and destination presentations are handled. The show attribute was the only control attribute in SMIL 1.0; for SMIL 2.0 implementations, two values are supported: *new* and *replace*. If the value of show is *new*, then both the source and destination presentations will be active after following the link. (They will most likely be activated in separate player instances, if the implementation supports this.) If the value of show is *replace*, then the source presentation is paused and — if supported by the player — its current state is saved for possible reactivation later. The destination presentation then replaces the source in the original player instance.

Attribute: sourcePlaystate

This attribute controls the behavior of the source presentation after a link is activated. It allows the following values: *play*, *pause* and *stop*. If *play* is specified, the source presentation continues playing. If *pause* is specified, the source presentation pauses at the point the link is activated; when the destination presentation completes, the source presentation should resume. If *stop* is specified, the source presentation is terminated.

8.2.6 Other Linking Control Attributes

SMIL provides a set of attributes that control when a link is activated, where the presentation containing the destination anchor is rendered, and the volume levels of the source and destination when a link is activated. SMIL also supports HTML's *alt*, *tabindex* and *accesskey* linking attributes, although it redefines them for use in the SMIL namespace. All of these attributes apply to both SMIL's *<a>* and *<area>* elements.

Some of the attributes below are defined in other XML languages as well as in SMIL. SMIL provides a uniform container model for (media) content in which the SMIL document may embed content in other formats, or other formats may embed SMIL content. For example, you could construct a SMIL presentation that contains an HTML object, or you could construct an HTML presentation that contains a separate SMIL document that gets rendered via a plug-in player. It could be that the outer, containing document defines an anchor and link address on top of an anchor/link defined in the embedded document. In such cases, the link defined by the containing (outer) document has precedence over the link in the embedded (inner) document.

If an embedded document contains a — non-conflicting — anchor/link, then activating this link impacts that embedded document only: SMIL 2.0 does not have an architecture that allows the embedded link to influence the containing document.



Attribute: **actuate**

SMIL allows links to be activated by user interaction (as in HTML), or it allows links to be activated automatically. Link activation is controlled by the **actuate** attribute. The *onRequest* value for **actuate** specifies that the link is followed using the human interaction interface of the player. The *onLoad* value specifies that the link is activated at the moment that the active duration of the object containing the link begins. (The name *onLoad* is somewhat of a misnomer, since objects can be loaded by the player well in advance of the start of their active duration. While *onStart* might have been a wiser choice, the name was used for historical HTML reasons.)

Attribute: **target**

The **target** attribute controls where the destination is rendered upon activation. SMIL's use of the HTML **target** attribute allows a destination presentation to be placed in a named SMIL region or an HTML frame; the identifiers used across regions and frames must be unique.

Attribute: **external**

The **external** attribute controls the type of application that is used to render the destination after it is activated. The SMIL **external** attribute specifies how the destination is processed: if it has a value of *true*, an external renderer of a type appropriate for the type of the media object will be used, and if the value is *false*, the destination will be opened by the same application as was used for the source presentation — this will usually be the SMIL player. The default is *false*. If a value of *true* is used, then the implementation environment (not the SMIL standard) determines which application should render the media object.

Attribute: **sourceLevel** and **destinationLevel**

Two attributes control the audio levels of the source and destination presentation after a link has been activated. The **sourceLevel** attribute takes a percentage value and applies it to the cumulative volume of the source presentation. The **destinationLevel** attribute takes a percentage value and applies it to the volume of the destination presentation.

Attribute: **alt**

This attribute specifies a text string which may be used by a user agent to identify an anchor. While it can be used in the context of linking, it is not defined as part of the *LinkingAttributes* module, but is part of the *MediaObjects* module (see Chapter 17: *Meta-Information, Media Description and XML Accessibility*).

Attribute: **tabindex**

The value of this attribute is an integer. It specifies a selection ordering of a set of anchors that are active (in the sense of SMIL timing). User agents may sup-



port tab-based selection of anchors — that is, they may support the use of the tab key to cycle through a set of active anchors — any one of which can then be selected by an appropriate UI device, such as a mouse or an enter key. The use of this attribute is identical to its HTML counterpart, except that only those SMIL elements that are in their active period (that is, being rendered or in their freeze period) are included in the tabbing order.

Attribute: **accesskey**

The value of this attribute defines a keyboard character or device rocker button that can be used to activate a link. If there is a conflict between the definition of an access key between the SMIL presentation and a media object in that presentation — for example, if an HTML object is being presented in a SMIL presentation, and both lay claim to the 'K' key — then the SMIL presentation will process the key and actuate the link. If the embedded content specifies an access key that is not in conflict with a SMIL definition, then the associated link is followed within the embedded content. If two anchors within a SMIL document share the same access key, the one rendered in a SMIL region having the highest z-index stacking order dominates.

8.3 Other Uses of Links in SMIL

The SMIL linking examples presented up to this point have concentrated on using linking for navigation. As we discussed earlier in this chapter, linking is also often used for activation control and message passing. We will briefly consider these uses in the following paragraphs; more advanced activation control is considered in the sections on SMIL timing and the exclusive element `<exc1>`.



8.3.1 SMIL Linking and Activation Control

Where the use of linking for navigation allows the focus of a presentation to be set to some non-linear piece of content, the use of linking for activation control typically allows a follow-on part of a presentation to be delayed until some explicit user action takes place. SMIL 2.0 introduced interactive timing in the `begin` and `end` attributes to manage this efficiently. SMIL 2.0 also introduced the `<exc1>` element for managing presentation-wide activation control. Since, however, not all SMIL 2.0 profiles support interactive timing or the exclusive construct, some designers may still need to use linking to manage a minimal form of presentation interactivity.

Figure 8-4 and Figure 8-5 considered two methods for managing a mix of navigation and activation control: both methods allowed a user to click on a 'next' icon to advance to the following story in a news presentation. In those examples, if the link was not followed, the next story would be reached when the current one ended. Two other examples of activation control are shown in Figure 8-9. Here we see a user's manual for a flashlight. The user is given some complex instructions (in this case, how to open the top), at which point the pre-

sensation pauses on line 8 until the user clicks on the anchor over the *Continue* button.

Line 8 in Figure 8-9 uses *lazy activation control*: if the user does not click within 90 seconds, the presentation continues on its own. This may be appropriate for some situations, but in others it may be more appropriate to give an open-ended delay. (The *findBatteries* segment starting on line 11 is one example: for some users, finding the correct batteries may require a trip to the store!) Line 15 gives a potential solution: the duration of the image containing the anchor is set to *indefinite* rather than to a fixed time period.

The specification of indefinite durations can be useful, but also dangerous. Many SMIL authors assume that indefinite means forever, but this is not correct. A duration of indefinite really means: forever, or the end of the parent time container, whichever comes first. Consider the following fragment, which is a variant on lines 11 through 17 of Figure 8-9:

```
***
<par id="findBatteries" endsync="v">
  <video id="v" src="findCorrectBatt.mpg" ... />
  
  <a href="insertBatteries">
    
  </a>
</par>
***
```

In this example, the duration of the *findBatteries* <par> is determined by the duration of the video v. This is because the image I1 has no explicit duration, and the image I2 will stay active only until the resolved end time of its parent (the <par>).

While the use of linking for interaction is most prevalent in SMIL 1.0 documents or for applications targeted to light-weight SMIL 2.0 profiles, this facility



Figure 8-9. Using linking for activation control.

should be used sparingly. One of the most expensive operations that a SMIL player must perform is the dynamic calculation of the effective playback time of a link target, and the subsequent scheduling of all of the media that must be active at that point. Low-performance SMIL platforms may not be able to process these calculations in a viewer-friendly manner. If at all possible, SMIL interactive timing should be used instead.

8.3.2 SMIL Linking and Message Passing

Early in the development of HTML, authors discovered that linking was a handy way of sending arguments, parameters and state information to and from servers. The typical mechanism for passing these messages in HTML looked like:

```
http://example.org/people/public/show.php4?persnr=554
```

The '?' character designated application-specific information that was piggy-backed onto the link address. Overloading a link address with additional information is typically considered "not done" within the hypertext research community, but it was considered as a "we have no other choice" mechanism by the development community.

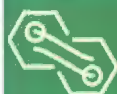
The stricter separation between content and structure in SMIL has had the positive side-effect that **overload** of link addresses is rarely used. This is not because SMIL removes all **need** for message passing, but more because a SMIL document typically sends extra parameters to media objects via the `<param>` element that is specified as content to the media object reference rather than their URI's.

8.4 Summary and Conclusion

This chapter has covered the basics of SMIL's temporal linking facilities. SMIL's linking support provides both short-cut anchors for entire nodes via the `<a>` element, and extensive support for partial-node anchors via the `<area>` element.

Not all SMIL 2.0 profiles implement SMIL linking functionality, and (unfortunately) not all players that include linking functionality do so correctly. This is due to the nature of SMIL 2.0 linking: the ability to faithfully produce a destination link context for timed media is, simply put, hard. It can consume considerable resources and it can make demands on media codecs that are not trivial to implement.

Designers interested in using SMIL linking to control activation of media objects should strongly consider SMIL's event-based interaction facilities. These facilities, which center around the use of the `<exc1>` element, are discussed in Chapter 13: *Advanced SMIL Timing Behavior and Control*.



8.5 Further Resources

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>.

XLink

XML Linking Language (XLink) Version 1.0, Steven DeRose, Eve Maler and Ron Daniel Jr. (eds), W3C Recommendation, 27 June 2001.
<http://www.w3.org/TR/xlink>.



Advanced SMIL Features

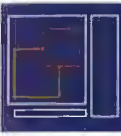
Part Three provides descriptions of the more advanced features of SMIL 2.0. This includes all of the advanced constructs from SMIL's media, layout, timing, animation, content control and meta-information modules.

Since this part takes a detailed look at individual constructs, it is less tutorial in nature than parts one and two. Still, every chapter shows how SMIL works by using examples from complete SMIL presentations. As with the earlier chapters, all of the presentations referenced in this section are available on the book's Web site.

Topics Covered



Chapter 11 considers the sub-setting of media assets in a presentation. Where basic SMIL functionality includes facilities for referencing complete objects, this chapter looks at the various elements and attributes that support spatial, temporal and logical sub-division of media content and parameters.



Chapter 12 describes the advanced facilities supported by the full SMIL 2.0 Language profile for hierarchical and multi-window layout. We consider facilities for logical media alignment in regions, sub-region position of media, hierarchical layout and the use of multiple top-level presentation windows.



Chapter 13 provides a description of advanced SMIL timing behavior and control. This chapter focuses on SMIL support for event based timing and the selective composition of media elements. It also provides a detailed description of the SMIL timing model.



Chapter 14 describes the advanced use of SMIL timing attributes. This chapter looks in detail at attributes that control fill behavior, repeat behavior and synchronization behavior. It also describes the facilities available for manipulating the order and pace of time within a SMIL presentation.



Chapter 15 provides an overview of SMIL Animation. The chapter provides an introduction to the SMIL animation model and then describes the elements and attributes available for both SMIL Basic Animation and SMIL Spline Animation.



Chapter 16 considers advanced support for SMIL Content Control. We describe the facilities provided to pre-fetch content before it is needed in an application, the facilities for defining and using custom test attributes and the facilities available for creating future-proof SMIL presentations.



Chapter 17 closes this part with a description of SMIL meta-data and media description facilities. While many of these constructs define basic functionality, they are included in this part because not all SMIL profiles provide complete meta-information support.

Goals

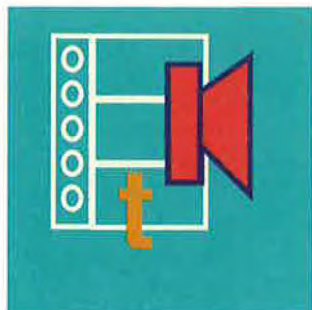
By the time you complete Part Three, you should have a thorough understanding of all of SMIL's elements and attributes. You will be able to not only read and write SMIL code for all of the SMIL profiles, but you have an in-depth understanding of why SMIL does things the way it does.

Prerequisites

In order to get the most out of this part, you should have read and understood the material in Part One and Part Two of this book. You should be familiar with the SMIL timing model and the general element and attribute architecture of the constructs in the various SMIL profiles. Perhaps the most important prerequisite is a willingness to dive into the details of the more esoteric aspects of the SMIL 2.0 specification.

11

Subsetting and Extending Media



SMIL provides a host of options for limiting the scope of media objects used in presentations. Some of these options are purely visual — such as defining layout behavior for cropping, scaling or stretching media objects to fit a display area. Other facilities exist for restricting the active area of media objects or for limiting the amount of continuous media that gets transferred from the server to the player/browser. Still other facilities exist for extending the way that media objects get processed by providing a general purpose parameter-passing mechanism.

This chapter provides descriptions of most of the SMIL media subsetting and extension mechanisms. Facilities that deal purely with the rendering transformations on media objects were covered in Chapter 6: *SMIL Basic Layout* and Chapter 12: *Advanced Layout Topics*; these will not be repeated here. Facilities for restricting the active area of media objects were introduced in Chapter 8: *Basic Linking*, but they are defined more completely in the sections below.

We begin with a discussion of spatial cropping of a media object's active selection area. We then consider temporal clipping (both physical and logical) of continuous media objects. Next, we look at the subsetting of text media via fragments. We conclude with SMIL's support for passing initialization parameters to media objects and media renderers.

11.1 Cropping Media Spatially

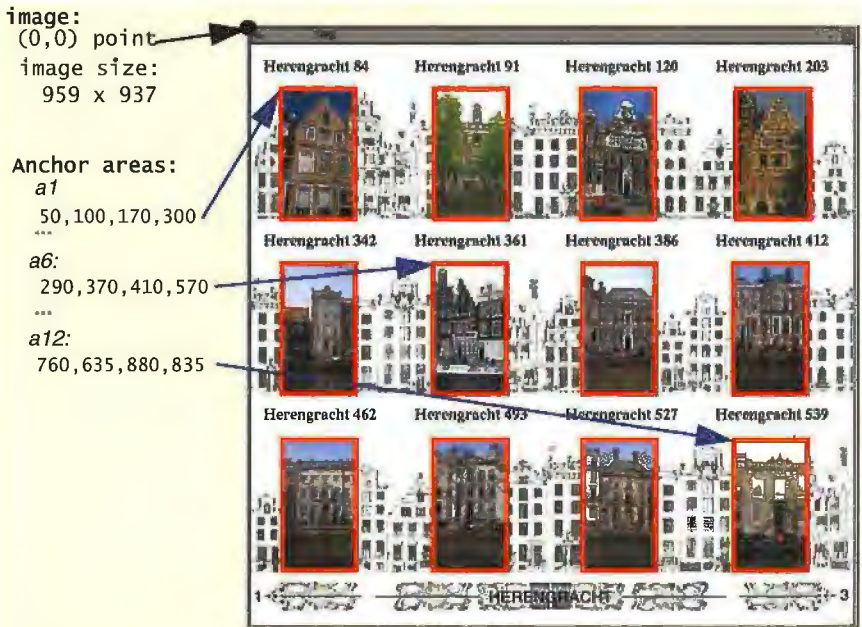
The main mechanisms for defining spatial cropping of media objects are defined in the SMIL layout modules and the SMIL linking modules. Spatial cropping for layout in the SMIL 2.0 Language profile is done using the `fit` attribute of the `<region>` element. Spatial cropping that restricts the active area of links is done using the `coords` and `shape` attributes of the `<area>` element.

The use of the `fit` attribute is discussed completely in our section on *Stacking and Scaling Attributes* on page 114. The use of the `coords` and `shape` attributes of the `<area>` element was introduced on page 168. That introduction will be expanded upon in the following section.

11.1.1 Understanding Spatial Cropping for Linking

SMIL's linking support defines three elements that are used to identify the active area of a link anchor: `<a>`, `<area>` and `<anchor>`. Of these, the `<a>` element always uses the entire geometry of the media object with which it is associated for the link anchor. (The `<a>` element may have timing attributes that limit the active *period* of the anchor, but not the active *area*.) The `<area>` and `<anchor>` elements allow the active area to be restricted to a portion of the media's geometry. They also allow limiting the active period to a portion of the anchor's duration. The `<anchor>` is a SMIL 1.0 element that is, for all intents and purposes, functionally equivalent to `<area>`. We will not consider `<anchor>` separately in this chapter.

Figure 11-1 illustrates the typical use of the spatial cropping provided by the `coords` and `shape` attributes of the `<area>` element. Here we see an image that has 12 anchors defined as child content. Each `coords` attribute specifies the



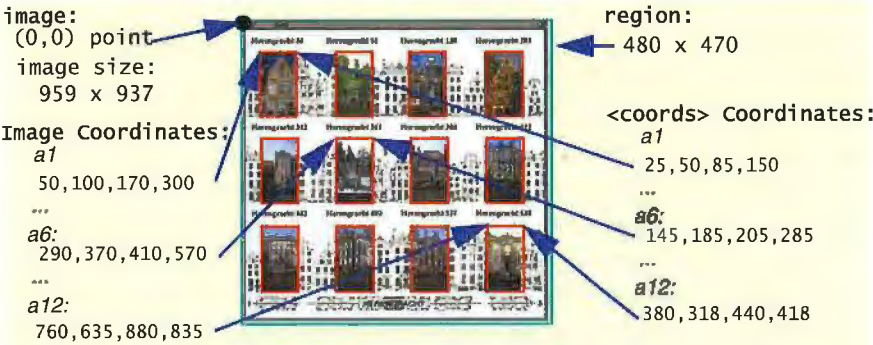
```
1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language" >  
2   ...  
3     
4     <area id="a1" shape="rect" coords="50,100,170,300" ... />  
5     ...  
6     <area id="a6" shape="rect" coords="290,370,410,570" ... />  
7     ...  
8     <area id="a12" shape="rect" coords="760,635,880,835" ... />  
9   </img>  
10  ...  
11 </smil>
```

Figure 11-1. Defining anchors using the `coords` attribute.

extents of an anchor. (Our example uses the default rectangle shape.) In most use cases, each <area> element definition will also include linking and/or temporal scope attributes, but these are not important for our discussion and have been left out of the code fragment. The image used in this example is a screen-dump that is used within a presentation: the window system's borders and control areas are part of the image, not part of the player background.

The use of the coords attribute seems very straightforward: each coordinate pair is mapped to a part of the image. Unfortunately, the values given in the coords attribute have only an indirect association with the coordinates of the image because they represent *region space* coordinates instead of media space coordinates. (This was done to provide compatibility with HTML.)

Why is this a problem? First, consider that the image used in the example is not small: it is 959 x 937 pixels. It may be that a presentation that uses the image will apply the visual scaling properties available in SMIL layout to reduce the space needed to show the houses. If the image was scaled to fit an area that was 50% of the original image size, the coordinates used in the coords attribute would also need to be resized by the presentation author. (See Figure 11-2.) A second problem is that the image may be clipped if it does not fit inside the region provided by the layout system. As a result, (part of) an anchor may be placed outside the viewable space of the image. This is a problem for linking, since the unseen part of the anchor is not active. A third problem is that an image may be positioned at a point other than the upper left corner of the



```
1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language" >
2 ...
3 <region id="M" width="480" height="470" fit="meet" ... />
4 ...
5 
6   <area id="a1" shape="rect" coords="25,50,85,150" ... />
7   ...
8   <area id="a6" shape="rect" coords="145,185,205,285" ... />
9   ...
10  <area id="a12" shape="rect" coords="380,318,440,418" ... />
11 </img>
12 ...
13 </smil>
```

Figure 11-2. Display-space and media-space issues with the coords attribute.

region. If this happens as a result of animating the image using sub-region placement (as discussed in Chapter 12), the coordinate values in the `coords` attribute also will need to be adjusted. Often, authoring software will handle these cases, but for documents edited by hand, the mis-match between media space and display space coordinates can lead to frustrations.

11.1.2 Elements for Spatial Cropping

This section defines the `<area>` element.

Element: `<area>`

The structure of the `<area>` element's cropping attributes is laid out in Table 11-1. The use of these attributes are similar to the use in HTML. In most cases, the `<area>` element is applied as a child of a media object to define an anchor's active area.

<area>			Profiles				
attributes	basic linking	coords	L	M	B	I	H
		shape					
parents	media object, <brush>		L	M	B	I	H
children	<animate>, <set>		L				H

Table 11-1. Structure of the `<area>` element's cropping attributes

11.1.3 Attributes for Spatial Cropping

This section defines the `coords` and `shape` attributes.

Attribute: `shape`

The `shape` attribute allows one of three shapes to be defined for use with the active area, plus one value that selects the entire area.

- *rect*: this defines the anchor as a rectangle. This is the default.
- *circle*: this defines the active area to be a circle.
- *poly*: this defines the active area as a (closed) polygon.
- *default*: this value specified that the entire rendering region should be used as the active area.

The use of the value *default* for `shape` can be useful for two reasons. First, the value can be animated in the presentation to assume another value; in this case, the default value can be seen as a placeholder. The second use of the default value is to specify that the entire area has no linking behavior. (This is done in conjunction with the `href="nohref"` attribute assignment.) While interesting in a linking content, this has no further utility for spatial cropping.

Note that the default value of the `shape` attribute is *rect*, not *default*.

Attribute: `coords`

The `coords` attribute contains a set of values that define the geometry of the active area in display coordinates. The `coords` attribute takes a list of coordinate values and/or shape modifiers as arguments.

The number of arguments in the list and their interpretation depends on the value of the *shape* attribute. When a geometry values is expected, it may be either a pixel value or a percentage value (including the % character).

- *rect*: the values list contains four values representing the left and top offsets of the upper corner of the anchor and the left and top corner of the bottom of the anchor.
- *circle*: three values are expected. The first two values are the left and top offsets of the center point of the circle. The third value is the radius.
- *poly*: a closed set of coordinate pairs are expected, each representing a left and top coordinate for a point in the polygon. If the polygon is not closed, the player is expected to provide a closing value pair.
- *default*: the list of values is ignored. The *default* value allows a *coords* attribute list to be specified when it is expected that this element will be changed using SMIL Animation: the (ignored) coordinate list is specified so that when the value of the *shape* attribute is changed via animation, a valid *coords* argument will be available to be animated.

Since the coordinates used by the *coords* attribute are resolved to region space, care should be taken when (automatic) scaling of media is used or when sub-region positioning is applied to the media object. See the discussion on page 240 for details.

11.2 Clipping Media in Time

SMIL 2.0 offers many options for specifying when a media element starts in a presentation and it offers many options for specifying how much of the media object will be played before it terminates. Most of the mechanisms available in SMIL allow subsetting of media to occur once the media has been fetched from the server. The *MediaClipping* module provides extra control that prevents unused portions of a media object to be transferred to the SMIL player.

11.2.1 Understanding Temporal Clipping of Media

Continuous media objects typically consist of a series of samples that, when combined, allow a video or audio object to be digitally defined. Audio objects can consist of anywhere between 4K and 88K samples per second, with each sample requiring 8, 16 or 32 bits. Video objects may contain anywhere from 5 to 30 samples per second, with each sample requiring between 0.5Kbits and nearly 6 Mbits. A *media clip* is a sequence of samples that occur over a defined period of time. That time may be measured in clock-based timestamp (that is, seconds 12-48 of an audio object) or in terms of a frame-based timestamp (frames 3,000-46,000).

The structure of a video-based media clip is illustrated in Figure 11-3. Here we see a clip that is defined with $t_{clipBegin}$ as a begin time and $t_{clipEnd}$ as an end time.





Figure 11-3. Media clip structure based on timestamps.

There are two problems posed in defining a media clip. First, the notion of the begin and end times need to be specified in terms that are useful for the media encoding and the SMIL player. Since the SMIL player is media format agnostic, this time specification is non-trivial. A second, more challenging problem is that the source media object may be highly compressed or encoded. This means that, at the server, it may be difficult to translate a time value to a particular encoded sample/frame in the audio or video.

SMIL does not attempt to solve these problems directly. Instead, it defines two attributes for specifying the begin and end times and gives each the option of using SMIL's notion of playback time as a selection criteria for the clip or the ability of using a media standard timing structure. It is the media server's responsibility (via the object's API) to translate the time value to the appropriate sample or frame.

The efficiency with which media clips can be obtained will vary from format to format and from media server to media server. Some formats may not allow efficient media clipping because of compression or because of the lack of a seek command in the object's interface. In the worst case, the first 90% of a large media item may be required to be able to render the last 10%. Even if a format is amenable to subsetting, a media server or transport protocol may not be able to deliver a partial media object. In both cases, the SMIL player should simulate media clipping internally.

11.2.2 Attributes for Temporal Clipping

The SMIL *MediaClipping* module defines two attributes for defining a media clip: `clipBegin` and `clipEnd`. (The SMIL 1.0 versions were `clip-begin` and `clip-end`.) No SMIL elements are required to support temporal clipping.

Attribute: `clipBegin`

The `clipBegin` attribute specifies the start point of a media clip. It is an offset from the beginning of the media object. The default value for `clipBegin` is 0 (the start of the media object).

The `clipBegin` attribute takes a single timestamp value as an argument. This argument consists of a *media metric specifier* and a *time value*. The media metric specifier will determine the format of the time value. SMIL 2.0 allows the following media metric specifiers:

- **npt**: the normal playtime metric. The normal play time of an object is a SMIL clock value. The syntax is:

```
clipBegin="npt=10s"
```

The meaning of this specification is that the clip begins 10 seconds into its normal play time. This specification is coarse (that is, it does not correspond to a particular frame), but it is useful for many subsetting applications. Use of the 'npt=' prefix is optional in all profiles except SMIL 1.0.

