

# Office Open XML

**Part 3: Primer**

**December 2006**



## Table of Contents

2	<b>Foreword</b> .....	xi
3	<b>Introduction</b> .....	xii
4	<b>1. Scope</b> .....	1
5	<b>2. Introduction to WordprocessingML</b> .....	2
6	2.1 Stories.....	2
7	2.2 Basic Document Structure.....	2
8	2.3 Main Document Story .....	3
9	2.3.1 Document Backgrounds .....	3
10	2.4 Paragraphs and Rich Formatting .....	4
11	2.4.1 Paragraphs.....	4
12	2.4.2 Runs.....	5
13	2.4.3 Run Content .....	7
14	2.4.4 Formatting Property Values .....	8
15	2.5 Tables.....	9
16	2.5.1 Introduction .....	9
17	2.5.2 Table Properties .....	10
18	2.5.3 Table Grid .....	11
19	2.5.4 Table Rows and Cells.....	12
20	2.5.5 Table Layout .....	14
21	2.5.6 Fixed Width Tables.....	15
22	2.5.7 AutoFit Tables.....	15
23	2.5.8 Complex Table Example .....	16
24	2.5.9 Vertically Merged Cells.....	17
25	2.6 Custom Markup.....	19
26	2.6.1 Smart Tags.....	19
27	2.6.2 Custom XML Markup.....	21
28	2.6.3 Structured Document Tags .....	23
29	2.7 Sections .....	26
30	2.7.1 Section Properties .....	27
31	2.7.2 Section Breaks .....	28
32	2.8 Styles .....	29
33	2.8.1 Styles Part.....	29
34	2.8.2 Style Definitions .....	29
35	2.8.3 Paragraph Styles.....	30
36	2.8.4 Character Styles.....	32
37	2.8.5 Linked Styles.....	33
38	2.8.6 Numbering Styles .....	36
39	2.8.7 Table Styles.....	36
40	2.8.8 Default Document Paragraph and Character Properties .....	40
41	2.8.9 Style Inheritance .....	40
42	2.8.10 Style Application.....	41
43	2.8.11 Latent Styles .....	42
44	2.9 Fonts.....	43
45	2.9.1 Font References .....	43

1	2.9.2	Font Reference Types.....	43
2	2.9.3	Ambiguous Characters .....	44
3	2.9.4	Font Table.....	44
4	2.9.5	Font Substitution Data .....	45
5	2.9.6	Font Embedding .....	45
6	2.9.7	Theme Fonts.....	46
7	2.10	Numbering.....	46
8	2.10.1	Numbering Part.....	47
9	2.10.2	Numbering Definitions .....	47
10	2.10.3	Abstract Numbering Definitions.....	47
11	2.10.4	Numbering Definition Instances.....	50
12	2.10.5	Applying Numbering to Paragraphs .....	51
13	2.10.6	The Complete Story.....	54
14	2.10.7	Numbering Styles .....	55
15	2.10.8	Referencing Numbering Styles.....	56
16	2.11	Headers and Footers .....	59
17	2.11.1	Header Part .....	60
18	2.11.2	Footer Part .....	60
19	2.11.3	Headers and Footers .....	60
20	2.11.4	Multiple Sections.....	63
21	2.11.5	Empty Header or Footer.....	64
22	2.12	Footnotes and Endnotes .....	64
23	2.12.1	Footnote Part .....	65
24	2.12.2	Endnote Part .....	65
25	2.12.3	Footnotes and Endnotes .....	66
26	2.12.4	Footnote and Endnote Types .....	67
27	2.12.5	Footnote and Endnote Reference.....	71
28	2.13	Glossary Document .....	72
29	2.14	Annotations .....	73
30	2.14.1	Introduction .....	73
31	2.14.2	Inline Annotations .....	74
32	2.14.3	Cross-Structure Annotations .....	74
33	2.14.4	Property Annotations .....	75
34	2.14.5	Comments .....	76
35	2.14.6	Comments Part.....	78
36	2.14.7	Revisions.....	78
37	2.14.8	Bookmarks.....	79
38	2.14.9	Range Permissions .....	80
39	2.14.10	Spelling and Grammar .....	81
40	2.15	Mail Merge .....	81
41	2.15.1	Mail Merge, WordprocessingML, and Hosting Applications.....	82
42	2.15.2	Connecting Documents to an External Data Source .....	82
43	2.15.3	Populating Merged Documents with External Data.....	83
44	2.16	Settings .....	85
45	2.16.1	Document Settings .....	85
46	2.16.2	Compatibility Settings .....	86
47	2.16.3	Web Settings .....	87
48	2.17	Fields and Hyperlinks.....	87