- **smpte** (also specified as **smpte-30-drop**): a video-based timecode specifier that uses a frame selector based on the NTSC standard. The *smpte* time value consists of an `hours:minutes:seconds:frames.subframes` specification. (Sub-frames are expressed in hundredths of a frame; a sub-frame value of 0 may be left unspecified.) An example of the syntax is:

```
clipBegin="smpte=00:00:10:24.97"
```

for a video clip that starts just before the 25th frame in the 10th second of the video. The *smpte* specifier is used with an encoding that contains approximately 30 frames per second, except that the first two frames of every minute are dropped (but not in each 10th minute). One can consider this a reasonably fine-grained time specification.

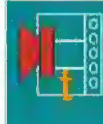
- **smpte-25**: This is similar to the *smpte* value, except that a frame selector is used that is based on the PAL standard. Each second consists of 25 frames, with each frame being further divided into 100 sub-frames.

The `clipBegin` attribute does not have support for special audio time encoding. The `clipBegin` attribute also does not have support for specifying an index into a compressed media object. The `clipBegin` *does* allow times to be expressed indirectly using the media marker format defined on page 248.

If specified together with other SMIL attributes, the `clipBegin` time is applied first.

Attribute: clipEnd

The `clipEnd` attribute is similar to the `clipBegin` attribute, except that it defines an end time for the clip. It accepts timestamps in the same manner as `clipBegin`. If the time defined by `clipEnd` is greater than the duration of the media object, the `clipEnd` attribute is ignored. The default value is the intrinsic duration of the media object.



11.2.3 Examples of Temporal Clipping

The following fragment illustrates the specification of a media clip:

```
<video id="c1" src="rtsp://www.example.org/wb.mpg"
      clipBegin="9.25s" clipEnd="npt=6:42:25" />
```

The start of the clip is set to 9.25 seconds of normal play time. (The 'npt=' prefix is the default.) The end time is at 6 hours, 42 minutes and 25 seconds.

If either clipBegin or clipEnd is missing, the default values are assumed:

```
<audio id="c2" src="http://www.example.org/wb.mp3" clipBegin="npt=4"/>
```

```
<audio id="c3" src="http://www.example.org/triangle.wav" clipEnd="npt=58s"/>
```

The first audio object will start at 4 seconds into the object and will play to completion. The second audio object starts at the beginning and plays until the end of the intrinsic duration or 58 seconds, whichever comes first.

If special timing attributes accompany clipBegin or clipEnd, the clip is extracted first:

```
<video id="c4" src="http://www.example.org/wb.mpg" begin="3s"
      repeatCount="2" clipBegin="npt=4" clipEnd="npt=58s"/>
```

When processing this video, first the clip is extracted. Next, the clip is rendered starting at 3 seconds after the nominal begin time of the element. The video clip between the clipBegin and clipEnd is repeated twice.

The clipBegin and clipEnd attributes cannot be used on the <area> element. In order to specify a temporal subset for a linking operation, begin, dur and/or end timing attributes must be used on the <area> element. Examples are given in *Specifying the Activation Period of Source Anchors* on page 166.

11.3 Clipping Media Using Media Markers

The basic clipBegin and clipEnd attributes allow a media clip to be defined in terms of its begin and end times. This assumes, of course, that the author knows the begin and end times of the clip in the media object. If you don't know how long the video is (or if it changes daily), this could be a problem. Luckily, SMIL also supports mechanisms for the logical indexing into media using the media marker attribute values. Media markers are timestamp formats that can be used instead of normal play time (npt) and SMPTE timecode media metric specifiers in clipBegin or clipEnd.

In this section, we'll introduce the concepts of media marker use. Media markers are a new facility in SMIL and not all players provide uniform support for indirect media marker clip files. The Ambulant player supports logical markers using an open media marker format that is available on the book's Web site.

11.3.1 Understanding Media Clip Markers

Figure 11-3 on page 244 illustrated the use of time-based clip markers. Contrast that approach with the situation in Figure 11-4. Here we see two video clips, each representing the top food story in an evening newscast. The SMIL version of this newscast extracts a set of clips from a single 30-minute video object. The problem to be solved is that on Tuesday, the top food story starts at about 12 minutes into the source file, while on Friday the top food story starts nearly 17 minutes into the source. In order to provide the correct index using standard `clipBegin/clipEnd` parameters, two separate SMIL files would need to be constructed: one for Tuesday's newscast and one for Friday.

An alternative is to use logical markers that point indirectly at the appropriate story and the correct time. Figure 11-5 illustrates this approach. A general clip file (*newsClips*) points to the start and end of the top food story for that day. This file can be created fresh daily to contain the correct food index values.



Tuesday's top food story:
Regional Sausage Festival



Friday's top food story:
Fishing Fleet's Fresh Catch



Figure 11-4. Media clip structure based on timestamps.

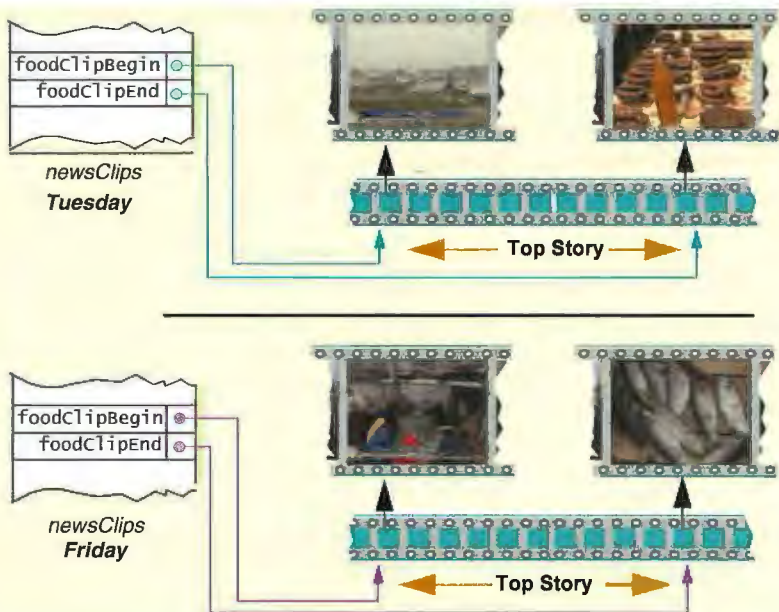


Figure 11-5. Media clip structure based on timestamps.

11.3.2 Attribute Extensions for Media Markers

The SMIL *MediaClipMarkers* module defines a single attribute value extension for use with `clipBegin` and `clipEnd`. (See *Clipping Media in Time* on page 243.) The extension is an attribute value substring that extends the set of values used to specify clips. No new elements and attributes are required for media markers.

The standard media metric specifiers allowed in `clipBegin/clipEnd` are:

- *npt*: normal play time
- *smpte-30-drop*: NTSC SMPTE timestamp
- *smpte-25*: PAL SMPTE timestamp.

These are extended with the following media metric specifier:

- *marker*: a URI name string that specifies a marker file and a location in that file. The file pointed at by the URI is relative to the `src` attribute of the associated media item, not to the XML base of the SMIL document or the document root. An example of the syntax is:

```
clipBegin="marker=newsClips#foodClipBegin"
```

The player/browser is expected to open the marker file and to extract the timestamp associated with the named fragment.

SMIL does not provide a syntax for the marker file. This means that each player/browser will need to publish the syntax for marker files it supports.

11.3.3 Examples Using Media Markers

The following fragment illustrates the specification of a media clip using logical media markers:

```
<video id="c1" src="rtsp://www.example.org/todaysNews.mpg"
clipBegin="marker=newsClips#foodClipBegin"
clipEnd="marker=newsClips#foodClipEnd"/>
```

The start and end of the clips are set to the time found at the referenced begin and end markers. The markers are stored in a marker file that is located relative to the named media object. If the media object *todayNews* changes and the associated *newsClips* media file is updated, the SMIL fragment will always point to the correct clip.

Logical media marker metric specifiers may be mixed with other forms. The following fragment illustrates this:

```
<video id="c1" src="rtsp://www.example.org/todaysNews.mpg"
clipBegin="npt=29.4s" clipEnd="marker=newsClips#foodClipEnd"/>
```

While this syntax is allowed, it will rarely be useful. The importance of mixing clock times with media markers is that if only one marker is specified, default timing semantics make sense. As with other uses of `clipBegin` and `clipEnd`, a non-specified begin time defaults to the beginning of the clip and a non-specified end time defaults to the end of the clip. If the value resolved through the marker file is invalid, the attribute is ignored.

11.4 Subsetting Media Via Fragment Identifiers

The media markers discussed on page 246 provide one example of the use of URI fragment identifiers to locate portions, or fragments, of files. Fragments of media objects will most often be specified for text media types. (They are also used within SVG files and within SMIL itself.) Individual media types can specify if fragment identifiers are allowed for their files and how fragmented displays are evaluated and rendered. The fragment identifier after a URI starts with a '#' character.

11.4.1 Understanding Fragment Identifiers

A fragment identifier points to a particular point in a target document. The target document may contain text or it may contain media. The renderer for the target document is expected to be able to interpret the fragment name and resolve it to a location within the file.

Naked Fragment Identifiers

The vast majority of fragment identifiers in use on the Web are *naked* — that is, they each consist of a single name, rather than having a multi-component structure. Typically, naked fragment identifiers are unique identifier references to particular elements in XML-defined media.



XLink, XPointer and XPath

In addition to naked fragment identifiers, SMIL also supports *XLink*-like elements and attributes for linking. *XPointer* and *XPath* fragment specification may also be supported, depending on the SMIL player.

XLink. XLink provides a generalized XML structure of elements and attributes for referencing external fragments. SMIL does not use Xlink because of its complexity. Instead, SMIL uses constructs similar to XLink for equivalent features and behaviors.

XPointer. XPointer provides attributes that can be used to define location syntax in URI attribute values. One such syntax is XPath.

XPath. Xpath allows fragments to be specified in terms of their structural placement in an XML document rather than by using an explicit XML ID. In this way, document content can be referenced without having to edit the target document source. The SMIL 2.0 specification allows the use of XPath syntax in a URI, but support is not required.

Passing Attributes via Fragments

In early Web documents — and in early versions of SMIL — parameters containing media-specific arguments and values were often passed to a media renderer via private extensions to fragment names using the form:

```
<video id="c1"
  src="rtsp://www.example.org/todaysNews.mpg?action=registered#story1/>
```

With the introduction of the `<param>` element and other forms of media parameter passing mechanisms in SMIL (see page 251), the use of fragments to pass arguments is discouraged.

11.4.2 Language Support for Fragment Identifiers

SMIL does not provide any elements or attributes to support fragment identification. General URI syntax is supported by SMIL; XPointer and XLink extensions may be available in some SMIL players.

11.4.3 Examples Using Fragment Identifiers

The following example illustrates the use of a naked fragment identifier:

```
<text id="z1" src="http://www.example.org/text.html#chapter3" ... />
```

The media renderer is expected to locate the named fragment. Scrolling behavior (that is, showing the fragment identified at the top of the display area) is managed by the media renderer.

Naked fragment identifiers may also be used with media markers:

```
<video id="c1" src="rtsp://www.example.org/todaysNews.mpg"
  clipBegin="marker=newsClips#foodClipBegin" ... />
```

See page 247 for a description of the use of media markers.

SMIL files may also refer to locations within the file using fragment identifiers, as in the destination anchor named *Credits*:

```

  <area id="ExitHH" href="#Credits" coords="6,290,28,323"/>
</img>
```

The following fragment illustrates a non-standard extension use of XPath syntax to identify a fragment in GRiNS (only!). An exclusive element has 5 10-minute children. Each follows the next at 10 minute intervals unless the second child of its parent's first preceding sibling catches a mouse click event:

```
<excl>
<par begin="0">
  <audio src="longAudio1" />
  <img ... />
</par>
<par begin="10:00; xpath(..preceding-sibling::*[1]/*[2].activateEvent)">
  <audio ... />
  <img ... />
</par>
...
<par begin="40:00; xpath(..preceding-sibling::*[1]/*[2].activateEvent)">
  <audio ... />
  <img ... />
</par>
</excl>
```

Readers interested in learning more about XPath, XPointer and XLink are encouraged to read the references suggested at the end of this chapter.

11.5 Initialization Attributes for Media Objects

The SMIL *MediaParam* module defines the `<param>` element for media initialization. The `<param>` is a child of a media object or the `<brush>` element. In addition, the SMIL *MediaParam* module defines three attributes that can be used to initialize media renderers: the `erase`, `mediaRepeat` and `sensitivity` attributes. The `erase` and `sensitivity` attributes may be used with media object references or the `<brush>` element. The `mediaRepeat` attribute is only available for use with media object references.

11.5.1 Understanding the Use of Media Parameters

The SMIL media parameter facilities allow initialization and other media control attributes to be defined and managed in a common manner. The provision of these elements and attributes obsolete the need to define custom extensions to media fragment identifiers.

The `<param>` element provides a media and transfer protocol agnostic means of specifying fetching, transfer and rendering initialization parameters. It allows a parameter to be named, a value to be passed and a value type to be specified. Multiple `<param>` elements may exist as children of a media object reference or of a `<brush>` element.



The `mediaRepeat` attribute allows any repeat behavior that is specified within the media object to be ignored during presentation. The attribute will only have effect with media objects that have a notion of repeat behavior. It assumes that the media renderer will be able to disable repeat behavior.

The `erase` attribute controls the visual persistence of media objects after their active duration and fill period ends. It can be used to supplement `fill` behavior and to preserve media content after the timing context of an object is completed. A typical use is to make sure that a piece of media remains visible until the end of a presentation (unless the display bits are explicitly used for other objects).

The `sensitivity` attribute controls a media object's response to SMIL events. If an object is sensitive to an event such as a mouse click, it will catch the event. If it is declared as being transparent to events, the SMIL engine will pass the event to another media object at a (lower) rendering order.

11.5.2 Elements for Media Initialization

This section defines the `<param>` element.

Element: `<param>`

The `<param>` element provides a general-purpose parameter passing mechanism for use with media objects or the `<brush>` element. The structure of the element is shown in Table 11-2.

The `<param>` element accepts the core SMIL 2.0 element and the `system`, `custom` and `skip-content` control attributes. It makes use of the `name`, `value`, `type` and `valuetype` attributes.

The `<param>` element is empty (it has no children). The parents for `<param>` are media objects and the `<brush>` element.

<param>			Profiles			
attributes	media parameter	name				
		value				
		valuetype				
		type				L
	system test, customTest					
	skip-content					
parents	core					
	media object, <brush>					L

Table 11-2. Structure of the `<param>` element.

11.5.3 Attributes for Media Initialization via `<param>`

The following attributes are used to control and initialize media object content.

Attribute: `name`

The `name` attribute identifies the parameter to be initialized. It is assumed to be supported by the associated media renderer. If the name is unrecognized or unsupported, the `<param>` element instance is ignored.

Attribute: value

The `value` attribute defines an argument to be sent to the media renderer. The processing of the value is not performed in the SMIL engine.

Attribute: valuetype

The `valuetype` argument is used to specify the type of value provided by the `value` attribute. Legal values for `valuetype` are:

- *data*: pass the data specified in the `value` attribute to the object renderer as a string. This is the default.
- *ref*: pass the data specified in the `value` attribute to the object renderer as a string that should be interpreted as a URI. This informs the renderer that the location specified in the URI contains further initialization information.
- *object*: pass the data specified in the `value` attribute to the object renderer as a string that should be resolved as the name of a media object within the same source document. The media object specified by this element is then passed as this parameter's value to the rendering process.
- *type*: inform the user agent that the media referenced identified by the `value` attribute is of the type defined by this attribute. Note that the `type` attribute is only legal if the value of the `valuetype` attribute is *ref*.

The set of parameter names and the acceptable values are determined by the rendering engine, not the SMIL player (or the SMIL standard).



11.5.4 Attributes for Player Media Initialization

This section defines the `mediaRepeat`, `erase` and `sensitivity` attributes.

Attribute: mediaRepeat

The `mediaRepeat` attribute is used as a short-hand control attribute that informs the media renderer whether or not to observe any repeat behavior inside the media object.

The legal values for `mediaRepeat` are:

- *strip*: the repeat behavior inside the media object should be ignored; the media object should not repeat.
- *preserve*: the repeat behavior inside the media object should be respected. This is the default.

The use of `mediaRepeat` assumes that the renderer is able to detect repeat behavior and it is able to strip this behavior on request. If the media renderer cannot process the `mediaRepeat` attribute, the attribute value is ignored. (The SMIL renderer will not simulate `mediaRepeat` behavior.)

Attribute: erase

The `erase` attribute is used to control rendering persistence after the termination of the timing context of the associated media object.

The legal values for the erase attribute are:

- *whenDone*: remove the media object after the application of all SMIL timing and rendering attributes (such as fill). This is the default.
- *never*: do not remove the media object from the screen after its timing content has completed. For discrete media, the image or text will stay on the screen. For continuous media, the final frame will remain visible.

A value of *never* does not mean that the media is always visible. If, for example, another media object is rendered in the same region as the media containing *erase="never"*, the temporal order will determine which object is visible.

An important side-effect of this attribute is that any link source anchors associated with the media object on which the erase attribute is placed remain active until the object is not longer visible.

Attribute: sensitivity

The *sensitivity* attribute allows the event sensitivity of an associated media object to be set. This is used to determine if the media object will respond to UI events such as mouse clicks.

The legal values for this attribute are:

- *opaque*: the object is sensitive to events over the entire area specified with the object. This is the default.
- *transparent*: the object is not sensitive to events. Any event active areas of media objects at a lower rendering priority will be exposed as if the object containing the *sensitivity* attribute was transparent.
- *percentage-value*: the non-negative CSS percentage value provided is compared with the opacity defined for the associated media object's alpha channel value. If the opacity value is defined in the alpha channel is less than the value provided as an argument to the *sensitivity* attribute, the object will be treated as *transparent*. Otherwise, it will be treated as *opaque*.

If a percentage value is provided for a media object that does not have alpha channel support, the default value of *opaque* will be used.

11.5.5 Examples of Media Initialization

The section gives examples of using the `<param>` element and the *erase*, *mediaRepeat* and *sensitivity* attributes.

Using `<param>`

The following example illustrates the use of the `<param>` element:

```
<text id="TX1" region="Text" fill="freeze" src="Assets/text.myFormat" >
  <param name="settings" value="http://www.example.org/text/salesSetup.css"
    valuetype="ref" />
  <param name="font" value="palatino" valuetype="data"/>
  <param name="contents" value="missionStatement" valuetype="object" />
</text>
```

This fragment shows a text object that takes three initialization parameters. First, a pointer to a collection of settings that are saved at the specified URI are passed to the object. The settings file is opened and processed by the media renderer. Next, an explicit font setting is passed as a character string value. Whether this overrides the value provided in the settings file is determined by the media object renderer. Finally, a collection of content defined at the object named *missionStatement* is passed to the renderer. Whether this content will be integrated with or will replace the contents in the *text.myFormat* file is up to the renderer. The SMIL player will render the result of the processed file.

Using mediaRepeat

The following example illustrates the use of the `mediaRepeat` attribute:

```

```

The media object referenced in this element is presumed to be an animated GIF image, which by default will repeat indefinitely. Instead, the `mediaRepeat` attribute will specify that the media renderer will not repeat the animation, but that the SMIL file will be used to explicit bound the duration of the repeat.

Using erase

The following example illustrates the use of the `erase` attribute:

```
<seq id="SX1">
  <par id="PX1">
    
    <audio src="tune.mp3" region="audio" dur="10s" />
  </par>
  <img ... />
  ...
  <img ... />
</seq>
```

In this sequence, the image *static.jpg* is displayed in a background region together with the music object *tune.mp3*. At the end of *tune*'s active duration, the timing context of *PX1* ends and the following image in the sequence is displayed. Ordinarily, the background image *static.jpg* would be erased at the conclusion of *PX1*, but because the `erase` attribute has been set to `never`, it will remain on the screen. Note that the use of the `erase` attribute is a brute force method of providing visual persistence. Often, the use of a clean SMIL structure is preferred.



Using sensitivity

The following code fragment illustrates the use of sensitivity:

```
***
<region id="a" height="100" width="100" z-index="1">
<region id="b" height="100" width="100" z-index="2">
***
<par>
...

  <area id="a1" href="http://www.example.org/buyNow.html"
    coords="10,10,90,90" />
</img>

  <area id="b1" href="http://www.example.org/sellNow.html"
    coords="10,10,90,90" />
...
</par>
...
<animate targetElement="top" attributeName="sensitivity"
  value="opaque" ... />
***
```

Here, two regions are defined that cover the same screen space but at different stacking values. In the body of the presentation, two images are displayed. The image *bottom* is displayed logically under image *top*. In order to allow any events associated with *top* to be ignored, the sensitivity attribute has been set to *transparent*. Any mouse clicks on *top* will be passed on to *bottom*. At some point later in the presentation, the sensitivity value is animated to *opaque*. Now, any link events are processed as if they occurred on *top* instead of *bottom*. (For a detailed discussion of the animation facilities in SMIL, see Chapter 15: *SMIL Animation*.)

11.6 Summary and Conclusion

SMIL provides a host of facilities for controlling the active area or the temporal duration of media items. Spatial cropping is provided via the `coords` attribute of the `<area>` element. Temporal cropping is provided by the `clipBegin` and `clipEnd` attributes. Fragments within objects can be identified by simple or complex fragment identification syntax. Finally, the initialization and control of the rendering of media can be performed using the `<param>` element and the various media control attributes.

Media subsetting is an activity that is supported in several SMIL profiles. The SMIL Language profile supports all of the constructs defined in this chapter. The XHTML+SMIL profile supports spatial cropping, but the Microsoft HTML+Time implementation has no support for timed links; this makes the spatial cropping support of marginal utility from a SMIL perspective.

Most of the media processing elements are supported only in the SMIL 2.0 Language profile. (This is because media processing is often tightly coupled with the nature of the host language document.) Readers should be aware that support for the advanced temporal subsetting settings of `clipBegin` and

clipEnd require support from the media rendering and transport engines. Check the capabilities of your media server carefully to make sure that the requested clipping activities will actually be supported.

11.7 Further Resources

XPointer

XML Pointer Language (XPointer) Version 1.0, Steven DeRose, Eve Maler and Ron Daniel Jr. (eds.), W3C Candidate Recommendation, 11 September 2001.
<http://www.w3.org/TR/xptr>.

XLink

XML Linking Language (XLink) Version 1.0, Steven DeRose, Eve Maler and Ron Daniel Jr. (eds.), W3C Recommendation, 27 June 2001.
<http://www.w3.org/TR/xlink>.

URI

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>.



13

Advanced SMIL Timing Behavior and Control



In the normal course of processing, the temporal 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` and `end` attributes. In further combination with the `<excl>` element, events provide a very powerful mechanism for conditional content activation.

SMIL 2.0 also supports a rich hyperlinking architecture. Unlike links in HTML, the fundamental concept of the SMIL link is that it models a *temporal seek* in a presentation. Rather than simply activating a destination element, the destination play state that is activated is identical to the state that the presentation would have been in if the destination 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 destination 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!

This chapter provides a comprehensive summary of SMIL 2.0 interactive and event behavior. We start with background information on SMIL’s event processing and the interaction facilities available to users. We then contrast event behavior with link-based interaction. We then discuss how scheduled and interactive timing is used in selective content activation using the exclusive (or `<excl>`). We next revisit the SMIL 2.0 timing model and look in detail on how objects are timed and scheduled. The chapter concludes with special elements that allow SMIL’s temporal behavior to be integrated into other XML languages.

Our discussion in this chapter is largely example driven. This should benefit authors of complex SMIL presentations. The material in this chapter will also provide extensive background information for readers who wish to implement or maintain their own SMIL player engines.

13.1 Event Based Timing

In SMIL 1.0, all of the begin and end times in a document could be determined (in theory!) by inspecting the document structure and the various temporal elements and attributes defined for a presentation. The only uncertainty provided by SMIL 1.0 was its support for temporal linking: following a link not only brought the focus of a presentation to another place in the structure of a document, it also required that the temporal consequences of beginning within a nested set of timelines be addressed explicitly.

SMIL 2.0 provides a set of rich enhancements to the SMIL 1.0 model. From a temporal perspective, the major enhancement to SMIL 1.0 was the introduction of event behavior in a document. An *event* is a signal provided to the player that some condition has occurred. The event may be related to an external condition, such as that a mouse click has occurred on some element in the document, or it may be related to an internal condition, such as that a particular other element in the presentation has begun or ended. For implementations that support a document object model (DOM) processor, events may also come from DOM method calls that target some element in the presentation.

13.1.1 Understanding Events, Links and Interaction

This section looks at various aspects of event processing in SMIL 2.0 and contrasts it with the link based processing that was provided in SMIL 1.0 (and continues to exist in SMIL 2.0).



Events and Interaction

In order to understand the structure of event timing, we review some basic timing concepts. Consider the following code fragment:

```
...  
<par id="P1">  
    
  <video id="V1" begin="3s" src="story.mpg" ... />  
</par>  
...
```

A `<par>` time container *P1* contains two child elements: a background image *I1* and a video *V1*. *I1* is defined to start 0 seconds after the beginning of *P1* — this is the default behavior of the image, but we include it here for clarity. The video *V1* is defined to start 3 seconds after *P1* (and thus, 3 seconds after the start of the image). This code fragment illustrates aspects of SMIL scheduled behavior semantics.

We can replace the begin time of the *V1* video to reference a *synchase* value: a *synchase* is the name of an element in the presentation that will act as the timing reference for another element. The SMIL syntax for the updated video object is:

```
<video id="V1" begin="I1.begin+3s" src="story.mpg" ... />
```

The begin of *V1* is now de-coupled from the beginning of its parent *P1* and associated with the start of peer element *I1*. (This relationship is called a *sync arc*; the name comes from the graphical synchronization relationships defined in the CMIF language). From a scheduler's perspective, nothing particularly complex has happened by moving from parent-relative timing to syncbase timing: the begin time of *V1* can still be resolved easily by working through the activation chain. The only extra check that the scheduler must make is that the temporal constraint imposed by the *P1* parent time container is respected: the video may not begin rendering before the parent <par> is active.

We can dramatically change the scheduling within this time container by introducing event-based timing. The following fragment gives an example:

```
***
<par id="P0">
  ***
  <par id="P1">
    
    <video id="V1" begin="I1.begin+3s" src="story.mpg" ... />
  </par>
  ***
  
</par>
***
```

Here, an outer parallel element *P0* is defined. It has *P1* as a child, as well as an image *X* that contains a button. The begin condition of *I1* has been changed to become relative to an *eventbase*: *I1* starts when the viewer clicks on the button image — assuming, of course, that the mouse click occurs within the first 20 seconds of this presentation fragment.



From the perspective of the SMIL scheduler, a fundamental new concept has now been introduced into the language. The actual begin time of *I1* cannot be resolved based on document structure and (syncbase) timing. The exact begin time of *I1* has become unpredictable; as a result, the begin time of *V1* has become equally unpredictable.¹

The SMIL 2.0 scheduler needs to take many aspects of the presentation above into account when defining an activation sequence. We will return to these in *SMIL 2.0 Timing Model, Revisited* on page 310. For now, we'll continue to look at how to specify event-based behavior without worrying about how it gets implemented.

1 There is another unpredictably: will the image *I1* *ever* start? The RealOne player and HTML+Time think so: they start the object *I1* after the image *X* receives a mouse click. The GRiNS and Ambulant players disagree: they do not start *I1* because the parent *P1* will end right after the start of the *P0*. (It ends because children with unresolved begin times do not contribute to the duration of the *P1* container, thus the container ends right after it begins.) We determine who is right in *Mixing Syncbase and Eventbase Activation* on page 317.

Interactive and Non-Interactive Events

In the previous example, we used an interactive event to start part of the presentation. An interactive link gets triggered in response to some form of user interaction. A mouse click is one common form, but not the only type of user interaction. Each SMIL profile that supports events is required to provide a list of interactive events that it supports. (We list the various types in *Event Support in SMIL Profiles* on page 292.)

In addition to interactive events, SMIL also supports non-interactive events. These are events that are generated by the SMIL player itself in response to behavior within the presentation. The following fragment illustrates the use of non-interactive events:

```
...
<par id="P3">
  
  <video id="V3" begin="I3.begin+3s" src="story1.mpg" repeatCount="2" ... />
  <video id="V4" begin="V3.endEvent" src="story2.mpg" ... />
</par>
...
```

We see a `<par>` element *P3* that contains two videos (*V3* and *V4*) and one image (*I3*). The image *I3* starts when *P3* starts. Video *V3* starts 3 seconds after the start of *I3*. Both the start of the image and that of the first video are examples of synchbase timing. The video *V4* starts when the *endEvent* event has been triggered. This occurs at the end of the active duration of *V3*: that is, after it has repeated twice. The start of *V4* is an example of eventbase timing.

Synchbase and Eventbase Timing

Consider the following fragment:

```
...
<par id="P4">
  
  <video id="V5" begin="I4.begin+3s" src="story1.mpg" repeatCount="2" ... />
  <video id="V6" begin="V5.endEvent" src="story2.mpg" repeatCount="2" ... />
  
</par>
...
```

In this example, which is similar but not the same as the one above, the `<par>` *P4* contains two videos (*V5* and *V6*) and two images (*I4* and *I5*). Image *I4* starts when *P4* begins, video *V5* starts 3 seconds after *I4* begins and video *V6* begins at the end of the active duration of *V5*. (This is behavior we have already discussed.) The image *I5* begins when the *end* synchbase condition occurs; this happens at the end of *V6*'s active duration. Note that both the *begin* condition on *V6* and *I5* trigger at the end of the active duration of the associated element. The only difference is that the *begin* of *V6* depends on an eventbase relationship while the *begin* of *I4* depends on an synchbase relationship.

In many cases, the use of a synchbase *begin* or *end* condition will be functionally equivalent to the use of an eventbase *beginEvent* or *endEvent*. There are two aspects of synchbase and eventbase timing that are potentially different:

- A synbase relationship is resolved based on computations made by the scheduler, not events sensed by the scheduler. As soon as image *V6* begins, the scheduler can compute the desired begin time of image *I5* based on information it has about *V6*'s duration and the repeat count. The begin time of *V6* is not pre-computed: the scheduler will wait with determining the actual begin time of *V6* until it gets a notification that *V5* has ended. This can have performance implications within the player: in most cases, synbase timing will be more efficient than eventbase timing, although the differences may in practice be marginal on desktop players.
- SMIL allows synbase values to be resolved to points before the begin time or after the end time of a parent element has terminated. While the actual begin of rendering starts when the parent is active, the scheduled element begin time is the time that it would have started rendering if the parent were active. The Eventbase timestamp will always correspond to the moment that things actually begin or end to render — thus, the start of the parent's active duration. In pathological cases, this can make a difference in the behavior of eventbase and synbase timing.

In general, you should use synbase timing whenever possible. Eventbase timing should be reserved for dealing with interactive events (such as mouse clicks).

Linking and Interaction

The eventbase timing of SMIL 2.0 was not available in SMIL 1.0. In SMIL 1.0, all forms of interaction were expressed using the link constructs discussed in Chapter 8: *Basic Linking*. From a user's perspective, presentations using linking can often appear to be identical to presentations that base interaction on events. From the perspective of the player, there are major differences.

The following example illustrates interaction via links:

```
***
<par id="P5">
  ***
  <par id="P6">
    
    <video id="V7" begin="I1.begin+3s" src="story.mpg" ... />
  </par>
  ...
  
    <area href="#I1" />
  </img>
</par>
***
```

Here, the `<par> P5` contains a child `<par> P6` and an image (*Y*). The image will start and be active for 20 seconds. If during this period the user clicks on the image, the begin time of the *I6* element will be resolved. This also activates *V7*. (You should compare this example with the fragment on page 288.)

The main difference between event-based and link-based interaction is that the element that serves as the link target does not need to be active in order for



the link to be followed, but the element that depends on an eventbase must be active in order for the event to be sensed.

Note that in the fragment above, the link target was placed in the temporal scope of the link source anchor. This is not necessary. When a link is followed, the presentation time is adjusted so that the temporal moment in the presentation is set to be identical to the moment at which the link target would otherwise have occurred. This seek behavior (either forward or backward) is impossible with events.

Linking, Events and SMIL Timing

As we read in *The Temporal Moment of the Destination Anchor* on page 166, the rules for determining the temporal behavior after a link is activated can be summarized as follows:

- *Destination is already active:* If the destination of the link is already active, the element is effectively restarted. The temporal moments of all of its peer and ancestral elements are also effectively reset based on the begin time of the link destination.
- *Destination is inactive, begin time is resolved:* If the begin time of the destination is resolved (that is, if it has a defined begin time because of explicit or implied timing semantics or if the begin time was resolved based on an earlier traversal of the link), then the presentation seeks to that temporal moment. The temporal state of the peers and ancestors is also realigned to match the temporal moment of the link destination.
- *Destination is inactive, begin time is unresolved:* This is the most complex case. If the destination is not active and if it has an unresolved begin time (it is waiting on an event or it has `begin="indefinite"` defined), the scheduler has to recurse up the document tree to find the first resolved parent. This time is applied to each of the children until the link destination is reached. The resolved begin time is applied to the link destination and it is saved by the scheduler for potential reuse in resolving this link (or any link that targets a child of the destination) in the future. The pseudo-resolved begin time for the link may be reset if a parent restarts or if some other condition exists that causes a new effective begin time to be defined.

Example 13-1 provides an example of how link timing is processed in SMIL. It shows an outer `<par> X` containing images *button1* and *button2* (these are used as link sources) and an interior `<par> Y`. *Y* contains four media objects: images *I1* and *I2*, a 15-second audio object *A* and a 30-second video object *V*. The `<par> Y` ends when the video *V* ends. We will look at several separate execution paths through this document to highlight linking behavior.

- 1) The presentation starts. Both images *button1* and *button2* (lines 9 and 12) are started, and both have source anchors activated. The user does not activate either link. At 10 seconds into the presentation, the `<par> Y` on line 2 begins. The video *V* on line 3 starts 5 seconds later, and 7 seconds after *V*



```

1 <par id="X" begin="0">
2   <par id="Y" begin="10s" endsync="V" />
3   <video id="V" begin="5s" src="30svideo.mpg" ... />
4   <img id="I1" begin="V.begin+7s" ... >
5   <img id="I2" begin="I1.click" .../>
6   <audio id="A" begin="I2.begin+5s" src="15sAudio" .../>
7 </par>
8 <a href="#I1">
9   <img id="button1" dur="90s" ... />
10 </a>
11 <a href="#A">
12   <img id="button2" dur="90s" />
13 </a>
14 </par>

```

Example 13-1. Determining the temporal moment of a link destination.

starts, the image *I1* on line 4 begins. (We are now $0+10+5+7=22$ seconds into the presentation.) If the viewer does not click on *I1*, the `<par>` on line 2 ends when *V* ends without ever seeing image *I2* or hearing audio *A*. The presentation ends after 90 seconds.

- 2) The presentation starts and executes as in (1). At 25 seconds into the presentation (3 seconds after image *I1* has started), the user click on *I1*. This starts image *I2* on line 5. After 5 seconds (at 30 seconds into the presentation and 20 seconds into video *V*), the audio *A* begins. Since the `<par>` ends when *V* ends, only 10 seconds of *A* will be heard.
- 3) The presentation starts. After 5 seconds, the user clicks on *button1* on line 9. This causes a link to be followed that starts the image *I1* on line 4. Since *I1* has a resolved begin time (it is 12 seconds after the start of `<par> Y`), the temporal moment of the presentation is set to the effective begin time of *I1*. This means that the video *V* starts being rendered in its 7th second (because of the start offset of *I1*), and the `<par> X` has its temporal moment set to 22 seconds into its effective 90 second duration. If, 5 seconds later, the user re-clicks *button1*, the time of *X* is reset to 22 seconds (and the times of *I1*, *V* and *Y* are also reset to reflect the start of *I1*).
- 4) The presentation starts. After 5 seconds, the user clicks on *button2* on line 12. This causes a link to be followed to audio *A*. *A* has an unresolved begin time, so the scheduler will look for the first ancestor with a resolved begin time: this is the `<par> Y`, with a resolved begin of 10 seconds into the presentation. The effective begin time for *A* is set to the beginning of `<par> Y`, which now start together. After 5 seconds of audio *A*, the video *V* begins, and 7 seconds later image *I1*. The start of audio *A* does not cause an inverse event to start *I2* — *I2* will only start if *I1* is selected when it becomes active. The effective begin time of audio *A* is *sticky*: if the user clicks on *button2* again, the most recently-determined begin time for *A* will be used. Note that if the user clicks on *button1* after selecting *button2*, the pseudo-resolved begin time of *A* will no longer be used, since the presentation will now act as though it the link to audio *A* had never been followed.



The implementation of linking behavior is very interesting — and very complex. This complexity does not only need to be managed when the scheduler is implemented, but each time a link is followed. For this reason, it is usually better to use pure events or events combined with the `<exc1>` element to define interactive presentations.

13.1.2 Event Support in SMIL Profiles

SMIL 2.0 allows events to be used as values in the `begin` and `end` attribute of timed objects. The event is used in conjunction with eventbase syntax:

```
{eventbase-element}.{event-symbol}[clock-value-offset]
```

The eventbase reference has the following components:

- *eventbase-element*: this is an element on which the event identified by the *event-symbol* will take place. The specification of an eventbase element is optional. If an eventbase element is specified, it must be followed by the literal `"."`.
- *event-symbol*: this is the name of the event to be waited on. If an eventbase element was defined, the event will occur on the named element and the waiting element will be notified via event propagation. If no eventbase element was specified, then the behavior depends on the event. Some events are global (such as *keyEvent*), but most use the element itself as the default synchase (such as *activateEvent*).
- *clock-value-offset*: This is a SMIL clock value, preceded by a `“+”` or `“-”` literal. The clock value specified an offset in time that is used in determining an actual `begin` or `end` time relative to the timestamp of the event.

The following list illustrates several forms of eventbase syntax:

```
begin="activateEvent"  
begin="X.activateEvent"  
begin="X.activateEvent+10s"  
begin="X.activateEvent-2s"
```

The first `begin` will resolve once a mouse click occurs in the timing context of the object using the `begin` attribute. The second `begin` resolves when a mouse click is sensed on object `X`. The third `begin` resolves to 10 seconds after the a mouse click was sensed on object `X`. The fourth form of `begin` resolves to a time that is two seconds before a mouse click was sensed on object `X`. Note that actual `begin` times may be constrained by the parent.

Note that this fourth form is perfectly legal, although it may not always be useful. Consider the following fragment:

```
<video begin="startButton.activateEvent-5s" src="tenSecVideo.mpg" ... />
```

The 10 second video will be considered to have begun 5 seconds before a mouse click on the object `startButton`. We do not know in advance when this will be, but we do know that at the moment the start button was clicked, five seconds of the video should have elapsed. As a result, only the last five seconds



of the video will be shown. (This makes some assumptions about the active period of the parent time container, but these are not critical to the example.)

All of our examples use the `begin` attribute; when the `begin` is resolved, the active duration begins with (the first) instance of simple duration. When used with `end`, the resolved time specifies the end of the object's active duration.

Non-Interactive Event Symbols Supported

Various SMIL profiles support a partially-overlapping set of non-interactive event symbols. While the names of the event symbols for a given event are often the same, this is not always so because of historical reasons.

Table 13-1 identifies events and the associated event symbols. The profiles supporting each event symbol are also specified. For columns with a **M** indication, this means that some mainstream mobile implementations — such as the 3GPP PSS5 (and later) profiles — support these events.

Of the set of support events, the *beginEvent* and *endEvent* trigger at the beginning or end of the actual active duration of an element, while the *repeatEvent*/*repeat* event triggers when a element begins the second or higher iteration of its simple duration. The media-related events of XHTML+SMIL may all be triggered by DOM method calls. The *topLayoutOpenEvent* and *topLayoutCloseEvent* events presume support for SMIL's multi-window layout mechanisms.

In case of event conflicts, each profile specifies a priority order for processing events. Interested readers are encouraged to consult the profile definitions.



Event	Event Symbol	Profile
Actual begin of active duration	beginEvent	L M H
Actual end of active duration	endEvent	L M H
Start of repeat of simple duration	repeatEvent	L M H
	repeat	L M H
A top-level window closes	topLayoutCloseEvent	L M H
A top-level window opens	topLayoutOpenEvent	L M H
A media element finishes loading	mediacomplete	L M H
An element loses sync with its timeline	outofsync	L M H
An element regains sync with its timeline	syncrestored	L M H
An element is paused	pause	L M H
A paused element resumes	resume	L M H
An element reverses temporal direction	reverse	L M H
A seek operation is performed	seek	L M H
An element's timeline is reset	reset	L M H
An invalid SMIL time attribute value is set	timeerror	L M H

Table 13-1. Non-interactive event symbols and profile support.

Interactive Events

Various SMIL profiles support a partially-overlapping set of interactive event symbols. While the names of the event symbols for a given event are often the same, this is not always so because of historical reasons.

Table 13-2 identifies events and the associated event symbols. The profiles supporting each event symbol are also specified.

For the XHTML+SMIL profile, most of the mouse interaction events come from the DOM-2 set of dynamic events. More events are supported than are shown, but we have focused on the overlapping set of event types.

The main authoring constraint for various profiles is the use of the name *click* for XHTML+SMIL events and the *activateEvent* name for the SMIL Language profile and derivatives.

Note that for mobile profiles, the presence of mouse events presupposes the availability of a pointing device or a keyboard.

Event	Event Symbol	Profile
A mouse click occurs over an element	activateEvent	L M
	click	H
The mouse is over an element	inBoundsEvent	L M
	mouseover	H
The mouse is moved away from an element	outOfBoundsEvent	L M
	mouseout	
A mouse button is pushed down	mousedown	
A mouse button is released	mouseup	
A mouse moves over the element	mousemove	H
Keyboard focus is obtained	focusInEvent	L
Keyboard focus is lost	focusOutEvent	

Table 13-2. Interactive event symbols and profile support.

13.1.3 Examples of Event Syntax and Semantics

Eventbase syntax can be applied to define the beginning or end of an element's active duration. It cannot be used to define the duration itself or to define other characteristics of the media object.

The following fragment illustrates the use of eventbase syntax as a value for a begin attribute:

```
<video begin="startButton.activateEvent" src="video1.mpg" ... />
```

The video will start when the user clicks on the media associated with the element whose ID is *startButton*. In order for the element to start, its time container must be active.

Events and the Temporal Scope of Elements

The following fragment illustrates the implications of the temporal scope of the parent on the child element waiting to start:

```
***
<par dur="10s">
  <video begin="startButton.activateEvent" src="Video2.mpg" ... />
</par>
***
```

If the user does not click on the *startButton* element within 10 seconds after the start of the `<par>`, the element will not be activated. In addition, *startButton* must also be active in order for it to receive a mouse event. Once the video starts, it will not extend the duration of its parent: in the example above, if a video is started 9 seconds after the parent `<par>` started, only one second of the video will be shown.

Specifying Eventbase Start and End Behavior

Both the start and end of an element may be determined using eventbase timing. Consider the following fragment:

```
***
<par dur="10s">
  <video begin="startButton.activateEvent" end="startButton.activateEvent"
    src="Video2.mpg" ... fill="freeze" />
</par>
***
```

This fragment defines a begin and end condition that use the same event. As specified, the video starts the first time that the start button is clicked. (The event is passed to the *begin* attribute whenever an element is inactive or when it is running and able to accept restarts; in these cases, the *end* attribute is ignored.) The next time that the *startButton* element is clicked, the video will not end — this is not because of the multiple instances of *activateEvent*, but because of the restart characteristics of the video element: since restart behavior is set to the default of always, the second mouse click will restart the video.

In order to avoid unwanted restart behavior, the toggle function should be implemented as follows:

```
***
<par dur="10s">
  <video begin="startButton.activateEvent" end="startButton.activateEvent"
    src="Video2.mpg" restart="whenNotActive" fill="freeze" ... />
</par>
***
```

Now, the first occurrence of the event will be used by the *begin* attribute. The second instance will go to the *end* attribute because the *begin* attribute is not waiting for events when its media is already active. (For more on restart behavior, please see *Restart Behavior* on page 341 of Chapter 14: *Advanced SMIL Timing Attributes*.)



Continuing with the toggle example, assume that we had written the fragment as follows:

```
***
<par dur="10s">
  <video begin="activateEvent" end="activateEvent"
    src="Video2.mpg" restart="whenNotActive" fill="freeze" ... />
</par>
***
```

Here, the eventbase no longer references an external element but the video itself. In the SMIL 2.0 Language profile, it is not possible to click on an element that has not started yet. As a result, the video will never start, since it will never receive a mouse event. This problem can be avoided by defining multiple begin conditions on an element.

Quick Tip

The SMIL timing model explicit states that the same event (such as a mouse click) will never be used to satisfy both a begin and end condition on the same element.

Mixing Eventbase and Scheduled Timing

Consider the following fragment, which further refines our on-going example:

```
***
<par dur="10s">
  <video begin="0; activateEvent" end="activateEvent"
    src="Video2.mpg" restart="whenNotActive" fill="freeze" ... />
</par>
***
```

When the `<par>` begins, the video element starts because it explicitly defines that a start at 0 seconds should occur. Once the video is active, it can receive mouse clicks. The first mouse click will terminate the element. Any following mouse click will restart the element.

Note that in all of these examples, the video remains visible after it is stopped because of the `fill="freeze"` behavior. The freeze behavior may be the default behavior or it may need to be defined explicitly, depending on other attributes on the element. For more information on fill behavior semantics, please see *Fill Behavior Control* on page 328 of Chapter 14: *Advanced SMIL Timing Attributes*.

Using Eventbase Timing to End the Active Duration

Eventbase timing, when applied to the end attribute, ends the active duration of an element, not its simple duration. As one final example, consider the fragment at the end of this paragraph. When the end attribute is triggered by the mouse click, the active duration of the entire element ends. This does not start a new repeat cycle, all repeat cycles end. If it is restarted by a mouse click, the



entire repeat cycle starts as if this were the first time it was played. (The <par> timing will still dominate.)

```
***
<par dur="60s">
  <video repeatCount="3" begin="0s;activateEvent" end="activateEvent"
    src="Video2.mpg" restart="whenNotActive" fill="freeze" ... />
</par>
***
```

13.2 Selective Time Composition

The introduction of eventbase timing not only has implications for the begin and end times of media objects and time containers, it also provides a framework for the introduction of new, interaction-based time containers into the SMIL 2.0 specification. SMIL supports two new elements for what is technically called *a-temporal object composition*: compositing objects based on interaction rather than on scheduled timing. The two elements are the <exc1> or exclusive time container, and the <priorityClass> element for supporting preemptive scheduling of alternative content.

This section describes both new elements and their associated attributes. We start with a general section on integrating event-based timing into SMIL temporal framework and then consider the composition and structure of the elements and attributes. We close this section with a series of examples.



13.2.1 Understanding Selective Composition

The standard SMIL 2.0 time containers are <seq> and <par>. Children of a <seq> have the property that they are evaluated sequentially based on simple begin time constraints. Children of the <par> have the property that they are all evaluated based on a common timeline. For many applications, these two containers provide a comprehensive foundation for structuring media presentations. For some applications, where selectivity in content is important, these two containers require careful authoring and structuring to meet all presentation needs. In SMIL 2.0, interaction-based timing is made significantly easier by the introduction of the <exc1> container.

The <exc1> is an advanced time composition container that has the property that at most one of its children is active at any time. If a child is active and a sibling becomes active, it causes the original element to either *terminate* or *pause*; the terminate/pause choice depends on the setting of various attributes on the <priorityClass>.

The use of the <exc1> element is illustrated in Figure 13-1. Figure 13-1(a) shows the main frame of a presentation. Selecting any of the small images in the sidebar will cause a media object to play in the main window. Two options are shown: if the young girl is selected, a collection of objects about her is presented in the main window, as shown in Figure 13-1(b); if the young boy is



Figure 13-1. Interaction-based structuring of a presentation.

selected, a collection of assets associated with him are presented in the main window, as shown in Figure 13-1(c).

As shown in Example 13-2, the presentation in Figure 13-1 is structured as a background image played in parallel with four thumbnail images and an `<excl>` container. The `<excl>` contains four simple video objects and an interior `<par>` that contains a sequence of images and videos that are shown together with some background audio. The begin times of each of the simple videos on lines 25-27 and the `<par>` are tied to mouse click events on the image thumbnails; when any of them are selected, the associated content plays. If con-

```

1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
2   <head>
3     <layout>
4       ...
5     </layout>
6   </head>
7   <body>
8     <par id="Birthday">
9       <par id="MenuImages" dur="indefinite">
10        
11        
12        
13        
14        <excl id="Videos" dur="indefinite" fillDefault="freeze">
15          <video begin="F1s.activateEvent" region="Video" src="Fz3.avi"/>
16          <video begin="W1s.activateEvent" region="Video" src="Wz3.avi"/>
17          <video begin="W1s.activateEvent" region="Video" src="Az3.avi"/>
18          <par begin="FBTs.activateEvent">
19            <seq>
20              
21              <video id="BTz3" region="Video" src="BTz3.avi" />
22              
23              <video id="BPz3" region="Video" src="BPz3.avi"/>
24            </seq>
25          <audio id="Music" region="audio" src="Ballgame.mp3"/>
26          </par>
27        </excl>
28      </par>
29      
30    </par>
31  </body>
32 </smil>

```

Example 13-2. Using the `<excl>` in a presentation.

tent was already active, it is replaced with new content. This is an example of (video) jukebox interactivity.

We would like the outer structure of this presentation to be static: whenever nothing is played, the thumbnails and the background image is shown. We also want the `<exc1>` to be active indefinitely, since the user should have the opportunity of selecting all of the content available in whatever order the viewer chooses. In order to facilitate this, both the duration of the *MenuImages* `<par>` on line 9 and the *Videos* `<exc1>` have *indefinite* durations defined. The most important initial thing to notice about this presentation is that the children of a `<par>` may be media content or other SMIL timing or content control elements.

In the *Birthday* presentation, the viewer is given the opportunity to select one of the four content groups, but no default content is activated at the start of the presentation. We can change this by re-coding the begin condition on line 15:

```
15 <video begin="0s; F1s.activateEvent" region="video" src="Fz3.avi"/>
```

By mixing scheduled and eventbase timing, the author can define that this video element will start at a particular scheduled time in addition to any event-related begin time.