1	2.17.1 Fields .....	87
2	2.17.2 Hyperlinks.....	88
3	2.18 Miscellaneous Topics.....	88
4	2.18.1 Text Boxes .....	88
5	2.18.2 Subdocuments.....	88
6	2.18.3 Importing External Content.....	89
7	2.18.4 Roundtripping Alternate Content .....	90
8	<b>3. Introduction to SpreadsheetML.....</b>	<b>93</b>
9	3.1 Workbook.....	93
10	3.1.1 Overview .....	93
11	3.1.2 Minimum Workbook Scenario .....	93
12	3.1.3 Example Workbook Properties .....	93
13	3.1.4 fileVersion .....	95
14	3.1.5 workbookView .....	95
15	3.2 Sheets .....	96
16	3.2.1 Minimum Worksheet Scenario .....	96
17	3.2.2 Example Sheet.....	96
18	3.2.3 Sheet Properties.....	97
19	3.2.4 Sheet Data .....	98
20	3.2.5 Supporting Features.....	102
21	3.2.6 Sheet Properties.....	103
22	3.2.7 sheetData Cell Table.....	103
23	3.2.8 Row.....	103
24	3.2.9 Cell.....	104
25	3.2.10 Supporting Sheet Features.....	106
26	3.2.11 Defined Names.....	106
27	3.2.12 AutoFilter.....	106
28	3.2.13 Merged Cells.....	107
29	3.2.14 Conditional Formatting .....	107
30	3.3 Shared String Table.....	108
31	3.3.1 Overview .....	108
32	3.3.2 File Architecture .....	109
33	3.3.3 Example: Plain Text .....	110
34	3.3.4 Illustration .....	110
35	3.3.5 The XML.....	110
36	3.3.6 Shared String Table .....	117
37	3.3.7 Cell Table .....	118
38	3.3.8 Example: Rich Text .....	119
39	3.3.9 Illustration .....	119
40	3.3.10 Shared String Table .....	119
41	3.4 Tables.....	122
42	3.4.1 Overview .....	122
43	3.4.2 File Architecture .....	122
44	3.4.3 Example: Table .....	123
45	3.4.4 Illustration .....	123
46	3.4.5 The Sheet XML.....	124
47	3.4.6 The Table XML.....	124

1	3.5 Calculation Chain .....	125
2	3.5.1 Overview .....	125
3	3.5.2 Example .....	125
4	3.6 Comments .....	129
5	3.6.1 Overview .....	129
6	3.6.2 Example .....	129
7	3.6.3 File Architecture .....	130
8	3.6.4 The XML.....	130
9	3.6.5 Authors.....	132
10	3.6.6 Comments .....	132
11	3.7 Styles .....	133
12	3.7.1 Overview .....	133
13	3.7.2 File Architecture .....	133
14	3.7.3 Organization in the Styles Part.....	134
15	3.7.4 Example .....	137
16	3.8 Worksheet Metadata .....	149
17	3.8.1 Overview .....	149
18	3.8.2 File Architecture – Relationships.....	151
19	3.8.3 Example .....	151
20	3.9 Pivot Table, Pivot Cache, and Common Types .....	164
21	3.9.1 Feature Overview .....	164
22	3.9.2 File Architecture .....	166
23	3.9.3 Example - Native with Range Source .....	167
24	3.10 Shared Workbook Revisions.....	184
25	3.10.1 Overview .....	184
26	3.10.2 How It Works.....	184
27	3.10.3 Example .....	185
28	3.11 Query Tables.....	194
29	3.11.1 Overview .....	194
30	3.11.2 Web Query Example.....	194
31	3.11.3 Text Import Example .....	194
32	3.11.4 Access Table Example.....	195
33	3.12 External Connection .....	196
34	3.12.1 Overview .....	196
35	3.12.2 OLAP Connection.....	197
36	3.12.3 Pivot XML fragment.....	198
37	3.12.4 Connection XML .....	198
38	3.12.5 Web Query .....	199
39	3.12.6 QueryTable XML.....	200
40	3.12.7 Connection XML .....	200
41	3.12.8 Unused Connection.....	201
42	3.12.9 ODBC .....	201
43	3.12.10 Connection XML .....	201
44	3.12.11 SQL.....	201
45	3.12.12 Connection XML .....	203
46	3.12.13 Text Import.....	203
47	3.12.14 Connection XML .....	204
48	3.13 External Links.....	205

1	3.13.1 Overview .....	205
2	3.13.2 Formula Example.....	205
3	3.13.3 Sheet XML .....	205
4	3.13.4 Workbook Relationships .....	207
5	3.13.5 Supporting Workbook Cache (Cell C2) .....	208
6	3.13.6 External Link (Cell C2).....	209
7	3.13.7 Supporting Workbook Cache (Cell B2) .....	209
8	3.13.8 External Link (Cell B2).....	210
9	3.13.9 Hyperlink Example.....	211
10	3.13.10 Worksheet XML.....	211
11	3.13.11 Relationship.....	211
12	3.14 Volatile Dependencies.....	212
13	3.14.1 Overview .....	212
14	3.14.2 File Architecture - Relationships .....	212
15	3.14.3 Example .....	212
16	3.15 Custom XML Mappings.....	214
17	3.15.1 Overview .....	214
18	3.15.2 File Architecture - Relationships .....	215
19	3.15.3 Conceptual Model.....	216
20	3.15.4 Example .....	216
21	3.16 Formulas.....	222
22	3.16.1 Introduction .....	222
23	3.16.2 Constants.....	222
24	3.16.3 Operators .....	222
25	3.16.4 Cell References.....	224
26	3.16.5 Functions .....	225
27	3.16.6 Names.....	225
28	3.16.7 Types and Values.....	225
29	3.16.8 Error values .....	225
30	3.16.9 Dates and Times .....	226
31	3.16.10 XML Representation.....	228
32	<b>4. Introduction to PresentationML .....</b>	<b>229</b>
33	4.1 Basics .....	229
34	4.1.1 Introduction .....	229
35	4.1.2 Basic Utilities .....	230
36	4.1.3 The Presentation Object .....	232
37	4.1.4 Presentation Properties .....	238
38	4.2 Slides, Masters, Layouts, and Placeholders.....	242
39	4.2.1 Introduction .....	242
40	4.2.2 Masters.....	242
41	4.2.3 Presentation Slide .....	245
42	4.2.4 Notes Page .....	246
43	4.2.5 Slide Layouts.....	247
44	4.3 Comments .....	247
45	4.3.1 Introduction .....	247
46	4.3.2 Functional Overview.....	248
47	4.3.3 Comment Author List .....	248