The scheduled timing does not need to be at the beginning of the exclusive element. In the following fragment, an `<exc1>` contains three children, all of whom start at scheduled periods and some of whom also start based on interactive or non-interactive events:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
...
  <par id="MenuImages" dur="indefinite">
    
    <exc1 id="mixedBag" dur="indefinite" fillDefault="freeze">
      <video id="Va" begin="12s" src="Fz3.avi" ... />
      <video id="Vb" begin="5s; F1s.activateEvent" src="wz3.avi" ... />
      <video id="Vc" begin="20s; Va.endEvent" src="Az3.avi" ... />
    </exc1>
  </par>
...
</smil>
```

When the `<exc1>` starts, nothing happens for the first 5 seconds. At this point, unless there has been a mouse click on the image *F1s*, the video *Vb* begins. Next, at 12s into the active duration of the `<exc1>`, the video *Va* begins. At the conclusion of *Va*, *Vc* starts — unless *Va* lasts longer than 8 seconds; if so, it is cut off by the start of *Vc*. At any point, a user may initiate the start of *Vb*. Once *Vb* ends, none of the other elements will be able to be played, since all have begin times that are not related to user activity. Note that the order of elements within the `<exc1>` is not important (except when a timing conflict occurs, in which the order in the `<exc1>` determines the play order).

In all of the examples of the `<exc1>` shown above, the content is started by scheduled or eventbase timing. The `<exc1>` also allows its contents to be started by synchbase timing or by containing the target link. The main semantic contribution of the `<exc1>` is that only one of its children can be active at any



moment. The way in which content gets activated is less important than this temporal property.

The default behavior of an `<excl>` is that the activation of new content will terminate any existing content. The existing content may also be paused instead of terminated by wrapping the content in a priority class and then defining pause semantics for members of that class:

```
<excl id="mixedBag" dur="indefinite" fillDefault="freeze">
  <priorityClass peers="pause">
    <video id="va" ... />
    <video id="vb" ... />
    <video id="vc" ... />
  </priorityClass>
</excl>
```

Whenever a peer within the priority class starts, it will now pause the content being displayed. When the new content completes, the paused material is resumed. A pause stack is defined, so that multiple levels of interrupts can be defined without any loss of presentation context.

Multiple priority classes may be defined, each grouping a logical collection of one or more children. Each priority class may define not only how its peers react to new content, but also to the way that content in higher or lower classes impact the rendering behavior of the content. (We will give examples when describing the `pauseDisplay` attribute on page 304.)

It is important to realize that the priority class elements are time transparent. This means that they define a logical collection of content, but not a temporal collection of content. In other words, a priority class is never scheduled, but only the children of priority classes are scheduled. The only impact that the priority class has is an ability to define a relative priority of content — such as defining if lower priority content can interrupt higher priority content — and the stop or pause behavior of its peers.

The `<excl>` element may be nested. The `<excl>` is not time transparent; it accepts all standard SMIL timing elements.

One of the most important side-effects of using the `<excl>` container is that it makes timeline-based authoring systems obsolete. It is impossible to draw the temporal relationships among the elements in Example 13-2 using a conventional timeline. Only a combination of structure-based authoring and flexible timing representation can express the functionality of presentations that use eventbase timing and the `<excl>` container. Figure 13-2 shows how the GRiNS Editor uses structure to identify how an exclusive element can be modelled. The purple boxes represent the `<excl>` element; each of the children are shown as timed element, but the timing of the entire `<excl>` is a-temporal.

13.2.2 Elements for Selective Composition

The SMIL 2.0 *ExclTimeContainers* module defines two elements for the selective presentation on one of their children during a presentation: `<excl>` and `<priorityClass>`. The `<excl>` element (pronounced *exclusive*) is a full SMIL

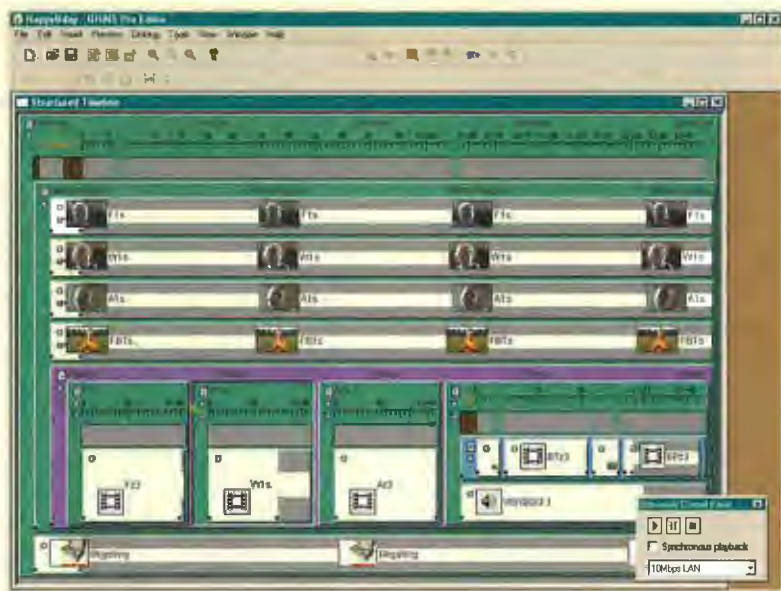


Figure 13-2. Structured timeline used with the `<excl>` element.



2.0 time container when supported by a profile. It supports all of the attributes of other SMIL 2.0 time containers. The `<priorityClass>` element is a content priority structuring element; it may only be used inside an `<excl>` element.

Element: `<excl>`

The structure of the `<excl>` element is given in Table 13-3. The `<excl>` is a full SMIL 2.0 time container that is similar in structure to the `<par>` element with the exception that it has the additional semantic that at most one of its children may be active at any time. (It is not an error condition if none of the children are active.)

The `<excl>` does not require any special-purpose attributes. It accepts all of the general timing, endsync, media description content control and core attributes associated with other SMIL time containers.

The parent of an `<excl>` may be another `<excl>` — that is, `<excl>` containers may be fully nested — or other SMIL 2.0 time containers. The `<excl>` may be wrapped in a link anchor via the `<a>` element and it may be one of the children of a `<switch>` construct. The parent of an embedded `<excl>` may also be an ancestor `<excl>`'s `<priorityClass>` element.

The children of an `<excl>` may be SMIL time containers (except for `<body>`), media objects or animation elements. The children may also be `<switch>` elements or `<a>` link anchors. The `<priorityClass>` element may also be a child of the `<excl>`, but in this case *all* children must be `<priorityClass>` elements.

When an `<exc1>` starts, it provides a default synbase for all of its children. That is, all children have their timing defined by default to be relative to the start of the `<exc1>` element. Unlike a `<par>` or `<seq>` container, the default begin time of the children of an `<exc1>` is not 0 but *indefinite*. This has implications for the duration of an `<exc1>` object: since all of the children have unresolved active durations at the start (because of their indefinite begin times), an `<exc1>` without an explicit duration will end as soon as it begins (unless `endsync="all"`). For this reason, make sure that at least one of the children has a scheduled begin or that the `<exc1>` itself has either an explicit duration or that it has a duration of indefinite. (See the examples on page 305 for more details.)

The primary uses for the `<exc1>` element are to support random-access

jukebox functionality or to serve as an aid in providing audio or text captions.

By default, when any child of an `<exc1>` is activated, it causes any other active child to stop playing. This default behavior can be modified with the `<priorityClass>` element, described next.

Element: `<priorityClass>`

The structure of the `<priorityClass>` element is given in Table 13-4. An author can use the `<priorityClass>` to group children of an `<exc1>` into content collections; these collections may then have a behavior defined that is applied to the children when they are preempted by other content.

The `<priorityClass>` accepts the media description, content control and core SMIL attributes. They also accept four new attributes: `pauseDisplay`, `peers`, `higher`, and `lower`. These new attributes are described below.

The only legal parent element for the `<priorityClass>` element is `<exc1>`.

The legal children for `<priorityClass>` are other SMIL time containers (including an embedded `<exc1>`), plus media, animation, linking and content control elements.

The `<priorityClass>` element does not define an explicit priority for its child elements. The priority is determined by its placement in the source file relative to the parent `<exc1>`. Therefore, priorities cannot be animated or changed at runtime.

<code><exc1></code>		Profiles		
attributes	<code>general timing</code>			
	<code>endsync</code>			
	<code>media description</code>	L		H
	<code>region</code>			
	<code>test</code>			
	<code>core</code>			
parents	<code><body></code>	L		H
parents & children	<code><priorityClass></code>			
	<code><seq></code> , <code><par></code>			
	<code><exc1></code>	L		H
	<code><switch></code>			
	<code><a></code>			
children	<code>media object</code>	L		H
	<code>animation</code>	L		H

Table 13-3. Structure of the `<exc1>` element

The `<priorityClass>` is *time transparent*. It simply defines child behavior if and when they are preempted. It does not have any begin, duration or end timing semantics: only the children of the priority class have temporal attributes.

The `<priorityClass>` element uses three attributes to assign behavior to its children when they are preempted by other content in the `<excl>`: `peers`, which determines the behavior among children in the same priority class; `higher`, which determines the behavior of children in the priority class when they are preempted by content in a higher priority class; and `lower`, which determines the behavior of children in the priority class when they are preempted by content in a lower priority class. In all cases, the behavior for the element active at the time the preempt takes place determines the pause/stop behavior and not the newly scheduled element.

If a child within a priority class is paused, one additional attribute can be used to control when the paused element is visible: `pausedDisplay` can be used to remove an item from view. (In practice, newly started elements will often overlay the paused element, but this is not always the case; this is especially true if SMIL Layout is not used.)

13.2.3 Attributes for Selective Composition

The `<priorityClass>` element defines several new attributes: `peers`, `higher`, `lower` and `pausedDisplay`.

Attribute: `peers`

The `peers` attribute is used within a `<priorityClass>` element to define how an active object will respond when it is preempted by another member of the same priority class. The legal values are:

- **stop:** The active element will stop (terminate) if another child in the same priority class begins. This is the default behavior for the priority class.
- **pause:** The active element will pause if another child in the same priority class begins. The paused element is placed on a pause stack; it will resume when no other child of higher priority is active.

<priorityClass>		Profiles		
attributes	peers			
	lower			
	higher			
	pausedDisplay			H
	media description			
	test			
parents & children	core			
	<excl>			H
children	<seq>, <par>			
	<switch>			
	<a>			
	media object			H
	animation			

Table 13-4. Structure of the `<priorityClass>` element



- *defer*: The active element will continue to play and the to-be-started element will wait until it completes. In effect, the new element is placed on the pause stack at the beginning of its active duration.
- *never*: The active element will continue to play and the to-be-started element is ignored.

When an element is blocked because of the *never* value, it is not considered to have started or ended. No *beginEvent* or *endEvent* will be propagated.

Attribute: higher

The *higher* attribute is used within a `<priorityClass>` element to define how an active object will respond when it is preempted by a member of a higher priority class. The legal values are:

- *stop*: The active element will stop (terminate) if another child in a higher priority class begins.
- *pause*: The active element will pause if another child in a higher priority class begins. The paused element is placed on a pause stack; it will resume when no other child of higher priority is active. This is the default behavior for the children within a lower priority class.

Note that the default value for *higher* is different from the default value for peers. This is done to provide convenient defaults that match typical use cases.

Attribute: lower

The *lower* attribute is used within a `<priorityClass>` element to define how an active object will respond when a child of a lower priority class attempts to start. The legal values are:

- *defer*: The active element will continue to play and the to-be-started element will wait until it completes. In effect, the new element is placed on the pause stack at the beginning of its active duration. This is the default for the lower attribute.
- *never*: The active element will continue to play and the to-be-started element is ignored.

Children that exhibit defer behavior do not propagate a *beginEvent* until they actually start.

Attribute: pauseDisplay

The *pauseDisplay* attribute lets an author define whether a paused child is visible while it is paused and it defines how paused elements that remain visible respond to mouse events. The legal values are:

- *hide*: Do not display the element while it is paused.
- *show*: Display the element while it is paused and allow it to respond to mouse events from within its paused state. This is the default value for the *pauseDisplay* attribute.



- *disable*: Display the element while it is paused but disable any response to mouse events.

The `pausedDisplay` attribute only applies to priority classes that have a peer value of *paused* or a higher value of *paused* and then only to those elements within the priority class that actually exhibit pause behavior. It does not apply to elements that are paused for some other reason within a SMIL document.

13.2.4 Examples of Selective Temporal Composition

This section provides examples of `<exc1>` and `<priorityClass>` behavior. The examples are organized by general use cases of the `<exc1>` element.

Supporting Simple Jukebox Behavior Using Events

The fragment in Example 13-3 illustrates a simple event-based jukebox. Three

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
...
  <par id="MenuImages" dur="indefinite">
    <img id="btn1" ... />
    <img id="btn2" ... />
    <img id="btn3" ... />
    <exc1 id="selectMe" dur="indefinite" fillDefault="freeze">
      <video id="v1" begin="btn1.activateEvent" ... />
      <video id="v2" begin="btn2.activateEvent" ... />
      <video id="v3" begin="btn3.activateEvent" ... />
    </exc1>
  </par>
...
</smil>
```



Example 13-3. An event-based jukebox.

images are used as buttons and three videos are placed in an `<exc1>`. When a viewer selects one of the button images, the associated content element starts. If during play another button is selected, the new content will replace the earlier started video.

Since this example illustrates the basic functionality of the `<exc1>` container, we can also use it to explain some of the consequences of using `<exc1>`'s in a presentation. First, consider what would happen if we replace the `<exc1>` container with a `<par>` element:

```
...
  <par id="selectMe" dur="indefinite" fillDefault="freeze">
    <video id="v1" begin="btn1.activateEvent" ... />
    <video id="v2" begin="btn2.activateEvent" ... />
    <video id="v3" begin="btn3.activateEvent" ... />
  </par>
...

```

(Everything else in the presentation is the same as in Example 13-3.) When the `<par>` is encountered, it is started and, because of the indefinite duration, it remains active even though none of its content is active. When a user clicks on a button, the associated video content will start. This is similar to the `<exc1>`. When the video ends, the `<par>` continues to be active; this is also similar to the

<exc1>. The difference between the <par> and the <exc1> comes when a user selects an image button while another video is active; the new video will start and will play in addition to the first video started. This situation can be avoided by attaching explicit end conditions to the children of the <par>:

```
...
<par id="selectMe" dur="indefinite" fillDefault="freeze">
  <video id="v1" begin="btn1.activateEvent"
    end="v2.beginEvent; v3.beginEvent" ... />
  <video id="v2" begin="btn2.activateEvent"
    end="v1.beginEvent; v3.beginEvent" ... />
  <video id="v3" begin="btn3.activateEvent"
    end="v1.beginEvent; v2.beginEvent" ... />
</par>
...
```

Now, each child begins on a mouse click for the associated button and ends when another element starts. Thus, a <par> can model the default behavior of <exc1>. For non-default behavior (that is, when we want to have the <exc1>'s pause and continue semantics), a <par> cannot be used.

Supporting Simple Jukebox Behavior Using Events

Before moving to other aspects of <exc1> behavior, we first re-code Example 13-3 so that it uses link behavior instead of event behavior to start the child elements. This is illustrated in Example 13-4. While most of the structure of the elements is the same, there are two important differences. First, the <exc1> is no longer a child of the image parent time container. When using link based activation, the <exc1> may be placed anywhere in the presentation. It does not need to be active when the source anchor is selected. Second, the children of the <exc1> have no explicit begin attribute. As a result, they exhibit the default begin="indefinite" behavior.

For equivalent behavior, an <exc1> that uses event-based behavior often provides an easier authoring interface than an <exc1> that is based on link

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
...
  <par id="MenuImages" dur="indefinite">
    <a href="#v1">
      <img id="btn1" ... />
    </a>
    <a href="#v2">
      <img id="btn2" ... />
    </a>
    <a href="#v2">
      <img id="btn3" ... />
    </a>
  </par>
...
  <exc1 id="selectMe" dur="indefinite" fillDefault="freeze">
    <video id="v1" ... />
    <video id="v2" ... />
    <video id="v3" ... />
  </exc1>
</smil>
```

Example 13-4. A link-based jukebox.

behavior. Note that both link-based and event-based activation can be combined in the same <excl>; this is also true for synbase timing.

Determining the Duration of an <excl>

The active duration of an <excl> is determined in the same way as is done for a <par>: if an explicit dur is defined, this is used, otherwise the duration is set to the active duration of its longest child. (In other words, the default value for endsync is last.) As with a <par>, children that have unresolved durations do not contribute to the active duration of the <excl>. Consider this fragment:

```
<excl>
  <video id="v1" begin="btn1.activateEvent" ... />
  ...
  <video id="vn" begin="btnn.activateEvent" ... />
</excl>
```

A set of videos is defined in an <excl>. The start times of the videos are all based on events. As a result, all of the timing is unresolved. This means that the effective duration of the <excl> is 0 seconds, which in turn means that the <excl> will end as soon as it begins. Unless someone can click the mouse very quickly, no video will be shown!

The situation can be avoided by explicitly adding a duration to the <excl>. If an appropriate duration exists, it can be used. If you don't know (or don't care) how long the <excl> should last, you can always use dur="indefinite". A potential problem with using indefinite is that the <excl> will remain active for a period that cannot be determined by the author. It is often useful to identify a child as the explicit end of the <excl> by using the endsync attribute:

```
<excl dur="indefinite" endsync="vn">
  <video id="v1" begin="btn1.activateEvent" ... />
  ...
  <video id="vn" begin="btnn.activateEvent" ... />
</excl>
```

In this example, the duration is set to *indefinite* and an element is identified as an explicit end. Any child may be used as the endsync target.

Using Scheduled Begin Times in an <excl>

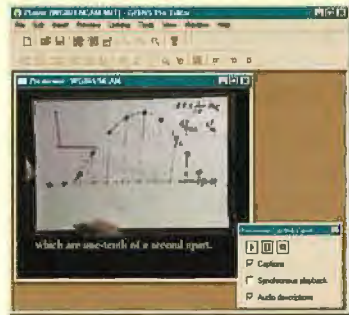
One of the motivating cases for the <excl> was as a container that could integrate audio captions to video material. In this use, a video is played with its audio component, but when complex concepts are discussed — or when visual material in the video is displayed but not described in the video's audio track — an additional bit of audio commentary could be enabled.

The use of the <excl> for audio captions is given in Figure 13-3. This figure combines a number of timing constructs to provide an adaptive presentation. When the presentation begins, a video element is started on line 9 if audio descriptions have been turned off. (Thus, no audio descriptions are wanted.) In parallel, an <excl> is evaluated. The <excl> contains one priority class with four children: one <par> on line 12 and three audio objects on lines 16-18. The





(a)



(b)

```

1 / <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
2   <head>
3     <layout>
4       ...
5     </layout>
6   </head>
7   <body>
8     <par>
9       <video src="lewin.rm" systemAudioDesc="off" .../>
10      <excl dur="indefinite">
11        <priorityClass peers="pause">
12          <par begin="0">
13            <video src="lewin.rm" systemAudioDesc="on" .../>
14            <video src="lewinCaps.rt" systemCaptions="on" .../>
15          </par>
16          <audio begin="34.5s" src="mrkr.wav" systemAudioDesc="on" .../>
17          <audio begin="50s" src="slashes.wav" systemAudioDesc="on" .../>
18          <audio begin="61s" src="deltat.wav" systemAudioDesc="on" .../>
19        </priorityClass>
20      </excl>
21    </par>
22  </body>
23 </smil>

```

Figure 13-3. Defining audio captions using the `<excl>` element.
(Media assets © 1998 M.I.T.)

`<par>` on line 12 has a scheduled begin that starts the child at 0 seconds after the `<excl>` starts. The children of the `<par>` each start at time 0 as well. If the audio descriptions and captions were both requested, these two elements will play. Note that here, two media objects are active in the `<excl>`, but since these are both the children of a single top-level element in the `<excl>`, this is not a violation of the single-active-child rule. If audio descriptions were not wanted but captions were requested, then only the captions element would be active. If neither captions nor audio descriptions were requested, nothing would be played in this `<par>`. Figure 13-3(a) shows both captions and the video being active.

At 34.5 seconds after the `<excl>` starts, the audio component on line 16 will interrupt the `<par>` on line 12. Since the priority class defines peers to pause other peers, the video and captions will freeze while the audio description

takes place. This audio description gives extra information about the diagram, as shown in Figure 13-3(b), and when it ends the paused video reactivates.

The example is constructed in such a way that any combination of system description and captions setting will yield a useful combination.

Setting the Pause Behavior

As noted, the default behavior within an exclusive is to terminate the active child when a new child starts. This is because the default value for the priority class settings is stop. An `<excl>` with a priority class defined that exhibits this behavior is shown in the following fragment:

```
<excl dur="indefinite" >
  <priorityClass peers="stop" >
    ...
  </priorityClass>
</excl>
```

The peers in the priority class will stop each other when a new peer starts. The `<excl>` element takes this as the default because it is assumed that this will be the typical use case.

The default behavior can be changed to other values for peers. For example, if we want to pause an active child when a successor is started, we can use:

```
<excl dur="indefinite" >
  <priorityClass peers="pause" >
    ...
  </priorityClass>
</excl>
```



If several priority classes are defined, then the relative behavior of active elements can be defined as follows:

```
<excl dur="indefinite" >
  <priorityClass peers="pause" lower="defer">
    ...
  </priorityClass>
  <priorityClass peers="stop" higher="pause">
    ...
  </priorityClass>
</excl>
```

In this fragment, any peers of the higher level priority class are paused when a new element starts and any request by a lower-level element are deferred until the higher-level element completes. Peers in the lower-level priority are stopped when new elements begin, but these peers are paused when higher level content starts.

Using pauseDisplay to Disable Interaction

Figure 13-4 illustrates a fragment containing two peer videos that are placed in one priority class. The `peers` attribute has been set to *pause*. The video objects render into non-overlapping regions. Each video starts when an associated button has been pushed. Since each video has a link anchor defined, we want to make sure that only the link associated with the active video is enabled. The



```
<excl id="selectMe" dur="indefinite" fillDefault="freeze">
  <priorityClass peers="pause" pausedDisplay="disable" >
    <a href="http://www.example.org/hisHomeSite.smil"
      <video id="v1" begin="btn1.activateEvent" region="videoL" ... />
    </a>
    <a href="http://www.example.org/herHomeSite.smil"
      <video id="v2" begin="btn2.activateEvent" region="videoR"... />
    </a>
  </priorityClass>
</excl>
```

Figure 13-4. Using the `pausedDisplay` attribute.

`pausedDisplay` attribute allows the paused video to remain visible without keeping its link active.

13.3 SMIL 2.0 Timing Model, Revisited

In this and the previous chapters we have introduced many of the advanced concepts used in defining the SMIL 2.0 element activation. In order to put all of these concepts into a common framework, this section provides a comprehensive overview of the SMIL timing model. It concludes with a number of interesting examples.

13.3.1 Element Activation, Duration and Termination

In many ways, there are only three topics of interest when it comes to timed elements: determining when they start, determining how long they last and determining when they end. This section considers all three of these aspects.

Timed Element Activation

A timed element starts its active duration when any one of a number of begin conditions have been met. These conditions can be explicitly defined or they can be inferred from the timing values of the element's parents or other ancestors.

The cardinal rule for determining when and *if* an element gets activated is that no element can become active if its parent is inactive. (And no element can remain active after its parent becomes inactive.) This applies to all SMIL timing behavior: event-based and schedule-based.

The default rules for determining when elements begin depend on the type of element and the type of the parent time container. In order to clarify SMIL behavior, we discuss how each container element is processed separately.

Media objects. A media element gets activated based on the characteristics of the parent time container unless modified by an explicit begin attribute. The begin attribute may specify that a media object will begin based on syncbase, eventbase of linking behavior. Continuous media items, such as audio and video, start by default at the beginning of the object unless modified by a clip-Begin or media marker attribute (if allowed).

The <seq> container. A <seq> container will start at the temporal moment defined by its parent time container, modified by the value of the begin attribute. The <seq> may begin based on scheduled or event timing. The sequential nature of the <seq> container may never be violated by other SMIL timing constructs, by interactive behavior (either link based or event based) or media timing on a begin condition.

Of all of SMIL's time containers, the <seq> provides the most restrictive set of begin rules for its children:

- The implicit start time for each of the children of a <seq> is the end of its predecessor element in the <seq>. For the first element, the implicit start time is the beginning of the <seq>.
- Children of a <seq> may use only a restricted set of values in their begin attribute: only a single, non-negative temporal offset is allowed. This means that a list of (multiple) start times, event timing, syncbase offsets, media marker offsets, accesskey values and wall-clock times are not allowed. The begin time may not be set to *indefinite* and the element may not contain a repeat value. Disallowing media marker offsets is probably an error in the SMIL specification, since media markers provide indexes into the media object rather than the SMIL schedule, but given the limited use of media marker timing, this is probably not a major issue.

The advantage of the <seq> (from a time model perspective) is that it is relatively easy to calculate the effective begin moments of the children. The effective begin of the <seq> itself is as complex as any other SMIL time container.

The <par> container. A <par> container will start at the temporal moment defined by its parent time container, modified by the value of the begin attribute. The <par> may begin based on scheduled or event timing. The begin time may be a wall-clock value or expressed via a syncbase relationship. The begin time may also have the value *indefinite*. The parallel nature of the <par> container does not mean that all of the elements are active at the same time, but that all of its children are bounded by the scope of the <par>'s active duration.

The children of a <par> all have a default start time that is equal to the syncbase (the beginning) of the <par>. The children of a <par> may further make unrestricted use of SMIL temporal semantics to define their begin times. The only operational restriction is that no content may be rendered before the start or after the end of the active duration of the <par>. If a continuous media element is scheduled to begin before the start of the parent <par>, only that portion that falls within the temporal scope of the <par>'s active duration will be rendered.



The <exc1> container. An <exc1> container will start at the temporal moment defined by its parent time container, modified by the value of the begin attribute. The begin time determination for the <exc1> is identical to that of the <par>. The critical difference between a <par> and an <exc1> is that at most one child of an <exc1> may be active at any one time. Nothing may violate this condition. Under certain circumstances, inactive children of an <exc1> may be in a paused state and may respond to certain UI events. For a temporal perspective, they are not active during these periods.

Unlike the children of a <par>, the children have a default begin time of *indefinite*. This means that, unless the time gets resolved during the active duration of the <exc1>, they will never begin. The children of an <exc1> may make unrestricted SMIL temporal semantics to define their begin times. As with the <par>, no content may be rendered before the start of the active duration of the <exc1>. If a continuous media element is scheduled to begin before the start of the parent <exc1>, only that portion that falls within the temporal scope of the <exc1>'s active duration will be rendered.

Dealing with cycles and unknown begin times. It is possible to define a cycle by having the begin times of a collection of objects depend entirely on the begin or end of each other. In these cases, it will be impossible to ever define a resolved begin time. Usually, none of the elements in the cycle will be activated; a SMIL player may reject to start a presentation if a cycle is detected.

If begin times are unresolved, they cannot be used to build a timing instance for the object. Elements may become resolved during the active duration of their parent, but if they are not resolved, they are ignored for timing purposes.

Timed Element Duration

The duration of timed objects depends on several factors: the media-defined duration of an object, the explicit duration given to an object by SMIL's timing attributes and the number of times (or length of time) that an object repeats. Throughout the book we have used the terms *intrinsic duration*, *simple duration* and *active duration* for these concepts.

Under certain circumstances, elements will not have a known duration. This can be because they have end conditions that potentially extend or limit a duration which cannot be resolved when timing calculations are made. Other objects will have unknown begin times; if you don't know when an object will begin, you will not know how long it will last. (You may know how long it will last if it ever gets the chance to start, but this is useless information for a scheduler — it is rather like knowing exactly what you would do with a large sum of money that may never actually arrive.)

In order to create a runtime schedule, SMIL needs to define an *implicit duration* of an object. Each time container has a set of rules for determining its implicit duration.

Media objects. The implicit duration of a media object is equal to the intrinsic duration, modified by any clipBegin/clipEnd interval or media marker tim-

ing. The implicit duration of discrete media is 0 seconds. The default implicit duration of a continuous media object is the intrinsic duration of that object.

The <seq> container. The implicit duration of a <seq> container is the sum of the active durations of its children. If any of the children of a <seq> has a duration of indefinite, the implicit duration of the <seq> is indefinite.

The <par> container. The implicit duration of a <par> depends on the setting of the endsync attribute. By default, the implicit duration of a <par> is the active duration of its longest child plus any resolved begin timing offsets.

The <excl> container. The implicit duration of an <excl> is the same as a <par>, with this difference: the default begin time of the children of the <excl> is *indefinite*, which means that none of them (by default) contribute to the <excl>'s timing. Thus the effective implicit duration is 0. If a resolved begin time exists for any children, this time is used to determine the implicit duration. Since the <excl> supports the endsync attribute, the default implicit duration is the duration of the longest element plus any resolved begin time offset.

Timed Element Termination

For timing purposes, an element will end at the end of its active duration. An element may persist beyond the end of its active duration if it has an effective fill period behavior defined other than remove. During this period it is not active for purposes of the SMIL timing model, but it may be perceived as being active because of its response to UI events.

The end time of an element may be resolved or unresolved. Unresolved end times result from event-based end specifications that have not yet occurred. The end of an element may also be specified to be indefinite. While it is convenient to associate indefinite with forever, a more correct definition of indefinite is a time that will be determined by another context. In terms of timing calculations, a value of indefinite is considered to be greater than any resolved value, and a value of unresolved is considered to be greater than indefinite.

13.3.2 Determining Schedules

The primary role of the scheduler is to determine the active duration of an object and to use that active duration to influence the active duration of any parents and children. There are two cardinal rules that must be taken into account in determining schedules:

- 1) An element without a resolved begin time cannot be scheduled, even if its duration is resolved. Elements without resolved begin times either do not participate in the scheduling computation or they may keep their parent active until their begin times are resolved.
- 2) An element with an end time of *indefinite* will result in an active duration of *indefinite*.



Explicit Value of dur Attribute	Implicit Element Duration	Explicit end Attribute	Simple Duration
unspecified	(not used)	specified (and no repeats specified)	Indefinite
dur="clock-value"	(not used)	(not used)	clock value
dur="indefinite"	(not used)	(not used)	Indefinite
unspecified	resolved	(not used)	implicit element duration
unspecified	unresolved	(not used)	unresolved
dur="media"	resolved or unresolved	(not used)	implicit element duration

Table 13-5. Computing the simple duration.

In addition to these rules, the SMIL 2.0 specification provides a table that can be used to determine the active duration of an object. Table 13-5 is based on the information in the SMIL specification.²

We will now consider computing the active duration in two cases: when no repeat behavior is defined and when repeat behavior is defined. (This distinction is made to simplify the discussion for the easy cases.)

Computing an Active Duration Ignoring Repeat Behavior

We can now determine the active duration for elements, using the following steps:

- 1) If a begin or end time for an element (including the children) is unresolved, the active duration is considered to be *indefinite*. Note that only elements that have unresolved eventbase start/end times or elements with cyclic behavior timing dependencies will have unresolved begin or end times.
- 2) Otherwise, if an explicit and resolved end time other than *indefinite* is specified but no dur attribute was specified, then the active duration is the time value obtained by subtracting the begin time from the end time.
- 3) Otherwise, if an explicit end value of *indefinite* is given and a dur attribute was not specified, the active duration is set to *indefinite*.
- 4) Otherwise, if no end attribute was specified and either no dur attribute was specified or a dur attribute was specified with values of *indefinite* or *media*, the active duration is the same as the simple duration as defined in Table 13-5. In general, this will result in an active duration of 0 seconds for discrete media objects or the intrinsic duration of continuous media objects. It may also result in a value of *indefinite*.

² This table is used to compute the simple duration. If a value is listed as not used, this does not mean it is not used in the computation of the active duration, but simply that the value is not important for determining the (intermediate) simple duration.

- 5) Otherwise, if an end value other than *indefinite* is specified and a dur attribute is provided, the active duration is the smaller of the dur attribute value and the value obtained by subtracting the begin from the end time.

While this all is very complex because of strange cases (such as when various attribute values conflict), the active duration will always be either:

- 0 seconds, for discrete media and objects with unresolved times;
- a duration determined by the dur attribute, the intrinsic duration of continuous media or child elements, or a value derived from subtracting an explicit end time from the effective begin time; or
- the special value *indefinite*.

Note that the implicit duration of the time containers <par>, <seq> and <exc1> is determined by evaluating the children of those containers, as discussed on page 312.

Computing an Active Duration Including Repeat Behavior

Once you understand the process of obtaining an active duration without considering repeats, the extra work needed when the repeatCount or repeatDur attribute is present is minimal. While there are a few special cases (that are beyond the scope of this book), the main extra step is:

- The active duration will be the minimum of the following three values:
 - a) the repeatCount times the computed simple duration (from the table),
 - b) the repeatDur as specified on the element, and
 - c) the value obtained by subtracting the begin time from the end.

If any of these values are unknown, substitute *indefinite*; this will only result in an *indefinite* final result if all three values are *indefinite*.

The active duration is nearly always the repeat count times either the intrinsic object duration or a repeat duration.

Applying min and max to the Active Duration

Once the initial active duration has been computed, one final step is required: the value must be compared with the value of the min and max attributes. The final active duration is the greater of the min attribute value (or 0 if min is unspecified) and the initial active duration, but it may not exceed the max attribute value (or *indefinite*, if no max was specified).

Computing the Final Schedules

Once the active durations for all elements in the presentation are known, a schedule can be determined. This schedule is usually not static: if events are used, the schedule must be recomputed for the dependent sub-tree every time an event condition becomes resolved or unresolved. The schedule may also need to be recomputed when the system or custom test elements provided by SMIL's content control facilities change the set of elements included in a presen-



tation. For presentations with internal links, the schedule will need to be recomputed when a link is followed.

The SMIL 2.0 specification allows multiple begin times to be specified for objects. A separate schedule needs to be computed for each possible resolved begin value. SMIL also allows repeat and restart behavior; this will also influence the schedule, since a significant part of the schedule may need to be recalculated each repeat or restart interval. Good schedulers can optimize this process, but it still all takes time.

13.3.3 Pathological Examples

Most of the interesting scheduling examples are the ones that yield useful active duration results that can then be used to build (efficient) schedules. We have given examples of such usable cases throughout the book. In this section, we look at some strange scheduling cases; these are used to illustrate the complexities of SMIL.

Cycles in Begin and End Lists

The first short example illustrates a simple case of cycles within a document.

```
<par id="P">
  <video id="A" begin="B.begin" ... >
  <image id="B" begin="A.begin" ... >
</par>
```

The `<par> P` will have an effective active duration of 0 because neither of its children have a resolved begin time. If we give `P` a `dur="10s"` attribute, `P` will start and be active for 10 seconds, but neither `A` nor `B` will be scheduled.

This cycle is obvious, but sometimes the cycles are hidden in chains of activations. A good scheduler will detect the chain and then refuse to play the document.

Invalid Begin Times

Consider the following simple fragment:

```
<seq id="S">
  <video id="A" begin="0" ... >
  <image id="B" begin="A.end" ... >
</seq>
```

While this example looks harmless, it will also result in a scheduling problem. The reason is that children of a `<seq>` element may only have begin offsets that consist of non-negative numbers. Syncbase, eventbase and other forms of begin conditions are not allowed.

An `<exc1>` Without Explicit Durations

The following example illustrates the most common problem with `<exc1>` timing. (The `btn1`, `btn2` and `btn3` objects are defined elsewhere in the presentation.) Here, even though the simple durations of each of the objects are defined, the begin times are unresolved. (They are not *indefinite*.) As a result, none of the



objects participate in contributing to *nonStarter*'s active duration. Since it also has no explicit duration declared, it resolves to a duration of 0: the element will be skipped.

```
<exc1 id="nonStarter" fillDefault="freeze">
  <video id="va" begin="btn1.activateEvent" dur="10s" ... />
  <video id="vb" begin="btn2.activateEvent" dur="10s" ... />
  <video id="vc" begin="btn3.activateEvent" dur="10s" ... />
</exc1>
```

Combining dur and end in an Element

SMIL allows the specification of a simple duration on an element and an active duration via the end attribute. This is illustrated in the following fragment. Each of the images will be scheduled based on their implicit begin time and the expected duration of 10 seconds per image. If a mouse event occurs, the schedule will be recomputed and the minimum of the mouse time and the scheduled end time will be used for the object. (The mouse always wins.)

```
<seq>
  <image id="Ia" end="activateEvent" dur="10s" ... />
  <image id="Ib" end="activateEvent" dur="10s" ... />
  <image id="Ic" end="activateEvent" dur="10s" ... />
</exc1>
```

Mixing Synchbase and Eventbase Activation

At the beginning of this chapter, we provided a short example that combined eventbase timing with synchbase timing. (See page 288.) On that same page, we provided a footnote saying that various SMIL players interpreted the timing in this example differently. We close this discussion of the SMIL Timing Model by revisiting this example.

Consider the following SMIL fragment:

```
...
<par id="P0">
  <par id="P1">
    
    <video id="V1" begin="I1.begin+3s" src="story.mpg" ... />
  </par>
  
</par>
...
```

In building a schedule for this code, the following steps are taken:

- The children *P1* and *X* of <par> *P0* start at 0 seconds after the beginning of *P0*. The active duration of *X* is 20 seconds; this defines one of the bounds for the active duration of *P0*.
- The active duration of *P1* depends on the durations of the children. Image *I1* has no explicit duration declared, but an it an intrinsic duration of 0 seconds. The video *V1* also has no explicit duration declared, but assume it has an intrinsic duration of 10s. Although resolved durations exist for *I1* and *V1*, neither has a resolved begin time. As such, neither will contribute



to the active duration of *P1*. *P1* is determined to have an active duration of 0 seconds.

- *P1* will start and then immediately stop. *X* will remain active for 20 seconds, but clicking on it will have no effect because *I1* and *V1* cannot be started outside of the active duration of *P1*.
- After 20 seconds, *X* ends and *P0* terminates.

The key to understanding this example is that elements that have no resolved begin time will have an effective active duration of 0. (Actually, they will simply be ignored for purposes of computing a parent's active duration.) As a result, what looks to be a reasonably simple example turns out to be a complex scheduler problem.

The complexity of this problem is illustrated by the fact that both the Real-One player and HTML+TIME in IE-6 get it wrong. For many users, keeping track of SMIL's complex timing relationships without the support of a sophisticated presentation authoring system can lead to a range of scheduling problems that are difficult to debug.

The SMIL 2.0 specification's discussion on timing is comprehensive but the material remains very complex. In trying to understand the details of timing, dare to trust your own judgements and try your presentation on a range of SMIL players. This will lead to maintainable, future-proof SMIL documents.



13.4 Integrating SMIL Timing in Other Languages

We close this chapter with one last timing topic: how do you integrate SMIL timing into XML languages that are not based directly on the SMIL 2.0 Language profile? 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 with the target language.

The SMIL 2.0 specification supports host language integration using two attributes: `timeContainer` and `timeAction`.

13.4.1 Time Container Integration Issues

One of the advantages of the SMIL 2.0 specification is that it provides a modularized architecture. This allows designers of other XML languages to take SMIL modules and integrate the associated functionality into their languages. This has been done successfully for SVG and XHTML.

One of the problems of the integration of SMIL functionality is to determine how SMIL's time containers can be modelled in the target language. There are essentially two choices: a set of namespace extensions can be defined for a language to add new elements and attributes, or a set of extra attributes can be integrated into the target language that allow special behavior to be associated with any element using those attributes.

While the namespace approach is useful in integrating large numbers of constructs into a language, the use of integration attributes provides a clean method of being able to add limited functionality to existing elements without having to redefine them via a namespace.

A simple example of integrating attribute-based extension functionality is given in the following HTML fragment:

```
<html>
  <P timeContainer="seq">
    <img id="H1" dur="5s" ... />
    <img id="H2" dur="7s" ... />
  </P>
</html>
```

Here, the `<P>` container is defined to have behavior of a SMIL `<seq>`. The children of the container are each given SMIL-like attributes to control their temporal behavior.

Note that it is up to language designers and browser suppliers to create an infrastructure that allows use of time container semantics. As with namespaces and other extension technology, these changes can only be integrated into the presentation environment if they are both supported in the language and in the implementation.

13.4.2 Time Container Integration Attributes

The SMIL *TimeContainerAttributes* module defines two attributes that facilitate integration of SMIL time containers into non-SMIL languages. These are `timeContainer` and `timeAction`. The module also makes the `fill` and the `endsync` attributes available to XML languages that do use SMIL as the host level language.

This section discusses only `timeContainer` and `timeAction`. The `fill` attribute is covered in detail in Chapter 13: *Advanced SMIL Timing Behavior and Control* (see *Fill Behavior Control* on page 328) and the `endsync` attribute is described in Chapter 7: *Basic SMIL Timing* (see *endsync* on page 150).

Attribute: `timeContainer`

The three time containers available in SMIL are `<par>`, `<seq>` and `<excl>`. They define behavior that is used to manage the temporal hierarchy of a SMIL document. In XML languages that have no notion of time or which do not manage time as a structure container, SMIL provides the `timeContainer` attribute. It allows an author to assert that an element in the non-SMIL language should assume SMIL time container semantics.

The legal values for this attribute are:

- *par*: defines a SMIL parallel time container;
- *seq*: defines a SMIL sequential time container;
- *excl*: defines a SMIL sequential time container; and
- *none*: states explicitly that the element *does not* implement a SMIL time container. This is the default value of `timeContainer`.



The `timeContainer` attribute cannot be applied to existing SMIL `<par>`, `<seq>` and `<exc1>` containers. (They would either be redundant or cause havoc.) They may, in principle, be applied to any element in a target host language, but such a language may limit the elements to which this attribute may be applied. The intrinsic duration of an element using `timeContainer` is defined to be equivalent to media object timing. Language designers using `timeContainer` must define how the intrinsic duration is defined.

Note that it is theoretically possible to apply `timeContainer` to a SMIL media object; this allows media objects with children to exhibit time container functionality. While this is useful for direct insertion of media in a presentation (such as supporting in-line text), this is currently not supported in the SMIL Language profile. None of the mainstream SMIL players (RealOne, GRiNS and Ambulant) support the *TimeContainerAttributes* module. It is supported in the HTML+TIME implementation of XHTML+SMIL.

Attribute: `timeAction`

Where the `timeContainer` attribute enables a 'foreign' element to assume SMIL time container semantics, the `timeAction` attribute allows an author to define how those semantics should apply to the associated element. The `timeAction` attribute does not define timing semantics, but it defines which semantics apply to the definitions in the current element.

Language designers may define the values for `timeAction`. The legal values defined in the SMIL specification are:

- *intrinsic*: specifies that the intrinsic timing behavior is controlled;
- *display*: specifies that timing behavior of the display is controlled, as defined by CSS (which can impact the layout of the presentation);
- *visibility*: specifies that timing behavior of the element's visibility is controlled, as defined by CSS;
- *style*: specifies that timing behavior of any in-line style is controlled, as defined by CSS;
- *class:classname*: specifies behavior of the inclusion of the given *classname* is controlled; and
- *none*: specifies that timing behavior does not control any actions.

The default behavior for `timeAction` is *intrinsic*.

SMIL time containers provide scheduling control over their children. Other control is exercised by attributes on the child elements. When using time containers, a scheduling-only behavior is obtained by the use of the *intrinsic* behavior. If the value of *intrinsic* is not selected, then the control of any other scheduling will be done in addition to *intrinsic*. This does not apply to the value of *none*.

For the use and restrictions on `timeAction`, please see the following examples section.



13.4.3 Examples of Time Container Integration

This section give several examples of the syntax and semantics of the time container integration attributes. Note that the application of these examples requires a language implementation that supports time container integration.

The timeContainer Attribute

The use and utility of the `timeContainer` element is strongly dependent on the implementation of the playback browser. In the HTML+TIME implementation of the XHTML+SMIL profile, the use of a `timeContainer` semantic is typically not required, since the language has namespace extensions that recognize basic SMIL containers. This is shown in the following fragment:

```
1 <html xmlns:t="urn:schemas-microsoft-com:time">
2 <head>
3 <style>
4   .time { behavior: url(#default#time2) }
5   .FatFirst:first-letter { font-size:200%; color:blue; float:left; }
6   .FatFirst:first-line { color:red; font-size:24pt; }
7 </style>
8 <?IMPORT namespace="t" implementation="#default#time2">
9 <TITLE>timeContainer and timeAction</TITLE>
10 </head>
11 <body bgcolor="gold">
12 <t:par id="BasicVideoshow">
13   <div class="time" timeContainer="seq" ... >
14     ...
15   </div>
16   <t:excl ... >
17     ...
18   </t:excl>
19 </t:par>
20 </body>
21 </html>
```



This example relies on the inclusion of the *time2* behavior of Internet Explorer 6. On line 9, we see a declaration of a SMIL `<par>` container, qualified with the HTML+TIME *t* namespace identifier. On line 10, we see the declaration of a SMIL sequential container using the `timeContainer` attribute. Line 13 shows the use of namespace extensions to define a SMIL `<excl>`.

In theory, the namespace extended version of a SMIL element can be used as a wrapper around child elements and the `timeContainer` attribute can be used inside of an existing host-language construct. In practice, some situations in HTML+TIME require the use of `timeContainer`, while others require the use of namespace qualified elements. Often, the distinction needs to be found by using trial and error.

Note that in the current implementation, the *time* style, which is linked to the *time2* behavior, needs to be included when `timeContainer` is used. Note also that the `<head>` section contains not only the HTML+TIME behavior declaration, but it also contains an extra style definition used in some of the examples below. We include it here for general reference.



```
<html xmlns:t="urn:schemas-microsoft-com:time" >
  ***
  <body bgcolor="gold">

    <span class="time" timeAction="display" begin="0" dur="2">
      My dog Gretchen and I went for a walk.
    </span>

  </BODY>
</html>
```

Figure 13-5. Using the *display* timeAction attribute value.

The timeAction Attribute

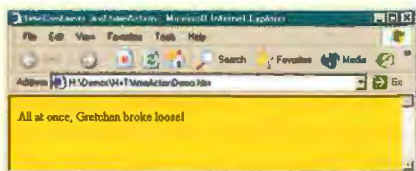
The use of the *timeAction* attribute in conjunction with the *time2* behavior can often lead to interesting presentations. In this section we develop a multi-step example that illustrates several aspects of *timeAction* behavior.

The presentation fragment in Figure 13-5 takes a single line of text and changes the visibility attribute to display it for two seconds.³ An HTML ** element is used to encode a time action that begins at the start of the presentation and lasts 2 seconds. After the 2 seconds pass, the text will be removed.

We illustrate the *class:classname* attribute value for *timeAction* in Figure 13-6. This line is defined to be visible starting at 2 seconds into the presentation and will last for 4 seconds. It has an embedded effect that changes the text styl-



(a)



(b)

```
***
<span class="time" timeAction="display" begin="2" dur="4">
  <p class="time Combo" timeAction="class:FatFirst" begin="2" dur="3.0">
    All at once, Gretchen broke loose!
  </p>
</span>
***
```

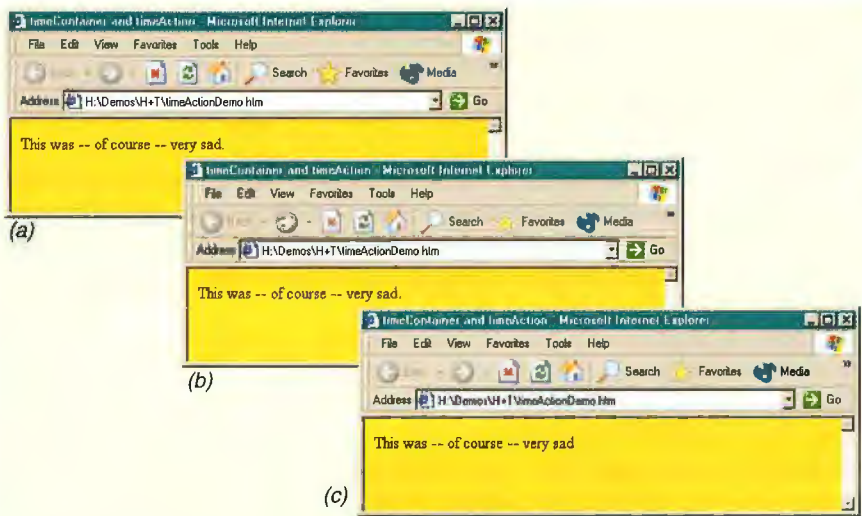
Figure 13-6. Using the *class:classname* timeAction attribute value.

³ Note that to save space, the HTML+TIME *time2* behavior declaration is not repeated. All of the examples use the *<head>* section illustrated in the fragment on page 321.

ing (Figure 13-6(a)). Since the embedded effect only has a duration of 3 seconds, the text is un-styled at the end of its display time (Figure 13-6(a)). Note that the timing in the embedded `<P>` is relative to the start of the document, not the parent `<span>`. This is because the `<span>` is not a time container. (See page 321 for the *FatFirst* class definition.)

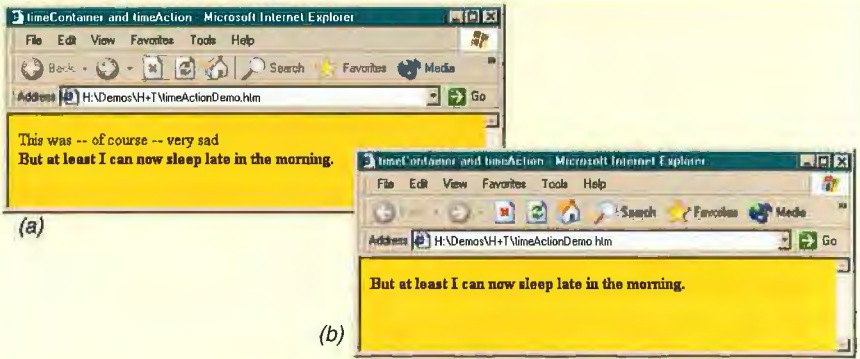
Figure 13-7 illustrates the behavior of the presentation if we give the `<span>` the time container semantics of a SMIL `<seq>` and then define three children that each color in part of a line of text. The `<span>` begins at 6 seconds into the presentation and lasts 6 seconds. Each of the children last 2 seconds. There is one image for each child's active period (Figure 13-7(a), (b) and (c)).

There are two important things going on here. First, the `timeAction` attribute defines both a display setting and the intrinsic timing of the `<seq>` container. Second, each of the *effects* of the `timeAction="style"` attribute on the children are scheduled in sequential order but the combined *content* of the children is visible for the entire 6 second duration of the `<seq>`. (The content is the text string: *This was -- of course -- very sad.*)



```
...
<span class="time" timeContainer="seq"
      timeAction="display" begin="6" dur="6">
  <span style="color:blue" class="time" timeAction="style" dur="2">
    This was
  </span>
  <span style="color:blue" class="time" timeAction="style" dur="2">
    -- of course --
  </span>
  <span style="color:blue" class="time" timeAction="style" dur="2">
    very sad. <br>
  </span>
</span>
...
```

Figure 13-7. Using the *style* `timeAction` attribute value.