1	4.3.4 Comment List .....	249
2	4.4 Animation .....	249
3	4.4.1 Introduction .....	249
4	4.4.2 Slide Transitions .....	250
5	4.4.3 Timeline Overview.....	251
6	4.4.4 Timeline Construction.....	252
7	4.4.5 Animation Behaviors.....	254
8	4.4.6 Conditional Properties .....	256
9	4.4.7 Build Animations .....	257
10	4.5 Slide Synchronization .....	258
11	4.5.1 Introduction .....	258
12	4.5.2 Slide Update Info.....	258
13	<b>5. Introduction to DrawingML.....</b>	<b>261</b>
14	5.1 Basics .....	261
15	5.1.1 Introduction .....	261
16	5.1.2 Overview .....	261
17	5.1.3 Basic Elements.....	261
18	5.1.4 Colors.....	261
19	5.1.5 Compatibility .....	262
20	5.1.6 Locked Canvas .....	262
21	5.2 Audio and Video .....	262
22	5.2.1 Introduction .....	262
23	5.2.2 Functional Overview.....	262
24	5.2.3 DrawingML Syntax.....	263
25	5.3 Styles .....	264
26	5.3.1 Introduction .....	264
27	5.3.2 Shared Style Sheet.....	265
28	5.4 Text.....	278
29	5.4.1 Introduction .....	278
30	5.4.2 Overview .....	279
31	5.4.3 Body Level Properties.....	281
32	5.5 Tables.....	289
33	5.5.1 Introduction .....	289
34	5.5.2 Table Styles.....	289
35	5.5.3 Table Definition .....	296
36	5.6 3D Aspects .....	300
37	5.6.1 Introduction .....	300
38	5.6.2 3-D .....	300
39	5.6.3 Styles .....	306
40	5.7 Coordinate Systems and Transformations .....	310
41	5.7.1 Introduction .....	310
42	5.7.2 Coordinate System .....	310
43	5.7.3 Shape Transformations .....	310
44	5.7.4 Group Transformations.....	313
45	5.7.5 Nesting Transformations.....	317
46	5.7.6 Transformation Matrices.....	318
47	5.8 Shape Properties and Effects .....	319

1	5.8.1	Introduction .....	319
2	5.8.2	Color Models .....	319
3	5.8.3	Color Transforms.....	325
4	5.8.4	Fills.....	327
5	5.8.5	Line Properties .....	332
6	5.8.6	Effects.....	334
7	5.9	Shape Definitions and Attributes .....	338
8	5.9.1	Introduction .....	338
9	5.9.2	The Coordinate Systems.....	339
10	5.9.3	Specifying a Preset Shape .....	340
11	5.9.4	Specifying a Custom Shape .....	342
12	5.10	Pictures.....	347
13	5.10.1	Introduction .....	347
14	5.10.2	Specifying a Basic Picture .....	347
15	5.10.3	Attaching Properties to this Picture .....	349
16	5.10.4	Transforming this Picture .....	350
17	5.11	WordprocessingML Drawing .....	352
18	5.11.1	Object Anchoring.....	352
19	5.11.2	Text Wrapping .....	353
20	5.12	SpreadsheetML Drawing .....	355
21	5.12.1	Introduction .....	355
22	5.12.2	Overview .....	355
23	5.12.3	Worksheet Drawings.....	355
24	5.13	Charts.....	358
25	5.13.1	Overview .....	358
26	5.13.2	XML Overview .....	368
27	5.13.3	Example.....	369
28	5.14	Chart Drawing.....	372
29	5.14.1	Introduction .....	372
30	5.14.2	Overview .....	373
31	5.14.3	Chart Drawings.....	373
32	5.15	Diagrams.....	374
33	5.15.1	Introduction .....	374
34	5.15.2	Element Property Set .....	375
35	5.15.3	Data Model.....	377
36	5.15.4	Color Transforms.....	381
37	5.15.5	Style Definition .....	387
38	5.15.6	Layout.....	390
39	<b>6.</b>	<b>Introduction to VML.....</b>	<b>416</b>
40	6.1	Introduction.....	416
41	6.2	Shape Element.....	416
42	6.2.1	Geometry .....	417
43	6.2.2	Placement.....	419
44	6.2.3	Formatting.....	422
45	6.2.4	Other .....	422
46	6.3	Group Element .....	424
47	6.4	ShapeType Element.....	424