```

***
<span class="time" timeContainer="seq"
      timeAction="display" begin="6" dur="6">
  ***
</span>

<span style="color:blue; font-weight:800;" class="time"
      timeAction="visibility" begin="10" dur="4">
  But at least I can now sleep late in the morning.
</span>
***

```

Figure 13-8. Using the *visibility* timeAction attribute value.

The distinction between timed effects and timed content is very important when using time containers and time actions. You should study this example and convince yourself that each of the text blocks turn blue for two seconds, but that the entire text string is visible for the entire period of the <seq> time container.

The last step in this example is to add a visibility attribute value to the set of effects in the presentation. This is illustrated in Figure 13-8. This fragment adds a final sentence to our man-and-his-dog story. Since it is the conclusion, the author puts it in bold. The fragment starts at 10 seconds into the presentation and ends 4 seconds later. The author had initially intended that the display shown in Figure 13-8(b) would result from the code fragment, but note that the begin timing of the fragment overlaps with the last 2 seconds of the previous example. As a result, we first see the two lines shown on Figure 13-8(a) and 2 seconds later (after the earlier part ends), the display gets reformatted to that shown in Figure 13-8(b). Note that as part of the reformatting, the text line moves up in the display: the text is not hidden but actually removed from the display list.

Each of the code fragments developed in this section can be used as stand-alone effects on text (or animations or any other content). They can also all be combined together into a single presentation. The entire example is shown in Figure 13-9.

```

<html xmlns:t="urn:schemas-microsoft-com:time" >
<head>
<style>
.time { behavior: url(#default#time2) }
.FatFirst:first-letter { font-size:200%; color:blue; float:left; }
.FatFirst:first-line { color:red; font-size:24pt; }
</style>
<?IMPORT namespace="t" implementation="#default#time2">
<TITLE>timeContainer and timeAction</TITLE>
</head>
<body bgcolor="gold">

<span class="time" timeAction="display" begin="0" dur="2">
    My dog Gretchen and I went for a walk.
</span>

<span class="time" timeAction="display" begin="2" dur="4">
    <P class="time Combo" timeAction="class:FatFirst" begin="2" dur="3.0">
        All at once, Gretchen broke lose!
    </P>
</span>

<span class="time" timeContainer="seq"
    timeAction="display" begin="6" dur="6">
    <span style="color:blue" class="time" timeAction="style" dur="2">
        This was
    </span>
    <span style="color:blue" class="time" timeAction="style" dur="2">
        -- of course --
    </span>
    <span style="color:blue" class="time" timeAction="style" dur="2">
        very sad. <br>
    </span>
</span>

<span style="color:blue; font-weight:800;" class="time"
    timeAction="visibility" begin="10" dur="4">
    But at least I can now sleep late in the morning.
</span>

</body>
</html>

```



Figure 13-9. The complete example, ready for testing.

It might have been wise for the author to wrap the entire presentation into an outer sequential container and to then use relative timing at the beginning of each of the elements. This seemingly trivial task is left as an exercise to the reader.

13.5 Summary and Conclusion

This chapter has provided a broad review of advanced timing features in SMIL 2.0. We began with a discussion of how event timing has been integrated into the language, and how event processing can be used to control the beginning and end of attribute activation. We then provided a summary of how SMIL's `<excl>` and `<priorityClass>` elements can be used to support selective activation of one out of a set of elements. We then considered the scheduling and

active duration calculations that are required to support general SMIL timing. We closed with a discussion of how time functionality can be integrated by language designers into other XML languages.

The material in this chapter is advanced. If you are new to SMIL or have only a superficial understanding of how SMIL's elements and attributes work, we encourage you to spend time building examples rather than building execution instance lists. At the same time, it is good to understand the intricacies of how event timing and synchbase activation can lead to more interesting and interactive content.

We encourage you to visit the book's Web site to obtain copies of sample presentations that make use of these features. As you become comfortable with our examples, you can start building your own. Ultimately, this should help spice up content on the Web!

13.6 Further Resources

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>.

IE-6's HTML+TIME

For information on the SMIL 2.0 XHTML+SMIL compatible HTML+TIME player in Microsoft's IE-6 browser, see:

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

DOM events

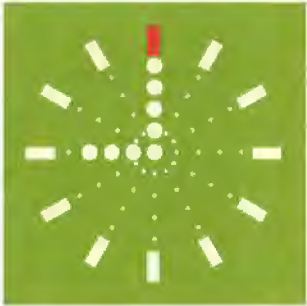
Document Object Model Events, Tom Pixley (ed.), W3C Recommendation, 13 November 2000,

<http://www.w3.org/TR/DOM-Level-2-Events/events.html>.



14

Advanced SMIL Timing Attributes



As explained in previous chapters, SMIL has four time containers that are used to schedule media and SMIL element content: `<par>`, `<seq>`, `<excl>` and `<body>`¹. All of the time containers take a wide range of SMIL timing attributes, plus attributes to control rendering and animation.

In addition to the standard SMIL timing and control attributes, three classes of attributes also exist to control generic aspects of time container processing: *fill behavior*, *restart behavior*, and internal multi-object *synchronization behavior*. SMIL also provides a set of attributes to control how the SMIL is to interpret the pace and direction of rendering: the *time manipulation attributes*. Table 14-1 provides an overview of these attributes and indicates the elements to which they apply.

The fill behavior attributes control rendering behavior of an object after its active duration has ended; this is useful to have object visibility match the duration of other objects in a container or to allow special effects for transitions

attributes		elements	Profiles		
Fill Behavior	fill	animation	M	B	1
	fillDefault	<prefetch>			
Restart Behavior	restart				H
	restartDefault				
Synchronization Behavior	syncBehavior	media object, <brush>			H
	syncBehaviorDefault				
	syncTolerance				H
	syncToleranceDefault	<body>, <seq>, <par>, <excl>			H
	syncMaster				H
Time Manipulation	autoReverse	animation			
	speed	<transition-filter>			H
	accelerate, decelerate				

Table 14-1. Structure of this chapter's attributes

1 The `<body>` element is only a time container in profiles that use SMIL as the host language; profiles based on XHTML+SMIL or SVG do not use `<body>` as a time container.

that are difficult to describe using standard timing. The restart behavior attributes allow an author to control how and when an element restarts with respect to external events or internal SMIL synchbases; this can be used to provide predictable behavior in interactive or event-driven application. The synchronization behavior attributes allow an author to determine which object or element should serve as the master media object for timing control purposes; this allows, for example, the timing of an audio element to pace the rendering of other objects in a presentation. Finally, the time manipulation attributes allow the speed and direction of the SMIL clock to be manipulated in many interesting ways — such as reversing the order of presentation of a media object; these are especially useful for controlling local animations on media objects.

This chapter discusses these advanced timing attributes in detail. Some attributes were introduced in earlier chapters, but all of the behavior aspects of element fill, restart, synchronization and time manipulation attributes are covered below.

14.1 Fill Behavior Control

Chapter 7: *Basic SMIL Timing* introduced SMIL's facilities for controlling object persistence using the `fill` attribute. In that chapter, we considered the simple attribute settings of *freeze* and *remove*. In Chapter 10: *Transition Effects*, as part of the discussion on the need for existing content associated with a transition to extend beyond its normal rendering lifetime, we discussed the *transition* attribute setting for `fill`. In addition to these three values, `fill` also accepts three other values: *hold*, *auto* and *default*. The discussion below unifies our presentation on fill behavior and discusses how all of these attribute values relate to each other. We start with a review of fill behavior, then discuss the attributes and attribute values for `fill` and `fillDefault`, and then look at several examples of using fill behavior.

14.1.1 Understanding the Fill Behavior

Timed objects in a SMIL presentation have several types of durations: their intrinsic, simple, active duration and rendered durations. The intrinsic duration is determined by the media object. The simple duration is determined by taking the intrinsic duration and adding explicit object timing (most often using the `dur` attribute). The active duration is determined by applying repeat behavior to the simple duration. When all of these times have been determined, the setting of the `fill` attribute determines the rendered duration: it defines how long a media object remains visible after its active duration has ended. The rendered duration depends on many things in a SMIL presentation, of which the most important is usually the temporal context of the parent element. This section reviews the important aspects of rendering duration and object persistence.



The Temporal Scope of Objects

The fact that fill behavior is an issue in a SMIL presentation is a direct result of the timed-based nature of SMIL documents. For languages and formats for which time is not central — like HTML or PDF — an object typically remains on the screen until the viewer performs some explicit action: scrolling past the object, following a link to another page or closing the browser. In a SMIL presentation, the temporal scope of an object usually determines its visibility. To understand the rules and issues in determining rendering duration we consider the three temporal topics: the temporal scope of continuous media object, the temporal scope of discrete media objects and the temporal scope of time containers.

The temporal scope of continuous media. In many ways, the temporal scope of a continuous media object provides the easiest and most obvious starting point for defining fill behavior. As illustrated in Figure 14-1, a continuous media object normally will be visible for the period defined by that object's intrinsic duration. Once the intrinsic durations ends, the object is usually removed from the display, although the actual fill behavior will be determined by several factors. If the simple duration is extended by applying an explicit duration which is longer than the intrinsic duration, the final frame of a visual media object will be extended. If the active duration repeats the simple duration, any extended visibility after the simple duration will also be repeated.

Once the active duration ends, the rendered visibility will be determined by a combination of the temporal scope of the parent element and the effective setting of the `fill` attribute. The effective fill behavior can be determined on the element itself or by one of its ancestors. There are, of course, default rules for determining how long an element remains visible.

It is important to realize that the fill behavior only determines the behavior of an object after the active duration: the fact that the last frame of a media object is extended by the simple duration is *not* due to the fill behavior but due to the definition of the `dur` attribute.

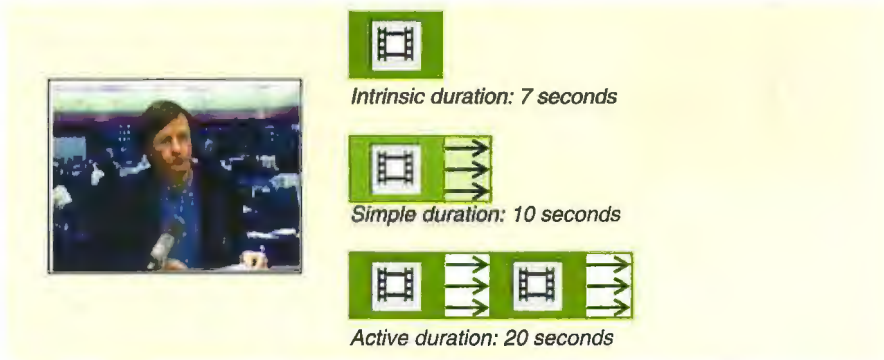
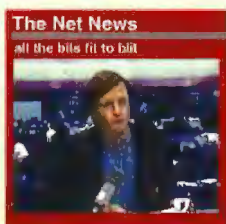


Figure 14-1. Temporal scope of a continuous media object.

The temporal scope of discrete media. The rendering of discrete media — images and text — presents a radically different set of persistence needs than the rendering of continuous media. Many newcomers to SMIL are surprised that the default duration of a discrete media object is 0 seconds.² Considering that the default duration of text and images on an HTML page is effectively infinite, they wonder why anyone would bother defining an image as part of a presentation when it will only be active for 0 seconds!

Although there are many justifications that have been offered for the default duration of 0, the real reason it is used is this: if you don't use 0, what *should* you use? The RealNetworks G2 player for SMIL 1.0 assigned a default duration of five seconds to each discrete media object. While five is a nice number, it is not really any more appropriate for discrete media than, say, three, six or ten seconds. The only consistently useful alternative to 0 seconds is to set the default at the other end of the temporal spectrum: an infinite duration. This is the approach used in applications that support a painter's canvas model: discrete media remains part of the visible portion of the presentation until it is explicitly covered by some other object.

The advantage of an infinite default duration is that you get to see the image. The disadvantage is that a presentation author needs to keep track of all of the objects displayed in the past to make sure that the screen is cleaned up during the presentation. The advantage of a 0 second default duration is that it makes the timing model simple and consistent. The disadvantage of a 0 second default is that it makes discrete content difficult to view unless each object explicitly overrides the intrinsic duration using the *dur* attribute. While specifying an explicit duration is possible, it presents a new problem: you often don't know in advance exactly how long you want some discrete content to stay active — the duration will often be determined by other objects on the screen. This is illustrated in Figure 14-2. Here, we want the image containing the headline text and logo to stay on the screen as long as the continuous media



Intrinsic video duration: 7 seconds

Intrinsic image duration: 0 seconds
Rendered duration: 7 seconds

```
<par>
  <video src="news.mpg"
  
</par>
```

Figure 14-2. Temporal scope of a discrete media object.

2 Actually, the default duration is really ϵ (epsilon) seconds: the smallest time increment available in the player in which an object can be rendered and then removed. The — largely philosophical — distinction between ϵ and 0 seconds allows an object in the presentation to be differentiated from an object that is not included in the presentation.

object is active. We could potentially find out the intrinsic duration of the video, but what we really want to specify is that the image should remain visible as long as the continuous object is active.

SMIL's fill behavior provides an elegant solution to this duration problem. It lets an author say: put a media object on the screen and keep it around until the presentation context changes. Of course, if you want to specify more exact control, SMIL provides the core timing attributes of `begin`, `dur` and `end`.

The temporal scope of a time container. The presentation context used to determine the rendering duration of the image in Figure 14-2 is not determined directly by the duration of the companion video object, but by the duration of the parent time container: the active duration of the parent will determine the visibility of the image. In this example, the active duration of the `<par>` is the duration of the video media object, but the active duration could also have been determined by other timing attributes. In Figure 14-3, we see a parent container with a `dur` attribute: this will determine the fill behavior of the `<par>` but not directly of the video, since the `<par>` fill behavior will start after 20 seconds, while the fill behavior of the video (*freeze*) starts when it ends.

SMIL's fill behavior lets you define the rendering duration of objects that are tied to the duration of the time container in which they are presented — and perhaps also to an entire timing hierarchy. This provides simple but powerful control over how media objects are managed for display.

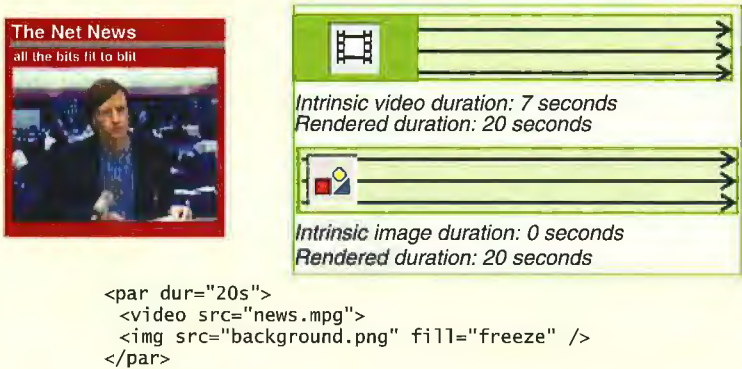


Figure 14-3. Temporal scope of a parent time container.

Linking, Interaction and Fill Behavior

The definition of a rendered behavior is an important concept for SMIL. It determines the composition of objects on the screen, but it also determines how long an object that has some form of linking anchor or interactive behavior associated with it will remain active. In general, elements remain 'clickable' as long as they are visible: the active duration may have ended seconds, minutes or hours ago, but the element remains selectable as long as it is in its fill period.

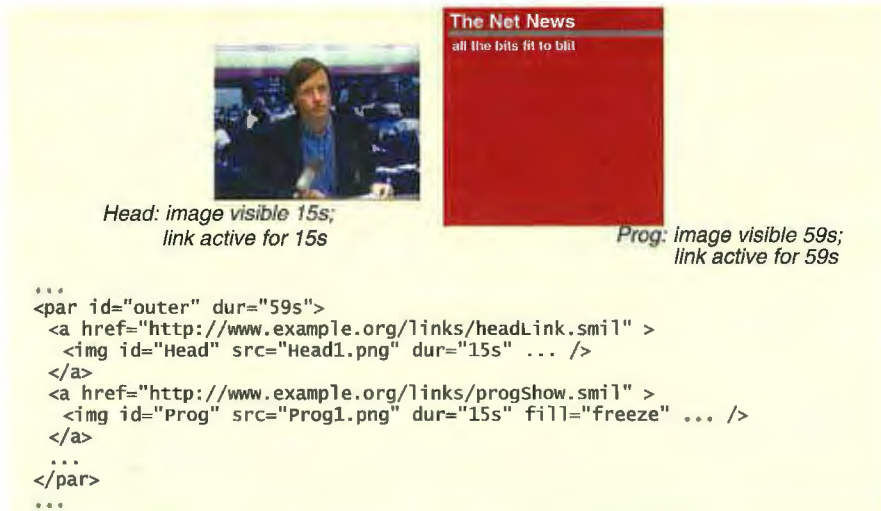


Figure 14-4. Link anchor sensitivity after the active duration.

Figure 14-4 shows an example of how linking behavior extends beyond an object's active duration. In this simple presentation, two images are defined: *Head* and *Prog*. Both are given explicit durations of 15 seconds. Image *Prog* is given `fill="freeze"` behavior, image *Head* is given no explicit `fill` attribute, which means that it has an effective behavior of `fill="remove"`. *Head* and *Prog* are placed in a `<par>` container with an explicit active duration of 59 seconds.

The active durations of *Head* and *Prog* end after 15 seconds. During their active durations, both have active link anchors. After the active duration expires, *Prog*'s anchor remains active for an additional 44 seconds (until the end of the parent `<par>`); *Head*'s anchor is inactive because it has been removed from the display.

This method of specifying the active period of a link anchor is an exceptionally useful way of providing behavior while reducing authoring effort.

Special Cases of Fill Behavior

The most often used value for the `fill` attribute is *freeze*. This keeps the final (or only) frame of a media object rendered until the end of its temporal context. While this is a useful tool, it is not always powerful enough to cover all situations. For example, when working with a transition, you often want to keep content around until after a transition completes, even if this is beyond the temporal scope of the original media. This is the motivation behind the `fill="transition"` behavior. In other situations — especially in a `<seq>` — you occasionally want to keep an object displayed until the end of that sequence. The `fill="hold"` value accomplishes this. As we will see, you should use it sparingly: most often, SMIL's structured timing elements provided a cleaner way of keeping media visible in the context of other media.

14.1.2 Fill Behavior Attributes

This section describes the behavior of two fill-related attributes: `fill` and `fillDefault`. The structure of these attributes, their legal values and the elements and profiles in which they are used is summarized in Table 14-2.

attribute	value		elements		Profiles		
fill	"remove", "freeze", "hold", "transition", "auto",	"default"	media object, <brush>	animation, <par>, <seq>, <excl>, <body>	M	B	1
fillDefault		"inherit"		<prefetch>			

Table 14-2. Structure of fill behavior attributes

Fill Behavior Elements

The specification of fill behavior does not require any special SMIL 2.0 elements. In general, fill behavior is applied to media object elements (such as `<video>`, `<img>`, `<animation>` and `<ref>`) or to SMIL time containers (`<par>`, `<seq>`, `<excl>` and — for SMIL host languages — `<body>`). A fill behavior applied to a time container defines the behavior for all children of that parent unless explicitly overridden by the child.

Attribute: fill

The `fill` attribute controls object persistence after its active duration has ended. It is also used to describe the rendering duration of an object when it has a simple duration that is shorter than the duration specified by the `min` timing attribute, but it *does not* define the rendering behavior of an object when the intrinsic duration is shorter than the simple duration.

The legal values for this attribute are:

- *remove*: remove the element from the display at the end of its active duration. Under some circumstances (described below), this is the effective default behavior.
- *freeze*: do not remove the element from the screen when its active duration ends. The object will remain displayed until the end of the fill period of its parent time container. For continuous media items, the frame that is displayed is the last frame in the last instance of the object's simple duration: if the object was only partially displayed — in terms of duration — then the last displayed frame would be frozen. Under some circumstances (described below), this is the effective default behavior.
- *transition*: keep the object visible until the end of the next transition that uses the same display area. The object is removed when the transition ends.

- *hold*: keep the element visible until the parent element ends. For `<par>` parents, this is equivalent to a behavior of *freeze*, since the ending of the `<par>` triggers the ending of its children. When used with a `<seq>` or `<body>` time container, it keeps the object visible throughout the entire duration of the `<seq>` container.
- *auto*: keep the element visible or remove the element, depending on other attributes on the element. If the element contains a `dur`, `end`, `repeatCount` or `repeatDur` attribute (regardless of the value), then use `fill="remove"` behavior. If the element does not contain a `dur`, `end`, `repeatCount` or `repeatDur` attribute, use `fill="freeze"` behavior. Note that the effective fill behavior based on the presence or absence of the listed attributes occurs only when the `fill="auto"` value has been explicitly defined.
- *default*: determine the fill behavior based on the (effective) value of the `fillDefault` attribute. This is the default behavior when no `fill` attribute is explicitly defined. (As discussed in the next section, the default behavior for `fillDefault` is *auto*.)

In cases where an element is frozen or extended, the actual visibility of the object will depend on other objects rendered during or after the element's definition. In particular, if a frozen object is partially covered by another object that is subsequently removed, the entire frozen object must be re-rendered (refreshed) by the player.

Attribute: fillDefault

The `fillDefault` attribute sets the default fill behavior of that element and its descendants. It accepts the *remove*, *freeze*, *transition*, *hold* and *auto* values of the `fill` attribute, but not the *default* value. It also defines an additional value:

- *inherit*: the associated element inherits the value for `fillDefault` from the parent of this element. If the parent does not exist, use *auto*. The *inherit* value is the default.

Only the SMIL Language profiles uses this attribute.

14.1.3 Examples of Fill Behavior

This section considers various aspects of defining a fill behavior in a SMIL 2.0 document. We organize the examples by the setting of the attribute value. Before we start, a word of caution for readers who use these examples as a basis for their presentations: some SMIL players (such as RealOne) freeze the final state of an entire presentation on the display when the presentation ends, while other players (such as GRiNS and Ambulant) clear the screen at the end. Neither of these approaches are described using SMIL fill semantics, but instead are choices of the various player developers. Both represent legal SMIL behavior, but both choices can influence the apparent fill behavior of the final state of the presentation.

The fill="remove" Value

The *remove* value for fill is used when you want to explicitly remove element content from view. The most obvious form is illustrated in Figure 14-5. Here we see a 10 second <par> that contains a 5 second teaser video and a background image. When the <par> starts, the background image is placed on the screen and the movie starts. At the end of the video, a message of interest to viewers is shown "under" the video.

What if we had not placed the explicit fill behavior on the video element? The element would then be defined as follows:

```
<video src="fiveSecTeaser.mpg" ... />
```

The correct behavior in this case is that a fill value of *default* is assumed. The fill="default" value tells the SMIL player to look at the setting for the fillDefault attribute on the parent <par>. (Note that it is the fillDefault value that is important, not the default fill value; a subtle but important distinction.) In our example, there is no fillDefault defined for the element, so we look to see if a fillDefault is defined for the parent. The <par> also has no fillDefault defined, so we bubble up the tree. Since the tree is fairly short, we can easily determine that the root of the tree (the <body> element) also has no fillDefault defined. In this case, the value of *auto* is used for fillDefault. The consequence is that the video object without a fill attribute and with no fillDefault in its ancestry, acts as if it had been defined as:

```
<video src="fiveSecTeaser.mpg" fill="auto" region="v" />
```

The *auto* behavior is defined as follows:

- 1) If the element explicitly defines a simple or active duration using the dur, end, repeatCount or repeatDur attribute(s), then the fill behavior will be identical to having stated a fill="remove" attribute value; or



```
<smil xmlns="http://www.w3.org/2001/TR/SMIL20" >
```

```
...
<body>
  <par dur="10s">
    <video src="fiveSecTeaser.mpg" fill="remove" region="v" />
    
  </par>
</body>
</smil>
```

Figure 14-5. Example of fill="remove" and fill="freeze" behavior.

- 2) If the element does not explicitly define a simple or active duration using the `dur`, `end`, `repeatCount` or `repeatDur` attribute(s), then the fill behavior will be identical to having stated a `fill="freeze"` attribute value.

Since in our example the video element does not explicitly define a simple duration, `fill="freeze"` behavior is provided.

Assume now that the video element had been specified with an explicit duration but no explicit fill behavior:

```
<video src="fiveSecTeaser.mpg" dur="5s" region="V" />
```

Here, an effective `fill="remove"` value would be used.

The fill="freeze" Value

In the example used in Figure 14-5, the image element that was played in parallel with the video object had a `fill="freeze"` attribute value declared. Note that the image had no explicit duration defined. In terms of its SMIL behavior, this means that it had a simple (and active) duration of 0 seconds and a fill period that lasts until the end of the parent time container. We could have gotten the same visual behavior by explicitly specifying a duration, but this only would have made presentation authoring more cumbersome. In cases where you don't know (or don't care about) the length of companion media object, the *freeze* value on the `fill` attribute is a good way of making objects persistent.

In the example in Figure 14-5, the image element with the `fill="freeze"` attribute was placed inside a `<par>` container. What would have happened in elements were placed in a `<seq>`? This is illustrated by Example 14-1. Here we see a sequence of *n* images that are displayed against a common background. Each of the images in the sequence has a slightly different combination of begin times, durations and fill values. (This was created by trained professionals who apparently had some extra time on their hands.) Luckily, each can be used to identify an important fill-related topic.

The first image (*Pict1.jpg*) begins at the start of the element *I1* in the sequence; it has no duration declared, so its active duration ends immediately. It has a fill behavior defined as *freeze*, which will keep it rendered until the start of the next element in the sequence. The question is: when does the next element in the sequence begin? Is it at the conclusion of the previous element

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

Example 14-1. Example of `fill="freeze"` behavior in a `<seq>`.

(which happens essentially immediately) or does it start when the rendering period of the following element begins? The answer is that the fill period of an object in a sequence ends when the content in the successor element begins its active duration. The active duration in our example begins at 3 seconds after the end of the previous image, thus the fill period is three seconds long.

As a quick aside, it may well be that the fill period of three seconds is exactly what the author wanted, but this is a dangerous way to author the presentation. For example, assume that we replace the initial sequence with the following fragment:

```

***
<seq>
  
  <par>
    
  </par>
  
  ***
  
</seq>
***

```

On the surface, wrapping the definition of *I2* in a `<par>` seems pretty harmless, except that the `<par>`'s active duration will now begin at the moment the active duration of *I1* ends (which is about 0 seconds after it starts); thus the fill period is also 0 seconds, thus you will not see *Pict1.jpg* — or, if you do see it, it will only flash briefly on the screen. Understanding this example is important to understanding the nature of SMIL's fill behavior in a sequence.

Back to the example of Example 14-1. After playing element *I2*, which has an active duration of 5 seconds, a freeze behavior is again declared to tide us over until the start of element *I3*. In this case, the fill period is 0, since the active duration of the successor begins immediately after element *I2* ends. (The `fill="freeze"` is superfluous here.) In a sequence, declaring a `fill="freeze"` behavior only makes sense if the active duration of the successor does not begin immediately.

The final image in this sequence, contained in element *In*, has a duration of 5 seconds. It, too, declares a `fill="freeze"` behavior. Since *I2*'s fill period ends at the end of the `<seq>`, this freeze will have little impact. The only exception is for players such as RealOne that freeze the final state of the presentation on the screen. If this is the last time container in the presentation and if the default behavior would otherwise be `fill="remove"` (because of a `fillDefault` somewhere on an ancestor), having an explicit `fill="freeze"` will keep the image on the display at the end of the presentation. Note that this is only for RealOne. It has no effect for the GRiNS or the Ambulant players.

The fill="hold" Value

The previous section discussed using `fill="freeze"` within a SMIL `<seq>` container. This value has only limited utility — and only under particular circumstances. A more generally useful value for fill in a `<seq>` is *hold*. This value





```

***
<seq>
  
  <seq fill="hold" >
    
    
    
  </seq>
</seq>
***

```

Figure 14-6. Example of `fill="hold"` behavior.

allows content displayed in the sequence to remain active until the end of the entire `<seq>`, not simply until the beginning of a successor's active duration.

The primary advantage of using `fill="hold"` is that it brings a painter's canvas semantic to a sequence. This is illustrated in Figure 14-6. Here we see three images that are displayed in a `<seq>` against a background image. (In the example, all three images are displayed in separate regions, but we also could have used the sub-region positioning features of advanced layout to place them within one region.) The result of the display is shown after the active durations of each of the images have expired. At the end of the `<seq>`, all three images remain on the display. In addition, the `<seq>` also has a `fill="hold"` behavior, so that the contents will stay active until the end of its parent.

The behavior of the `<seq>` in Figure 14-6 could also have been created by placing all of the image elements in a `<par>` and assigning them explicit begin times. This is demonstrated in Example 14-2. In many ways, the `<par>` approach is preferable from a SMIL perspective (although it does require basic addition skills), but the `fill="hold"` functionality was considered useful as an

```

***
<par dur="15s">
  
  
  
  
</par>
***

```

Example 14-2. Example of effective `fill="hold"` behavior in a `<par>`.

```

***
<seq >
  
  
  
  
</par>
***

```

Example 14-3. Example of fill="hold" behavior in a <seq>.

authoring metaphor in the early days of SMIL. (It was similar to the way that RealPix managed image content.)

The code in Figure 14-6 uses dur timing to provide a sequencing of images. We could also have used begin timing. If this was the case, then the final image would still have needed an explicit duration; without one, it would never be visible because the sequence ends as soon as the active duration of the final object ends. The sequence using begin timing is shown in Figure 14-3.

The fill="transition" Value

Where fill="freeze" and fill="hold" allow content to remain active until the end of the parent time containers, another method of rendering control was required for transitions. With transition behavior, content often needs to stay on the screen long enough to last through the transition itself, but not any longer. (The duration of the parent will almost always be too long.) The fill="transition" value was created to support this need.

The *transition* value of fill is useful in a <seq>. It allows a predecessor to remain on the screen until a transition completes. The behavior presented is very similar to fill="hold" except that the temporal scope of the fill period ends at the end of the next transition rather than at the end of the parent.

An example of the specification of transition behavior is given in Example 14-4. The first two images have fill="transition" declared. This will extend their rendering period through the following 1 second fade. The third image does not need a fill because there is no follow-on transition defined.

The main difference between this and other forms of fill is that there are no restrictions on the scope of the transition: it may come in a successor element in

```

***
<seq >
  
  
  
</par>
***

```

Example 14-4. Example of fill="transition" behavior in a <seq>.



the same `<seq>`, but it may also occur much later in the presentation, in another time container. (This would be the case if the final image also had a `fill="transition"` defined.) This provides both a powerful means of providing transition effects and a potential pitfall for authors: if you add a `fill="transition"` to an element and then neglect to define a follow-on transition, the content might stay on the screen during the rest of the presentation.

The `fill="auto"` Value

The main use of the `fill="auto"` value is to serve as an internal default value for the `fill` attribute. It provides a fallback functionality that can be used when no other behavior is defined. The key to understanding the auto behavior is that in most cases when an explicit active duration is defined, it is assumed that this will result in a desire to limit the visibility of an object to that defined period. Thus, when `dur`, `end`, `repeatCount` or `repeatDur` is used to craft an explicit duration, the object is removed when this duration is complete.

If, on the other hand, an implicit duration is used to define timing for the object, the assumption is that the visual duration should also be implicitly extended to the end of the container. As a result, the object is frozen until the end of the parent container. For a detailed walk-through of `fill="auto"` behavior, see the example on page 335.

The `fill="default"` Value

The main use of `fill="default"` is to explicitly state that the value of the `fillDefault` attribute should be applied. Since this is the default when such an attribute has been defined, the value is almost never explicitly used in practice. It is intended to provide a consistent modelling of SMIL fill behavior. For a detailed walk-through of `fill="default"` behavior, see the example on page 335.

Examples of `fillDefault`

The `fillDefault` attribute can be used to assign an explicit default value for all fill behavior on an element and its descendants. Consider the code in Example 14-5. This example, which is a re-coding of Figure 14-6, shows how a single `fillDefault` value can be applied to all of the children. The `fillDefault`

```
***
<seq fillDefault="hold">
  
  <seq >
    
    
    
  </seq>
</seq>
***
***
```

Example 14-5. Example of `fill="freeze"` behavior in a `<seq>`.

attribute can simplify authoring and maintenance of presentations, although it does have the disadvantage that in large, highly nested presentations, the fill value that is applied to an interior node may not always be obvious. As an example, if you overlook the definition of the `fillDefault` attribute in the first `<seq>` of Example 14-5, the persistence of the images defined on the interior nodes will be very bewildering.

14.2 Restart Behavior

The SMIL timing and activation model provides attributes that control when an element starts and stops. It also provides attributes to control duration. With the advent of interactive begin and end times, the timing model was extended to include attributes to control element *restart behavior*. An element can restart if the same event that caused it to start in the first place happens again. Since this behavior can be very useful or very irritating, SMIL provides the `restart` attribute to help authors control when and how restarts should take place. It also provides a `restartDefault` attribute to define default functionality for the `restart` attribute.

Before discussing restart semantics, it is important to note the distinction between restart behavior and repeat behavior. An element can be repeated when it has a `repeatDur` or a `repeatCount` attribute. While a repeated element will restart, this is not the restart behavior addressed by the `restart` attribute. There are similarities in semantics, but for purposes of the discussion below, restart semantics will always be considered to be in response to SMIL 2.0's `begin` attribute behavior.

14.2.1 Understanding Restart Behavior

The basic model used for restart behavior in SMIL is illustrated in Figure 14-7. Here, a video element is started at the beginning of the presentation and is

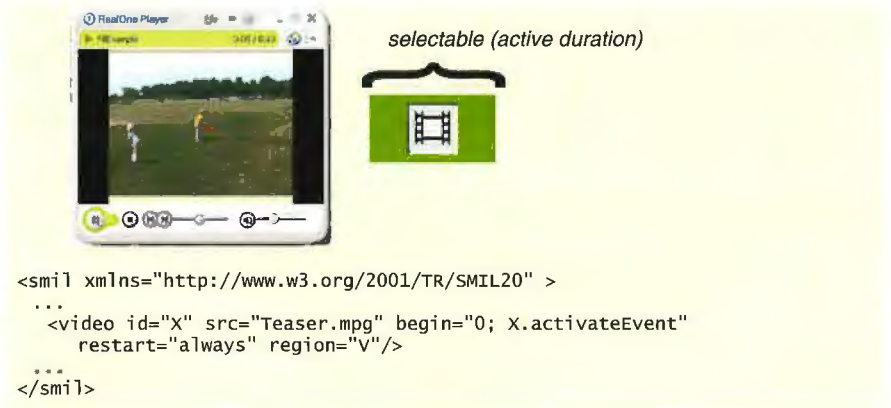


Figure 14-7. Unrestricted restart behavior.

allowed to restart when a user clicks on the video object. The restart attribute is set to *always* to allow restarts, which is usually the default behavior.

The restart attribute allows bounds to be placed on restart behavior of an element. The general choices are to always allow restarts, to never allow restarts, and to allow restarts only when the object is not active. It is not possible to state that an object can be restarted only when it is active, although you can effectively get this behavior by using `fill="remove"` on the object.

There are many subtle aspects of restart behavior that need to be understood by SMIL player developers. Here is a summary:

- *What happens when an active element restarts?:* The first thing that happens when an active element restarts is that it ends its current iteration. While this seems obvious, it has a number of concrete consequences. These include that any other elements that are waiting on an end event of the running object get this event: the element first ends then begins a-new. Once it restarts, any objects that are waiting on a begin event for that element will receive the event. (This may in turn cause *them* to restart.)
- *Can an element be selected for restart during its fill period?:* Yes, assuming that the restart attribute allows restarting. Note that although an element is selectable during the fill period, it is outside its active duration.
- *Is there a notion of state in a presentation, so that a restarted object 'knows' that it has restarted?:* In general, no. The SMIL scheduler certainly maintains state information on the restart history of objects, but once an object restarts, it is considered to be the first (and only) instance of that object. Any fill behavior is reset and any objects that depend on the start of the element in question get restarted/ended.
- *Can an element restart after its parent time container has ended?:* No. Once a parent ends, none of its children are active or can be reactivated, unless the parent time container restarts. The only way that an element can restart after its parent time container has ended is if the parent (or direct ancestor) is restarted.
- *Can a parent time container restart during the active duration of a child object which is not restartable?:* Yes. The restart behavior of a parent container depends on its restart attribute setting. If the parent restarts, all of the children will begin their rendering lifetimes as if they were starting for the first time.

The restart behavior of elements must always be evaluated in the context of their operational environment. The combination of event behavior and restart behavior opens a host of possibilities for interesting media presentations, but the impact on the performance of player may be non-trivial. In general, the restarting of media items that are not stored locally can also have a substantial performance impact on a presentation — streaming servers will usually restart by re-sending a stream from the beginning rather than expecting the player to cache the object.

14.2.2 Restart Behavior Attributes

This section describes the behavior of two restart-related attributes: `restart` and `restartDefault`. The structure of these attributes, their legal values and the elements and profiles in which they are used is summarized in Table 14-3.

attribute	value		elements	Profiles
restart	"always", "never", "whenNotActive"	"default"	media object, <brush>	<par>, <seq>, <excl>, <body>
restartDefault		"inherit"		

Table 14-3. Structure of restart behavior attributes.

Restart Behavior Elements

The specification of restart behavior does not require any special SMIL 2.0 elements. In general, restart behavior is applied to media object elements (such as `<video>`, `<img>`, `<animation>` and `<ref>`) or to SMIL time containers (`<par>`, `<seq>`, `<excl>`) and — for SMIL host languages — `<body>`). A restart behavior applied to a time container does not enable individual child elements to also be restartable; local restart behavior is dependent on the setting of the `begin` attribute of each child.

Attribute: restart

The restart attribute allows an element to define how it should react to restart events. These events may come from SMIL event based timing (as part of one or more `begin` conditions for an element) or, when available, via a restart method in a profile's DOM. The legal values for the `restart` attribute are:

- *always*: The value *always* allows an element to restart from its beginning whenever a restart event or condition exists. The element is restarted even if it is already active.
- *whenNotActive*: The value *whenNotActive* restricts the period during which restarts requests are honored to be outside of the element's active duration. Usually this means the fill period, but it can be any period during which the element's parent is active but the element is not active.
- *never*: The value *never* states that an element may not be restarted during this iteration of the parent. It does not mean that the element will never restart: if the parent restarts, the child may begin again as part of the parent's behavior. SMIL does not have a way of enforcing a 'run only once' semantic (other than by appropriate authoring of a document).
- *default*: The value *default* states that the restart behavior is determined by the value of (the element's) `restartDefault` attribute. This is the default value for the `restart` attribute. Note that since the default behavior for the



`restartDefault` attribute is *always*, the `restart="always"` assignment represents the effective default behavior of the `restart` attribute.

The `restart` attribute is supported by the SMIL 2.0 Language and the XHTML+SMIL profiles.

Attribute: `restartDefault`

The `restartDefault` attribute defines the default restart behavior for an element and its descendants. It accepts the *always*, *whenNotActive* and *never* values of the `restart` attribute, but not the *default* value. It also defines an additional value:

- *inherit*: the associated element inherits the value for `restartDefault` from the parent of this element. If the parent does exist or if no ancestor defined a `restartDefault` value, use *always*. The *inherit* value is the default value for `restartDefault`.

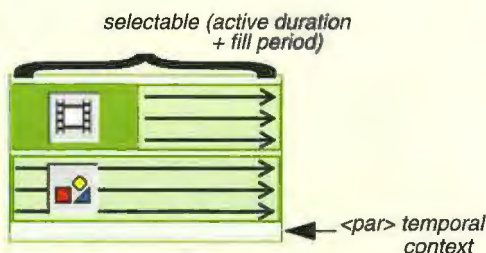
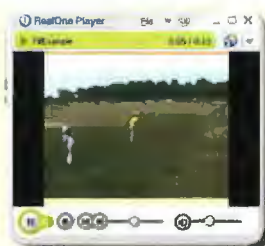
Only the SMIL Language profile uses this attribute.

14.2.3 Examples of Restart Behavior

This section provides various examples of apply restart control behavior. The examples are organized by `restart` attribute value, although some descriptions illustrate the behavior of multiple values. For an overview of general event-based timing and start/end semantics, please see Chapter 13: *Advanced SMIL Timing Behavior and Control*.

Examples of `restart="always"` Behavior

The basic restart behavior used in SMIL 2.0 was illustrated in Figure 14-7 on page 341. The video object could be restarted at any point during its active



```
<smil xmlns="http://www.w3.org/2001/TR/SMIL20" >
...
<body>
  <par dur="20s">
    <video id="x" src="Teaser.mpg" begin="0; x.activateEvent"
      region="V"/>
    
  </par>
</body>
</smil>
```

Figure 14-8. Unrestricted restart behavior extended into the fill period.

duration. Figure 14-8 illustrates an extension of the period of selectivity to include the fill period. A presentation containing a background image and a video is shown in which the video starts at the beginning of the presentation (at 0s after the start of the parent <par>) and either runs to completion and freezes, or is cut off after 20 seconds. The lack of a `restart` attribute and the lack of a `restartDefault` attribute in the parent yields a default behavior of `restart="always"`; the video can be restarted at any point during the 20 second duration of the parent <par>.

Although it is permissible to restart an object during the fill period, the parent of the object to be restarted must be active in order for the child to restart. Figure 14-9 illustrates the (interesting) case where the video *X* is defined as being restartable. *X* has a `fill="transition"` defined, meaning that the rendering of the object will extend past the temporal context of its parent <par> *P1*. Based on the behavior shown in Figure 14-8, one would expect that the object remains selectable during its active duration and the fill interval — or does it? What should happen is that it remains selectable during the active duration only, which also defines the end of the parent <par>. Once the parent ends, the restart behavior ends. The last video frame remains visible during the transition, but it should not be selectable for restart.

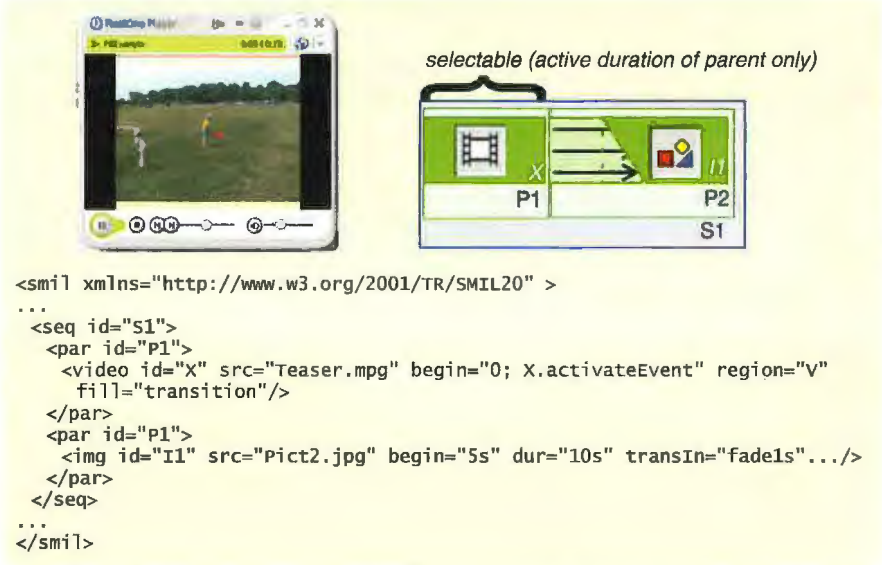


Figure 14-9. The restart behavior only applies while the parent is active.

Examples of restart="whenNotActive" Behavior

In general, an object may be restarted either during its active duration or during its fill period (assuming it has an active parent). In Figure 14-10, the restart

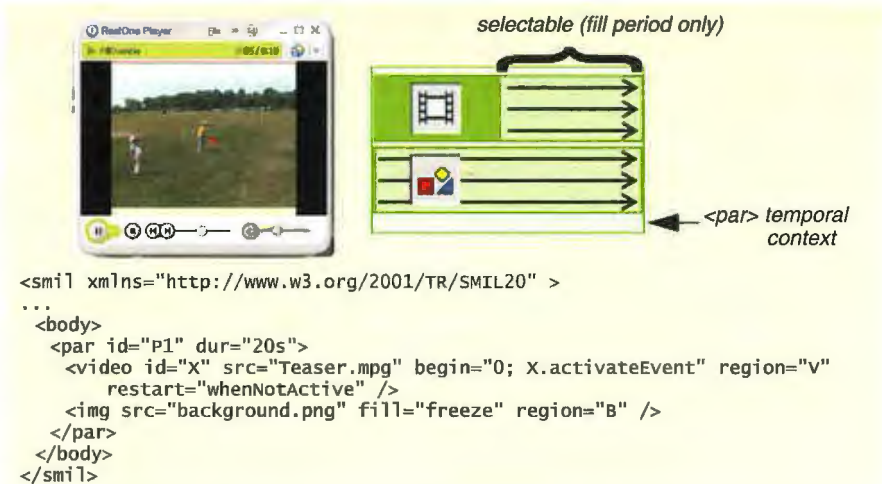


Figure 14-10. Restricting restart behavior to the fill period only.

condition has been set to `restart="whenNotActive"` and the default fill behavior is once again `fill="freeze"`. Under these circumstances, the video will be able to be restarted only during the fill period (which only occurs if the duration of the video is less than 20 seconds).

Note that in this example, the fill period is present because the video object default fill behavior. Recall from the discussion on default fill semantics (see page 335) that if the `dur="20s"` had been placed on the `<video>` as in Example 14-6, the video would not have its final frame used as fill behavior (that is, it would default to `fill="remove"` semantics). In this case, the video would never be able to be restarted, since it has no fill period during which it could be selected. As we will see, this is a cumbersome way of defining `restart="never"` semantics.

```

<smil xmlns="http://www.w3.org/2001/TR/SMIL20/Language" >
...
  <body>
    <par >
      <video id="X" src="Teaser.mpg" begin="0; Y.activateEvent" dur="20s"
        repeat="whenNotActive" region="v"/>
      
    </par>
  </body>
</smil>

```

Example 14-6. Restricting restart behavior to a non-existent fill period.

Examples of `restart="never"` Behavior

The `restart="never"` behavior explicitly states that an object should not be restarted during its active duration or its fill period. This is not a guarantee that the element will never restart, however. Consider Example 14-7. Here, an outer

```

<smil xmlns="http://www.w3.org/2001/TR/SMIL20/Language" >
  ***
  <body>
    <par begin="0s; X.activateEvent">
      ***
      <par>
        <video id="X" src="Teaser.mpg" begin="0"
          restart="never" region="V"/>
        