1	6.5 VML Usage in the Office Open XML Format.....	425
2	6.5.1 OfficeArt Shapes.....	425
3	6.5.2 SpreadsheetML Comments.....	427
4	6.5.3 WordprocessingML Text Box .....	428
5	<b>7. Introduction to Shared MLs.....</b>	<b>431</b>
6	7.1 Math .....	431
7	7.1.1 Accent Object.....	431
8	7.1.2 Bar Object.....	432
9	7.1.3 Border Box Object .....	432
10	7.1.4 Box Object .....	432
11	7.1.5 Delimiters .....	433
12	7.1.6 Equation Array Object.....	433
13	7.1.7 Fraction Object.....	434
14	7.1.8 Function Apply Object.....	434
15	7.1.9 Group Character Object .....	434
16	7.1.10 Upper and Lower Limits .....	435
17	7.1.11 Matrix Object .....	435
18	7.1.12 N-ary Object .....	436
19	7.1.13 Phantom Object .....	436
20	7.1.14 Radical Object.....	437
21	7.1.15 Scripts (Superscript, Subscript, SubSuperscript, PreSubSuperscript) .....	437
22	7.2 Metadata .....	438
23	7.2.1 Metadata Properties .....	439
24	7.2.2 Core Properties .....	440
25	7.2.3 Extended Properties.....	440
26	7.2.4 Custom Properties.....	440
27	7.2.5 Variant Types.....	440
28	7.3 Custom XML Data.....	440
29	7.4 Bibliography.....	441
30	7.4.1 Types of Sources.....	441
31	7.4.2 Child Elements.....	442
32	7.4.3 Author .....	444
33	7.4.4 LCID, Guid, Tag, and RefOrder.....	445
34	<b>8. Miscellaneous Topics .....</b>	<b>447</b>
35	8.1 Additional Characteristics.....	447
36	8.2 Embeddings .....	448
37	8.2.1 Embedded Packages.....	448
38	8.2.2 Embedded Objects .....	448
39	8.2.3 Embeddings in a WordprocessingML Document.....	449
40	8.2.4 Embeddings in a SpreadsheetML Document .....	451
41	8.2.5 Embeddings in a PresentationML Document.....	452
42	8.3 Future Extensibility.....	453
43	8.3.1 Terminology .....	453
44	8.3.2 What is Future Extensibility?.....	454
45	8.3.3 Future Extensibility Requirements .....	454
46	8.3.4 Future Extensibility Constructs .....	455
47		

# 1 Foreword

2 This multi-part Standard deals with Office Open XML Format-related technology, and consists of the following  
3 parts:

- 4     • Part 1: "Fundamentals"  
5     • Part 2: "Open Packaging Conventions"  
6     • **Part 3: "Primer"(this document)**  
7     • Part 4: "Markup Language Reference"  
8     • Part 5: "Markup Compatibility and Extensibility"

# 1 Introduction

- 2 This Part is one piece of a specification that describes a family of XML schemas, collectively called *Office Open*  
3 *XML*, which define the XML vocabularies for word-processing, spreadsheet, and presentation documents, as  
4 well as the packaging of documents that conform to these schemas.
- 5 The goal is to enable the implementation of the Office Open XML formats by the widest set of tools and  
6 platforms, fostering interoperability across office productivity applications and line-of-business systems, as  
7 well as to support and strengthen document archival and preservation, all in a way that is fully compatible with  
8 the large existing investments in Microsoft Office documents.

# 1. Scope

2 This Part contains a detailed introduction to the following Office Open XML topics:

- 3     • WordprocessingML
- 4     • SpreadsheetML
- 5     • PresentationML
- 6     • DrawingML
- 7     • VML
- 8     • Various shared MLs

9 The organization of this Part is much the same as its corresponding reference Part, Part 4, and is intended as a  
10 gentle introduction to Part 4.

## 2. Introduction to WordprocessingML

2 This clause is informative.

3 This clause contains a detailed introduction to the structure of a WordprocessingML document.

### 4 2.1 Stories

5 A WordprocessingML document is composed of a collection of stories. Each *story* represents a distinct region  
6 of text within the document. The following kinds of region exist: comment (§2.14.5), endnote (§2.12.2),  
7 footer (§2.11.2), footnote (§2.12.1), frame, glossary document (§2.13), header (§2.11.1), main story (§2.2),  
8 subdocument (§2.18.2), and text box (§2.18.1).

9 With one exception (a glossary document), all stories in a document utilize a common set of properties that  
10 determine the presentation of the contents of each story. These properties include font information, style  
11 definitions, numbering definitions, and document settings.

### 12 2.2 Basic Document Structure

13 The main document story of the simplest WordprocessingML document consists of the following XML  
14 elements:

- 15 • document — The root element for a WordprocessingML's main document part, which defines the  
16 main document story.
- 17 • body — The container for the collection of block-level structures that comprise the main story.
- 18 • p — A paragraph.
- 19 • r — A run.
- 20 • t — A range of text.

21 A *run* is a region of text in a story with a common set of properties. The text in a WordprocessingML document  
22 must be contained within one or more runs. A *paragraph* is a collection of one or more runs that is displayed  
23 as a unit. A run must be contained within a paragraph.

24 Consider the following Main Document XML for a simple WordprocessingML document:

```

1   <?xml version="1.0"?>
2   <w:document xmlns:w="...">
3       <w:body>
4           <w:p>
5               <w:r>
6                   <w:t>Hello, world.</w:t>
7               </w:r>
8           </w:p>
9       </w:body>
10      </w:document>

```

## 2.3 Main Document Story

The contents of the main document story—the only story that is required in a valid WordprocessingML document—are encapsulated within the body element. The content of the main document body is a collection of block-level structures, which are those WordprocessingML elements that can contain and/or be sibling elements with a WordprocessingML paragraph.

Within the document body, the valid set of block level content is:

- Paragraphs
- Tables
- Custom markup (custom XML, structured document tags)
- Section properties
- Annotations (comments, revision markers, range permission markers)
- Alternate format chunks

Each of these block-level content constructs (the 'building blocks' of WordprocessingML) is defined in the following subclauses.

### 2.3.1 Document Backgrounds

As well as containing a body, a document element can also contain the definition of the document's background via the background element and its contents. This background applies to all printed pages within this document. A document background in WordprocessingML can have a single color, as well as the application of various drawing effects such as color gradient or pattern, and a tiled or stretched image. All background information in a WordprocessingML document is stored using the Vector Markup Language (VML) syntax. The single exception to this is the background color, which is stored natively in WordprocessingML using the bgColor attribute.

Consider a simple background in WordprocessingML, which consists of a single color with a gradient fill applied:

```

1   <w:background w:bgColor="5C83B4">
2     <v:background id="_x0000_s1025" o:bwmode="white" fillcolor="#5c83b4
3       [3204] o:targetscreensize="800,600">
4       <v:fill color2="fill darken(118) method="linear sigma" focus="100%">
5         type="gradient"/>
6     </v:background>
7   </w:bgPict>
```

8 The background consists of two components: a background fill color of RGB value 5C83B4, and the background  
9 gradient stored as a VML transformation.

## 10 2.4 Paragraphs and Rich Formatting

### 11 2.4.1 Paragraphs

12 The most basic unit of block-level content within a WordprocessingML document, *paragraphs* are stored using  
13 the p element. A *paragraph* defines a distinct division of content that begins on a new line. A paragraph can  
14 contain three pieces of information: optional paragraph properties, inline content (typically runs), and a set of  
15 optional revision IDs used to compare the content of two documents.

16 Consider the paragraph fragment "*The quick brown fox jumped ...*" which is centered on a paragraph. As all the  
17 text in the paragraph is emphasized using italics, in the XML, the contents of the paragraph will have the  
18 justify-center property, and each run within the paragraph (as well as the run properties for the paragraph  
19 mark) stores the italics property; for example:

```

20   <w:p>
21     <w:pPr>
22       <w:jc w:val="center"/>
23       <w:rPr>
24         <w:i/>
25       </w:rPr>
26     </w:pPr>
27     <w:r>
28       <w:rPr>
29         <w:i/>
30       </w:rPr>
31       <w:t>The quick brown fox jumped...</w:t>
32     </w:r>
33   </w:p>
```

34 Notice that each run specifies the character formatting information for its contents, and the paragraph  
35 specifies the paragraph level formatting (the center-justification). It is also notable that since leading and  
36 trailing whitespace is not normally significant in XML, some runs require a designating specifying that their  
37 whitespace is significant via the xml:space element.

- 1 A paragraph's properties are specified via the pPr element. Some examples of paragraph properties are  
 2 alignment, border, hyphenation override, indentation, line spacing, shading, text direction, and widow/orphan  
 3 control.
- 4 It should also be noted that a pPr element may contain a set of run properties within a rPr element – these  
 5 properties are applied to the run which contains the glyph which represents the paragraph mark and not the  
 6 entire paragraph.

## 7 2.4.2 Runs

- 8 The next level of the document hierarchy is the *run*, which defines a region of text with a common set of  
 9 properties, represented by the r element. An r element allows the producer to combine breaks, styles, or  
 10 formatting properties, applying the same information to all the parts of the run.
- 11 Just as a paragraph can have properties, so too can a run. All of the elements inside an r element have their  
 12 properties controlled by a corresponding optional rPr run properties element, which must be the first child of  
 13 the r element. In turn, the rPr element is a container for a set of property elements that are applied to the rest  
 14 of the children of the r element. The elements inside the rPr container element allow the consumer to control  
 15 whether the text in the following t elements is bold, underlined, or visible, for example. Some examples of run  
 16 properties are bold, border, character style, color, font, font size, italic, kerning, disable spelling/grammar  
 17 check, shading, small caps, strikethrough, text direction, and underline.

18 Consider the following run within a WordprocessingML document:

```
19   <w:r>
20     <w:rPr>
21       <w:b/>
22       <w:i/>
23     </w:rPr>
24     <w:t>quick</w:t>
25   </w:r>
```

- 26 The run specifies two formatting properties in its run contents: bold and italic. These properties are therefore  
 27 applied to all content within this run.
- 28 A producer can break a run into an arbitrary number of smaller runs, provided each smaller run uses the same  
 29 set of properties, without changing the content of the document.

30 Consider the content "only one word is emphasized" in a WordprocessingML document. An efficient producer  
 31 could choose to output this content using two runs, as follows:

```
32   <w:r>
33     <w:t xml:space="preserve">only one word is </w:t>
34   </w:r>
```

```

1   <w:r>
2     <w:rPr>
3       <w:i/>
4     <w:rPr>
5       <w:t>emphasized</w:t>
6   </w:r>
```

7 However, a less efficient producer might use four runs, as follows:

```

8   <w:r>
9     <w:t>only one</w:t>
10    </w:r>
11    <w:r>
12      <w:t xml:space="preserve"> word is </w:t>
13    </w:r>
14    <w:r>
15      <w:rPr>
16        <w:i/>
17      <w:rPr>
18        <w:t>empha</w:t>
19    </w:r>
20    <w:r>
21      <w:rPr>
22        <w:i/>
23      <w:rPr>
24        <w:t>sized</w:t>
25    </w:r>
```

26 Although the latter example uses four runs rather than two, the net run information applied to each region of  
27 text is identical, and both are equally valid.

28 Of course, a run might need to be broken. For example, the properties of only some the text in that run are  
29 changed, requiring the changed part to be put into its own run. Another example involves the insertion of  
30 some sort of marker into the middle of an existing run. That requires the run be broken into two with the  
31 marker inserted between them.