      </par>
    </par>
  </body>
</smil>

```

Example 14-7. Restricted restart="never" behavior.

<par> element has an event-driven begin specified, based on the video object X. An inner <par> defines this video element with the restart="never" assignment to explicitly turns off a restart. During the activation of this presentation, the video will start. Since the activation of the outer <par> will have the behavior of restarting the inner <par> — but only indirectly the child video of that inner <par> — the video will restart. This is because the restart of a parent will reset any restart status for the descendants of that parent.

14.3 Synchronization Behavior Control

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 allows a presentation to define whether there can be slippage in implementing the composite timeline of the presentation; syncTolerance defines how much slip is allowed; and syncMaster allows a particular element to become the “master timebase against which all others are measured.

This section considers all three attributes plus two additional attributes that control the default behavior of an embedded time line or time container: syncBehaviorDefault and syncToleranceDefault.

We begin with a short overview of synchronization issues. We then consider the individual attributes and values that provide synchronization behavior control. We close this section with several examples of synchronization behavior use.

14.3.1 Understanding Synchronization Behavior Control

In networked media presentations, several components determine the availability of media samples for rendering:

- *Media Server*: the media server is often located on a different computer than that of the media player. If a given media server has many simultaneous



requests pending for delivering media content, it may not be able to sustain a guaranteed delivery rate for data objects within any one stream. This may cause a presentation delay at the client-side player or browser.

- *Network Infrastructure:* the Internet usually provides non-guaranteed, best effort service when delivering data packets associated with a media stream. Individual packets may get lost or may be routed via delay-sensitive routes. Even when used with streaming protocols, packets may either never arrive or they may arrive later than required for optimal playback. This may cause a presentation delay at the client side player.
- *Player Architecture:* the local architecture available to the player will also determine the playback characteristics of a piece of media. If many time-sensitive streams need to be supported in parallel, or if the memory or processor resources available at local clients is insufficient, a presentation delay may result during playback.

While the capacity of the international Internet increases each year, the demands on that capacity also increase. For popular events, even the most powerful media servers can become taxed to the point where they can no longer offer reliable media serving services. And while the perception is that all client-side machines contain enough processing power to handle media presentation, the reality is that the diversity in clients — from desktops through PDA's to telephones — means that there will always be a shortage of resources somewhere in the transmission and delivery pipeline.

Delays in media presentations are usually unavoidable. From a SMIL perspective, since these delays are a fact of presentation life, a mechanism needs to be defined to help an author determine how these delays can best be managed in the content of an individual presentation. In early media players, the internal architecture of the player determined how delays were managed, but as presentations become more complex, a one-size-fits-all approach is not always appropriate.

Figure 14-11 illustrates three cases of synchronization delay problems. In Figure 14-11(a), we see a single video object whose delivery has encountered some sort of delay. A SMIL player will usually have few alternatives in this case: since it can't display frames that don't exist, it can either simply delay the video until the required frames arrive, or it can request that the server send only new frames so that the presentation skips over the unavailable data.

Figure 14-11(b) illustrates a more complex case. Here, two time-sensitive media objects are being delivered, one of which encounters a delay. The media player now needs to decide if the delayed video object needs to impose a similar delay on the (un-delayed) audio object, or whether the audio object can simply continue to be presented. Both of these cases illustrate that a SMIL presentation must manage several levels of time: the time relationships within a media object and the time relationships across media objects and within a presentation.

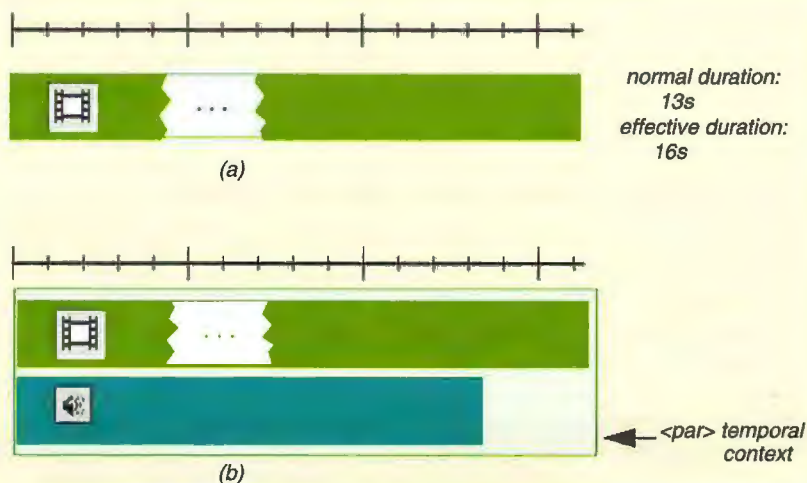


Figure 14-11. Delay during object presentation.

SMIL provides a fine-grained synchronization control model that allows advanced authors to influence how media delivery delays are managed. The `syncBehavior` attribute is used to determine the behavior of an element relative to that of its siblings and parents. Either a hard synchronization relationship can be defined (using the `locked` value) or a soft synchronization relationship can be defined (using the `canSlip` value). While it may seem that a hard relationship is always required, this is often not the case. If there is a delay in presenting background music in a presentation, this is often less important than a delay in supporting synchronized audio and video. By explicitly relaxing synchronization relationships for unimportant objects, the player and the infrastructure can often provide more support for elements that have stricter synchronization needs.

Even when a hard synchronization relationship has been defined, there may be some degree of flexibility available in processing the relative skew among media elements. The `syncTolerance` attribute can be used to determine the amount of skew that is acceptable before the player needs to take action in re-aligning media clocks. An example of when skew may be allowed is when a voice-over is used on top of a video; less skew tolerance exists when lip synchronization is required.

SMIL provides these attributes because fine-grained synchronization is often associated with the semantics of the presentation rather than its SMIL structure. For most uses, a streaming media player will provide a set of defaults that allow it to process information efficiently in most cases. The extra SMIL attributes can be used when an author explicitly wants to direct player resources to maintaining critical relationships at the expense of less critical ones.

In using synchronization behavior support, keep in mind that SMIL deals with media at the inter-object level, not at the intra-object level. If detailed synchronization requirements exist, it may be more useful to resolve these by constructing a composite audio/video object than providing SMIL with two independent media objects.

14.3.2 Synchronization Behavior Attributes

This section describes three synchronization-related attributes: `syncBehavior`, `syncTolerance` and `syncMaster`. It also considers two attributes for defining default behavior: `syncBehaviorDefault` and `syncToleranceDefault`. The structure of these attributes, their legal values and the elements and profiles in which they are used is summarized in Table 14-4.

attribute	value		elements		Profiles	
syncBehavior	"canSlip", "locked", "independent"	"default"	media object, <brush>	<par>, <seq>, <excl>, <body>	L	H
syncBehaviorDefault	"independent"	"inherit"				
syncTolerance	clock value	"default"				H
syncToleranceDefault		"inherit"				
syncMaster	"true", "false"					

Table 14-4. Structure of synchronization behavior attributes.

Synchronization Behavior Elements

The specification of fine-grained synchronization behavior does not require any special SMIL 2.0 elements. The coarse-grained synchronization elements in SMIL are `<seq>`, `<par>` and `<excl>`; these define how the contents of a time container are to behave during a presentation without regard to implementation problems or constraints. The fine-grained control attributes allow a time container to react in a predictable manner to server, transmission or player rendering constraints. In general, synchronization behavior is applied to media object elements (such as `<video>`, `<img>`, `<animation>` and `<ref>`) or to SMIL time containers (`<par>`, `<seq>`, `<excl>` and — for SMIL host languages — `<body>`). A synchronization behavior applied to a time container defines the behavior for all children of that parent unless explicitly overridden by the child.

Attribute: syncBehavior

The `syncBehavior` attribute controls the runtime synchronization scheduling of an element with respect to its parent time container. Ordinarily, a parent time container (such as a `<par>`) defines a strict timeline upon which all of the child elements are scheduled. Sometimes, child elements will not be able to be scheduled precisely because of delays encountered at the media server, across the

network work or at the player. In these cases, the `syncBehavior` attribute can specify whether the entire parent timeline should be artificially delayed to maintain global synchronization or whether the local element can vary within the parent timeline.

The legal values for this attribute are:

- *canSlip*: An element that has *canSlip* declared may slip in time relative to that of the parent time container. This means that the element may be rendered before or (more typically) after its exact scheduled time. This has the consequence that a *soft* synchronization relationship is maintained with the parent — and by extension, with all of its siblings. When soft synchronization is defined, the `syncTolerance` value is ignored. Any delay in this element will not impact the presentation of other elements.
- *locked*: An element that has *locked* declared will remain a strict synchronization relationship with other elements in the parent time container. This means that any delay in one of the children will impact the rendering of all elements. This has the consequence that a hard synchronization relation is defined. The ‘hardness’ of the relationship — that is, the degree to which transient errors are allowed — is defined by the `syncTolerance` attribute. Note that SMIL 2.0 does not proscribe a particular behavior to ensure that locked elements are kept temporally consistent: the approach used by the player architecture may include delaying all elements when one is delayed, or forcing delayed elements to jump ahead in time to remain consistent with other elements.
- *independent*: An element that has *independent* declared defines a local timeline that, while initially scheduled in the context of the parent time container, does not depend on the timing of other elements for its rendering. This has broad consequences, including the responses to pause and interactive behavior of the siblings. For example, it means that it ignores any hyperlink seek operations on its parent (and other ancestors).
- *default*: An element that has *default* declared obtains its behavior from the setting of the `syncBehaviorDefault` attribute. This is the default setting for the `syncBehavior` attribute.

Given the unpredictable nature of network media delivery, it is impossible to define that a given media object should be rendered without problems; the synchronization behavior attribute only defines how the player should respond to any delays associated with that element.

Attribute: `syncBehaviorDefault`

The `syncBehaviorDefault` attribute defines a default synchronization behavior for an element and its children. It accepts the *canSlip*, *locked* and *independent* values of the `syncBehavior` attribute, but not the *default* value. It also defines an additional value:



- *inherit*: the associated element inherits the value for `syncBehavior` from the parent of this element. If the parent does not exist or if no ancestor define a `syncBehaviorDefault` behavior, the synchronization behavior will be player dependent. The *inherit* value is the default value for `syncBehaviorDefault` attribute.

Only the SMIL Language profile uses this attribute.

Note that unlike other default attributes in this chapter, the actual default synchronization used by a media player is implementation dependent. This is a result of the complexity of media player architectures and media playback environment. A value of *locked* is used by the RealOne player as default. A value of *canSlip* is used by GRiNS, Ambulant and many mobile SMIL players.

Attribute: `syncTolerance`

The `syncTolerance` attribute defines tolerance to synchronization delays for time containers in which the synchronization behavior has been declared to be locked. This attribute define the amount of skew or delay that is acceptable before the player needs to enforce re-synchronization of all of the effected elements.

The legal values for this attribute are:

- *clock-value*: this attribute value consists of a SMIL clock value, measured in element simple time, that defines the amount of skew allowed within a hard synchronization relationship. Times are typically seconds or fractions of a second.
- *default*: this attribute value declares that the tolerance is determined by the `syncToleranceDefault` attribute. This is the default value for the `syncTolerance` attribute.

The shorter the synchronization tolerance, the more precise the synchronization relations among peer elements. This can be useful for maintaining lip synchronization, but it may impose a substantial (and unrealistic) performance requirement on the player architecture and the delivery infrastructure. Note that very tight synchronization relationships are often better migrated to the media definition than to the runtime synchronization among independent elements.

Attribute: `syncToleranceDefault`

The `syncToleranceDefault` attribute defines a default synchronization tolerance value for an element and its children. It accepts a clock-value but not the *default* value of the `syncTolerance` attribute. It also defines an additional value:

- *inherit*: the associated element inherits the value for `syncTolerance` from the parent of this element. If the parent does not exist or if no `syncToleranceDefault` was defined, the synchronization tolerance will be player dependent. The *inherit* value is the default.

Only the SMIL Language profiles uses this attribute.

Attribute: syncMaster

The syncMaster attribute can be used to identify which element should be used to identify a master synchronization timing relationship. The syncMaster attribute can be used to inform a SMIL browser of a desired temporal hierarchy within a group of peer elements. The syncMaster only has effect if a hard synchronization relationship has been declared among elements.

The syncMaster attribute accepts the following Boolean values:

- *true*: the associated element is declared the synchronization master.
- *false*: the associated element is declared to be a synchronization slave.

If more than one peer element declares itself the synchronization master, only the first non-paused element encountered will be given master properties.

The syncMaster attribute is *not* used in the SMIL 2.0 Language profiles. At present, it is only supported by the XHTML+SMIL profile.

14.3.3 Examples of Synchronization Behavior Control

The following code fragment defines a <par> element that contains two media objects. Both media object define a hard synchronization relationship with their parent <par>:

```
...
<par>
  <video src="fiveSecTeaser.mpg" syncBehavior="locked" ... />
  <audio src="music.mp3" syncBehavior="locked" ... />
</par>
...
```

In both cases, the hard synchronization relationship is with the parent and not directly with the other children. If there is a delay in either child, the player will need to take action to keep the other object in sync.

Since audio continuity is often more important than video continuity, an author may want to give explicit priority to audio data by specifying the following fragment:

```
...
<par>
  <video src="fiveSecTeaser.mpg" syncBehavior="canSlip" ... />
  <audio src="music.mp3" syncBehavior="locked" ... />
</par>
...
```

It may also be useful to localize synchronization constraints to a portion of a presentation. In the following presentation, an audio object is presented in parallel with a video and an animation:

```
...
<audio src="music.mp3" ... />
...
<par syncBehavior="canSlip">
  <video src="fiveSecTeaser.mpg" syncBehavior="locked" ... />
  <animation src="logo.svg" syncBehavior="locked" ... />
</par>
...
```



The audio will play as a continuous source of background music. The <par> has been defined so that any slippage from its children will not impact (delay) the rendering of the audio. Within the <par> the video and the animation objects have been defined to remain in explicit synchronization with each other via the parent's timeline.

A SMIL player often needs to perform a substantial number of internal checks to make sure that all media elements remain synchronized. The task of the player can be eased — or, perhaps more correctly, this work can be focused — by declaring a portion of the presentation to be beyond the strict temporal scope of the fine-grained synchronization. This is illustrated in the following fragment:

```
...

...
<par>
  <video src="fiveSecTeaser.mpg" ... />
  <audio src="music.mp3" syncBehavior="locked" ... />
</par>
...
```

Here, the background image is independent of the detailed timing of the rest of the presentation. This allows the resources of the player to be focused on the inner media objects. Note that in this example, the synchronization behavior of the video will be determined by the implementation of the player. Most players will enforce either a best effort scheduling policy or an explicit hard synchronization policy. You should check the documentation for the SMIL player to determine the defaults.

In each of these cases, the syncBehavior attribute describes how the player should react to problem situations. If an author wants to guarantee that all media assets are rendered without server or network delays, then these objects will need to be pre-fetched and stored locally. (This may be done using a separate download agent or SMIL's content control facilities, when supported.) Note that the synchronization problem is often greatest on devices that have the least pre-fetching processing capabilities. In these cases, the complexity of the presentation should be reduced.

14.4 Time Manipulation

For most conventional media playback purposes, time is assumed to elapse at a constant pace. The media player does its best to gather all of the elements it needs for the presentation so that a smooth presentation can be created. For some types of media (and for some types of playback) it is useful to consider pace of time as a variable rather than a constant. If the pace of time is increased, the rate at which a media object is played also increases, and if the pace is slowed, the presentation also slows.

SMIL provides support for time manipulation by supplying four attributes: `speed`, `accelerate`, `decelerate` and `autoReverse`. Each of these attributes are discussed in this section.

The implementation support for time manipulation attributes is sparse in SMIL players. Most presentation-centric players provide no support at all: only XHTML+SMIL profile players accept these attributes. In media-centric SMIL (such as SVG), the attributes may be supported but most current SVG players do not provide time manipulation extensions. Only the internal SVG engine in the GRiNS player supports time manipulation. Since there are only a few implementations, we will provide a summary treatment of the various attributes in this section. Interested readers are directed to the SMIL 2.0 specification for more details.

14.4.1 Understanding Time Manipulations

When a parallel element plays, it defines a timeline for all of its children. This timeline defines a temporal framework for the `<par>`. When a child of a `<par>` plays, it often defines a local temporal framework. This framework can be based on the intrinsic duration of the media object and a known sampling rate. It is the responsibility of the SMIL player/browser to implement the conversion of the local temporal framework to that of the parent. Under normal rendering circumstances, the local and parent temporal frameworks use a common notion of time. This common notion is made explicit in the following fragment by providing a `speed="1.0"` factor to each element:

```
...  
<par speed="1.0">  
  <video src="fiveSecTeaser.mpg" speed="1.0" .../>  
  <audio src="music.mp3" speed="1.0" ... />  
</par>  
...
```



Here, the relative speeds of each component's clock are set to be equal.

Such a common notion of the pace of time is not required, however, as long as the player can translate from the local notion of time to that of the parent. Consider the following hypothetical presentation fragment:

```
<par speed="2.0">  
  ...  
</par>
```

The `speed="2.0"` tells the player to play the element (and all of its content) at twice the speed it would otherwise have used. If an attribute value of `speed="0.5"` had been used, the container would have played at half its speed.

The manipulation of time in the previous example involves the *active duration* of an object. All times that are used to define the active duration are manipulated: this includes the intrinsic duration and any duration that is the result of applying the `dur` attribute. A specification of an active duration via a `repeatDur` attribute will also be temporally scaled:

```
<par speed="2.0">
  <video dur="10s" repeatDur="20s" src="ball.mpg" ... />
</par>
```

Here, the active duration will be 10 seconds. In this time, a video that would normally consume 10 seconds of play time will now show in 5 seconds. This means that the video will be repeated two times. (A `repeatCount` is not scaled.)

SMIL extends the behavioral possibilities on media rendering by also defining an `autoReverse` attribute. If set to *true*, an element will run backwards after first completing a forward temporal path. The speed with which an object appears to start and stop — its acceleration/deceleration times — can also be manipulated.

There are several aspects of time manipulation to consider when integrating objects in a presentation. Often, time manipulation works very nicely with media types such as animations: having the animation run faster or slower is not a difficult concept, and having an animation auto-reverse is also clear. For other media, the mappings are less clear either because of the way that the media is stored or the way that it is encoded. Consider a video server that provides streaming support for videos encoded in the MPEG-1 format. Such an external server often can not deliver frames in reverse order. Even if it could supply reverse frames, the decoding of MPEG-1 relies on a forward progression through the video file. Playing a video backwards is conceptually simple but often practically impossible.

The SMIL time manipulations model allows certain media to ignore the manipulation function if it cannot be supported. This itself has consequences for a presentation. In the following fragment, an animation is played in parallel with a video. Both have a `speed="2.0"` set, but the video renderer will ignore this attribute. Unfortunately, since both have been declared to share a hard synchronization relationship with the parent, the player will spend a great deal of effort in re-synchronizing media that is inherently out of sync.

```
***
<par speed="1.0" syncTolerance="0.25s">
  <video src="fiveTeaser.mpg" speed="2.0" syncBehavior="locked".../>
  <audio src="music.mp3" speed="2.0" syncBehavior="locked" ... />
</par>
***
```

In cases such as this, it is best to always use *canSlip* behavior for object synchronization.

Although the time manipulation applies to the active duration of element and its children, it does not impact the timing of the parent. Any begin or end conditions measured in an ancestor's or peer's timeline will not be affected.

14.4.2 Time Manipulation Attributes

This section describes four attributes for manipulating time: `speed`, `accelerate`, `decelerate` and `autoReverse`. The structure of these attributes, their legal values and the elements and profiles in which they are used is summarized in Table 14-5.

attribute	value			elements			Profiles	
autoReverse	"true", "false"			media object, <brush>	animate, <par>, <seq>, <excl>	<transition- filter>		
speed	decimal number ...	≠ "0.0"	"1.0"					
accelerate, decelerate		"0.0" to "1.0"	"0.0"					

Table 14-5. Structure of the time manipulation attributes.

Time Manipulation Elements

The specification of fine-grained time manipulation does not require any special SMIL 2.0 elements. The time containers, media objects, transition filters and animation elements may all potentially use the time manipulation facilities of SMIL although, as noted, support is not widespread. A time manipulation behavior applied to a time container defines the behavior for all children of that parent unless explicitly overridden by the child. The total affect can accumulate is multiple levels of time manipulation are defined.

Attribute: autoReverse

The `autoReverse` attribute specifies whether an attribute should reverse the logical direction of rendering at the end of its simple duration. This only takes effect if the simple duration is shorter than the active duration and will only have meaning if the element can support reverse rendering.

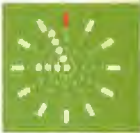
The `autoReverse` attribute allows the following values:

- *true*: The assignment `autoReverse="true"` causes an element's media to play once forward, and then once backwards. As a result, it's simple duration is doubled. If the active duration has not expired and if a repeat attribute has been specified, the forward/reverse rendering behavior will continue.
- *false*: The assignment `autoReverse="false"` causes an element to not reverse at the end of its simple duration. This is the default behavior.

The simple duration of the object must be resolved before the reverse motion can take place.

Attribute: speed

The `speed` attribute defines the local playback speed of an element relative to that of its parent time container. The value may be fractional, indicating a rela-



tive decrease in speed. Negative values are also allowed; they will cause an element to play in the reverse direction.

The speed attribute accepts the following value:

- **number:** this number defines the playback speed in element time. It is a relative value (since the speed of the parent may also have been modified.) A value of 0 is valid — it *does not* mean that the clock is frozen; it will be ignored. The default value is 1.0, meaning the same rate as the parent.

The simple duration of the object must be resolved before a negative speed manipulation can take place.

Attribute: accelerate

The `accelerate` attribute specifies a perceptual acceleration factor that is applied to the beginning of a temporal pass through an object. If a decelerate factor is also used, the sum of the acceleration and deceleration must be less than or equal to 1.0.

The `accelerate` attribute accepts the following value:

- **number:** this value defines a real number in the range 0.0-1.0 that is interpreted as a time bound during which an acceleration function will apply to a media object. The smaller the number, the quicker the acceleration. The default value is 0, meaning no acceleration.

If the sum of the acceleration and deceleration factors is greater than 1, both will be ignored. If a repeat is specified on the element, the `accelerate` will be applied on each iteration.

Attribute: decelerate

The `decelerate` attribute specifies a perceptual deceleration factor that is applied to the end of a temporal pass through an object. If an `accelerate` factor is also used, the sum of the acceleration and deceleration must be less than or equal to 1.0.

The `decelerate` attribute accepts the following value:

- **number:** this value defines a real number in the range 0.0-1.0 that is interpreted as a time bound during which a deceleration function will apply to a media object. The smaller the number, the quicker the deceleration. The default value is 0, meaning no deceleration.

If the sum of the acceleration and deceleration factors is greater than 1, both will be ignored. If a repeat is specified on the element, the `decelerate` will be applied on each iteration.

14.4.3 Examples of Time Manipulation

This section provides several examples of the use of time manipulation attributes. Please note that while the format of elements and attributes is that of the SMIL Language profile, significant support for time manipulation is only provided in the XHTML+SMIL profile.



Specifying the Speed Factor

The following fragment illustrates several properties of using the speed attribute on elements in a time container:

```
***
<par speed="2.0">
  <video src="fiveSecTeaser.mpg" speed="-0.5" .../>
  <audio src="music.mp3" speed="0.5" ... />
</par>
***
```

The speed on the <par> element is defined to be twice its nominal speed. The video defines a speed of a negative 50%, meaning that it will play backward at an effective normal playback speed: 50% of 2.0. The audio element will play at normal speed in the forward direction: 50% of 2.0.

Specifying an Acceleration and Deceleration Factor

The element in the following fragment will have an acceleration and deceleration effect over 40% of their simple duration. The effect will be repeated for each of the four iterations of the object.

```
***

  <animateMotion dur="10s" accelerate="0.2" decelerate="0.2"
    by="10,0" repeatCount="4" .../>
</img>
***
```

The motion animation will provide an acceleration during the first 2 seconds, then move at a slightly faster than normal rate for 6 seconds and then slow-down during the final 2 seconds.

Specifying an Auto-Reverse

In the previous example, if 20 second video is moved, the motion can also be repeated in a pendulum-like motion by specifying an auto-reverse function:

```
***
<video dur="20s" src="flags.mpg" ... >
  <animateMotion dur="20s" accelerate="0.2" decelerate="0.2"
    by="10,0" repeatCount="4" autoReverse="true" .../>
</img>
***
```

Each time the movement reverses, the acceleration and deceleration takes place.



14.5 Summary and Conclusion

This chapter provides an overview of several advanced timing control attributes for controlling fill behavior, restart behavior, synchronization behavior and time manipulation.

The fill behavior attributes allow the visual persistence of objects to be extended past the active duration. They can also be extended past the duration of their parent time container. Of all of the attributes discussed in this chapter, a `fill="freeze"` setting will probably be the most used advanced timing effect in SMIL.

The restart behavior attributes allow the integration of event-based timing in a presentation to be controlled in a coordinated fashion. Once events are allowed, a mechanism is required to stop multiple events from (re)starting an attribute. The model for event behavior was developed in Chapter 13: *Advanced SMIL Timing Behavior and Control*.

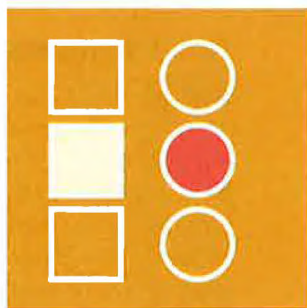
The synchronization behavior attributes in SMIL allows a presentation author to define critical sections of a presentation directly rather than relying on default behavior from the player. The main use of these attributes will be to control which sections of a presentation are *not* sensitive to exact synchronization requirements so that the players resources can be focused on key code sections.

We concluded this chapter with a discussion of advanced timing attributes for controlling the interpretation of the system clocks in a hierarchical manner. This facility, which is especially useful for animation, is currently only broadly supported by Microsoft's IE-6 renderer for XHTML+SMIL. Future support can be expected in SVG implementations.



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 containing 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 element but process the content (its child elements) as if the unknown element did not exist.

(2) If the element containing 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"
12      target="destination" skip-content="false">
13      <video id="F" src="snarf.mpg" region="b"/>
14    </a>
15    
16  </par>
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 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 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"
12      target="destination" skip-content="false">
13      <video id="F" src="snarf.mpg" region="b"/>
14    </a>
15    
16  </par>
17  ...
18 </body>
19 </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				
	skip-content	L			
	core				
parent	<head>	L			
child	<customTest>	L			

Table 16-1. Structure of the `<customAttributes>` 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.

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.

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

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

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 if 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"

```

1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
2   <head>
3     <customAttributes>
4       <customTest id="Annotations" title="View Annotations"
5         defaultState="true" override="visible"/>
6       <customTest id="medicalStaff" title="Medical Staff Use"
7         uid="http://www.example.org/Doctors/list.txt"
8         defaultState="false" override="hidden"/>
9     </customAttributes>
10    ...
11  </head>
12  <body>
13    ...
14    <switch>
15      <par systemBitrate="56000">
16        <video region="Image" dur="11s" src="video/Symptoms.mpg"/>
17        
19          <animateMotion ... />
20          <animate attributeName="width" ... />
21          <animate attributeName="height" ... />
22        </img>
23      </par>
24    <seq>
25      <par dur="5s">
26        
27        
29      </par>
30    </seq>
31  </switch>
32  ...
33  <par dur="29s" customTest="medicalStaff">
34    <video region="Image" dur="29s" src="video/ScalpelBreaks.mpg"/>
35    
37  </par>
38  ...
39 </body>
40 </smil>

```

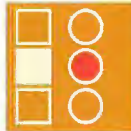


Figure 16-1. Using custom test attributes.

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 *GRI*NS 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.

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.

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

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

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

```
1 <smil xmlns="http://www.w3.org/2001/SMIL20/Language">
2   ...
3   <par>
4     
5     <prefetch id="case1" src="closingLogo.png" mediaSize="100%" dur="10s"/>
6     <prefetch id="case2" src="bigVideo.mpg" bandwidth="80%" />
7   </par>
8   ...
9   <seq>
10    <video id="Feature" src="bigVideo.mpg" region="M" end="activateEvent"/>
11    
12  </seq>
13  ...
14 </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 inheritance 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
 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
"palms" 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
pausedDisplay 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
- "replac" 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
- systemLanguage 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
- UNETT 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
xml:ns 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