32 The following run contains two sentences:

```

33   <w:r>
34     <w:t>Hello, world. How are you, today?</w:t>
35   </w:r>
```

36 If the first two words are bolded in these sentences, the run will need to be broken into two runs in order to  
37 store the formatting, as follows:

```

1   <w:r>
2     <w:rPr>
3       <w:b/>
4     </w:rPr>
5     <w:t xml:space="preserve">Hello, world. </w:t>
6   </w:r>
7   <w:r>
8     <w:t>How are you, today?</w:t>
9   </w:r>

```

10 Apart from text, a run can also contain numerous kinds of textual content (§2.4.3) A run can also contain a set  
 11 of revision IDs used for document "merge and compare".

## 12 **2.4.3 Run Content**

13 The lowest level of this hierarchy is *run content*, that content that can be stored within a single run in a  
 14 document. In WordprocessingML, the types of run content include:

- 15 • Text
- 16 • Deleted text
- 17 • Soft line breaks
- 18 • Field codes
- 19 • Deleted field codes
- 20 • Footnote/endnote reference marks
- 21 • Simple fields
- 22 • Page numbers
- 23 • Tabs
- 24 • Ruby text
- 25 • DrawingML content
- 26 • Embedded objects
- 27 • Pictures

### 28 **2.4.3.1 Text**

29 The most common run content is the t element, which is the container for the text that makes up the  
 30 document's content. A t element can contain an arbitrary amount of text, up to and including the entire  
 31 document's contents. However, typically, long runs of text are broken up into paragraphs and strings of text  
 32 having different formats, or are interrupted by line breaks, graphics, tables, and other items. A t element must  
 33 be enclosed within an r element; i.e., a run of text. An r element can contain multiple t elements, interspersed  
 34 among other elements.

35 Aside from the t element, there are three types of text in WordprocessingML:

- 36 • delText - Deleted text
- 37 • instrText - Field codes

- 1       • delInstrText - Deleted field codes
- 2   These four types of text are defined using unique elements in WordprocessingML so that simple consumers  
3   can determine the text of the document simply by grabbing the contents of the t node, without needing to  
4   check where revisions start and end, etc. to determine the state of the text contents.
- 5   It is also notable that these are the only elements in a WordprocessingML document's main document part  
6   that can contain a XML text node.

#### 7   **2.4.4      Formatting Property Values**

- 8   Most of the children of an rPr or pPr element have a single val attribute that is limited to a specific set of  
9   values. For example, the b (bold) element causes the text that follows it to be bold when the b element has a  
10   val attribute with value on. If the val attribute isn't present for the b element, it defaults to "on". Therefore,  
11   <w:b/> is equivalent to <w:b w:val="on"/>.
- 12   Aside from the default values, which are documented with each element, this is particularly important when  
13   specifying the difference between omitting a formatting property and explicitly turning it off.
- 14   For example, consider the following run:

```
15       <w:r>
16        <w:rPr>
17          <w:b w:val="off"/>
18        </w:rPr>
19        <w:t xml:space="preserve">Hello, world. </w:t>
20      </w:r>
```

21   This run explicitly declares that the bold property is turned off for this text, as opposed to the following run:

```
22       <w:r>
23        <w:t xml:space="preserve">Hello, world. </w:t>
24      </w:r>
```

25   This run says nothing about the bold property. This distinction is particularly important when dealing with  
26   content that is formatting using styles - if the content was not contained in a styled paragraph, both would be  
27   identical. However, in the case where the paragraph is styled, the former would never be bold regardless of  
28   the style information, whereas the latter would express the bold property as set by the style, since it's  
29   omission of the bold property means "whatever the underlying formatting is".

30   Some elements have val attributes that offer a richer set of choices than on and off; the u (underline) element  
31   is one such element. In this case, the same rules apply, the omission of the property simply means use the  
32   underlying properties.

## 1    2.5    Tables

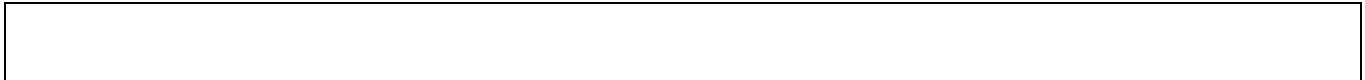
2    Another type of block-level content in WordprocessingML, A *table* is a set of paragraphs (and other block-level  
3    content) arranged in *rows* and *columns*.

### 4    2.5.1    Introduction

5    Tables in WordprocessingML are defined via the `tbl` element, which is analogous to the HTML `<table>` tag.  
6    The `table` element specifies the location of a table present in the document.

7    A `tbl` element has two elements that define its properties: `tblPr`, which defines the set of table-wide properties  
8    (such as style and width), and `tblGrid`, which defines the grid layout of the table. A `tbl` element can also  
9    contain an arbitrary non-zero number of rows, where each row is specified with a `tr` element. Each `tr` element  
10   can contain an arbitrary non-zero number of cells, where each cell is specified with a `tc` element.

11   Consider an empty one-cell table (i.e.,; a table with one row, one column) and 1 point borders on all sides:



12

13   This table is represented by the following WordprocessingML:

```

14    <w:tbl>
15      <w:tblPr>
16        <w:tblW w:w="5000" w:type="pct"/>
17        <w:tblBorders>
18          <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>
19          <w:left w:val="single" w:sz="4" w:space="0" w:color="auto"/>
20          <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>
21          <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>
22        </w:tblBorders>
23      </w:tblPr>
24      <w:tblGrid>
25        <w:gridCol w:w="10296"/>
26      </w:tblGrid>
27      <w:tr>
28        <w:tc>
29          <w:tcPr>
30            <w:tcW w:w="0" w:type="auto"/>
31          </w:tcPr>
32          <w:p/>
33        </w:tc>
34      </w:tr>
35    </w:tbl>

```

1 This table specifies table-wide properties of 100% of page width (tblW's type attribute specifies how the width  
 2 value in the w attribute shall be interpreted—pct specifies a measurement of fiftieths of a percent) and the  
 3 set of table borders (tblBorders), the table grid which defines a set of shared vertical edges within the table  
 4 (discussed later), and a single row.

## 5 2.5.2 Table Properties

6 The tblPr element defines table-wide properties, properties which are applied to each row and cell in the  
 7 table. The complete set of table-wide properties can be found on the definition for the tblPr element.

8 Consider the following simple WordprocessingML table:

--	--

9

10 This table defines outside and inside table borders, etc; and is set to 100% of page width - both table-wide  
 11 properties. The resulting table is represented by the following WordprocessingML:

```
12 <w:tbl>
13   <w:tblPr>
14     <w:tblW w:w="0" w:type="auto"/>
15     <w:tblBorders>
16       <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>
17       <w:left w:val="single" w:sz="4" w:space="0" w:color="auto"/>
18       <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>
19       <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>
20       <w:insideH w:val="single" w:sz="4" w:space="0" w:color="auto"/>
21       <w:insideV w:val="single" w:sz="4" w:space="0" w:color="auto"/>
22     </w:tblBorders>
23   </w:tblPr>
24   <w:tblGrid>
25     ...
26   </w:tblGrid>
27   <w:tr>
28     ...
29   </w:tr>
30 </w:tbl>
```

31 In this example, the tblW element defines the total width of the table, which, in this case, is set to a type of  
 32 auto, which specifies that the table should be sized to fit its contents. The tblBorders element specifies each  
 33 of the table's borders, and specifies a one point border on the top, left, bottom, right and inside horizontal and  
 34 vertical border. The table-wide properties can be overwritten on an individual row basis by specifying table  
 35 property overrides within the table row properties.

### 1    2.5.3      Table Grid

2    The `tblGrid` element defines the *grid* for the table. All columns in the table (including the space before and  
 3    after a row) reference this grid. Each `gridCol` defines a single grid column within the table's layout, which is  
 4    used to define the presence of a vertical line within the table. A `tblGrid` element can contain an arbitrary  
 5    number of `gridCol` elements, where each `gridCol` element represents one grid column in the table and defines  
 6    a single grid entry. When cells are laid out within this table, as discussed below, all cells will be forced to snap  
 7    the shared column edges defined by this grid.

8    Returning to the earlier 'one-cell empty table' example, the table has one column with a width of 10,296  
 9    twentieths of a point. This measurement (twentieths of a point, or twips) is frequently used in  
 10   WordprocessingML, and translates to 1/1440th of an inch (one-twentieth of a point, which is itself 1/72nd of  
 11   an inch):

```
12     <w:tblGrid>
13       <w:gridCol w:w="10296"/>
14     </w:tblGrid>
```

15   Consider the following, more complex table that has two rows and two columns; the columns are not aligned:


16

17   This table is represented by laying out the cells on a table grid consisting of three table grid columns, each grid  
 18   column representing a logical vertical column in the table:


19

20   The dashed lines represent the virtual vertical continuations of each table grid column, and the resulting table  
 21   grid is represented as the following in WordprocessingML:

```
22     <w:tblGrid>
23       <w:gridCol w:w="2952"/>
24       <w:gridCol w:w="4416"/>
25       <w:gridCol w:w="1488"/>
26     </w:tblGrid>
```

```

1   <w:tr>
2     <w:tc>
3       <w:tcPr>
4         <w:tcW w:w="7368" w:type="dxa"/>
5         <w:gridSpan w:val="2"/>
6       </w:tcPr>
7       <w:p/>
8     </w:tc>
9     <w:tc>
10    <w:tcPr>
11      <w:tcW w:w="1488" w:type="dxa"/>
12      </w:tcPr>
13      <w:p/>
14    </w:tc>
15  </w:tr>
16  <w:tr>
17    <w:tc>
18      <w:tcPr>
19        <w:tcW w:w="2952" w:type="dxa"/>
20        </w:tcPr>
21        <w:p/>
22      </w:tc>
23      <w:tc>
24        <w:tcPr>
25          <w:tcW w:w="5904" w:type="dxa"/>
26          <w:gridSpan w:val="2"/>
27        </w:tcPr>
28        <w:p/>
29      </w:tc>
30    </w:tr>

```

31 Notice that each of the cells which do not span one grid column (i.e., span two adjacent vertical lines) must  
32 specify this fact by supplying a gridSpan element with a value which determines how many grid columns this  
33 cell will span. Each gridCol element represents a shared 'column' in a table (to which the cells will snap) even  
34 if it doesn't appear visually.

### 2.5.4 Table Rows and Cells

36 A table row is defined using a tr element, which is analogous to the HTML <tr> tag. The tr element acts as a  
37 container for a row of cells with the table's content.

38 A tr element has one formatting child element, trPr, which defines the row properties (such as the row's  
39 width) and whether it can split across a page. Each property is defined by an individual child element under the  
40 trPr element. The complete set of table row properties can be found on the definition for the trPr element. As

1 well, a table row can contain two types of content: custom markup (custom XML or structured document  
 2 tags), and table cells.  
 3 The cells in a row contain the table's content and are defined by tc elements, which are analogous to HTML  
 4 <td> tags.  
 5 A tc element has one formatting child element, tcPr, which defines the properties for the cell. Each unique  
 6 property is specified by a child element of this element. The complete set of table cell properties can be found  
 7 on the definition for the tcPr element. As well, a table cell can contain any valid block-level content, which  
 8 allows for the nesting of paragraphs and tables within table cells.  
 9 In the example below, the tcW element defines the width of the cell, where the attribute w is the value in  
 10 twips. Here the width of the cell is 8,856 units, where *units* are defined by the attribute type. In this case, dxa  
 11 represents twips.

```
12   <w:tr>
13     <w:tc>
14       <w:tcPr>
15         <w:tcW w:w="8856" w:type="dxa"/>
16       </w:tcPr>
17       <w:p/>
18     </w:tc>
19   </w:tr>
```

20 The tc element contains the cell's content, which, in this case, is an empty p element.  
 21 Consider a table having one cell, which contains the text "Hello, world":

Hello, world
--------------

22  
 23 This table's content is represented by the following XML:

```
24   <w:tr>
25     <w:tc>
26       <w:tcPr>
27         <w:tcW w:w="1770" w:type="dxa"/>
28       </w:tcPr>
29       <w:p>
30         <w:r>
31           <w:t>Hello, World</w:t>
32         </w:r>
33       </w:p>
34     </w:tc>
35   </w:tr>
```

- 1 At both the row and cell levels, the properties must also specify how the rows and cells will be placed on the  
 2 table grid.
- 3 The trPr element contains information about the number of grid units which should be omitted ('skipped')  
 4 before and after the row is complete using the gridBefore and gridAfter elements, allowing rows to start at  
 5 different columns on the grid, as well as a preferred width for that leading/trailing space using the wBefore  
 6 and wAfter elements. The tcPr element also contains grid information pertaining to how many grids a cell  
 7 spans using the gridSpan element, which determines how many grid units are consumed by the current cell,  
 8 as well as a preferred width for that cell using the tcW element.
- 9 In the earlier complex table having two rows of two differently sized cells, a consumer shall represent that  
 10 table containing three grid columns (one per distinct vertical line). Consider the following XML for the first row  
 11 of that table:

```

12   <w:tr>
13     ...
14     <w:tc>
15       <w:tcPr>
16         <w:tcW w:w="5145" w:type="dxa" />
17         <w:gridSpan w:val="2" />
18       </w:tcPr>
19       <w:p />
20     </w:tc>
21     <w:tc>
22       <w:tcPr>
23         <w:tcW w:w="2145" w:type="dxa" />
24       </w:tcPr>
25       <w:p />
26     </w:tc>
27   </w:tr>
```

- 28 Again, the gridSpan element is the number of grid columns that cell spans when being laid out on the table  
 29 grid. In this example, the first cell of the first row contains two grid columns. As well, the cell specifies its  
 30 preferred width using the tcW element, which tells the consumer the width desired by that cell at layout time.
- 31 It is important to note that every width in a table is a preferred width - because the table must satisfy the grid  
 32 at all times, conflicting table properties must be resolved by overriding preferred widths in a specific manner,  
 33 shown below.

### 34 2.5.5 Table Layout

- 35 Given the information shown in the table shown above, the table is specified as a series of properties:

- 36 • Table-level properties (e.g., preferred width)  
 37 • Table column grid

- 1     • Row-level properties (e.g., grid units before/after row start/end)  
 2     • Cell-level properties (e.g., number of grid units spanned)

3   In order to manipulate this set of properties into a table, the following logics are used, depending on the type  
 4   of table:

## 5   **2.5.6      Fixed Width Tables**

6   The first type of table is a fixed width table, a table that does not dynamically resize based on its contents. In a  
 7   fixed width table, the table information is used in the following manner:

- 8     • The table grid is used to create the set of shared columns in the table and their initial widths as defined  
 9       in the `tblGrid` element  
 10     • The table's total width is defined based on the `tblW` property – if it is set to `auto` or `nil`, then the  
   width is not yet determined and will be specified using the row and cell information.  
 12     • The first table row is read and the initial number of grid units before the row starts is skipped. The  
   width of the skipped grid columns is set using the `wBefore` property.  
 14     • The first cell is placed on the grid, and the width of the specified grid column span set by `gridSpan` is  
   set based on the `tcW` property.  
 16     • Each additional cell is placed on the grid.  
 17     • If at any stage, the preferred width requested for the cells exceeds the preferred width of the table,  
   then each grid column is proportionally reduced in size to fit the table width.  
 19     • If the grid is exceeded (e.g., `tblGrid` specifies three grid columns, but the second cell has a `gridSpan` of  
   three), the grid is dynamically increased with a default width for the new grid column.  
 21     • For each subsequent row, cells are placed on the grid, and each grid column is adjusted to be the  
   maximum value of the requested widths (if the widths do not agree) by adding width to the last cell  
   that ends with that grid column. Again, if at any point, the space requested for the cells exceeds the  
   width of the table, then each grid column is proportionally reduced in size to fit the table width.

## 25   **2.5.7      AutoFit Tables**

26   In an AutoFit table (one which specifies that it should “AutoFit to table contents”), the table information is  
 27   used in the following manner:

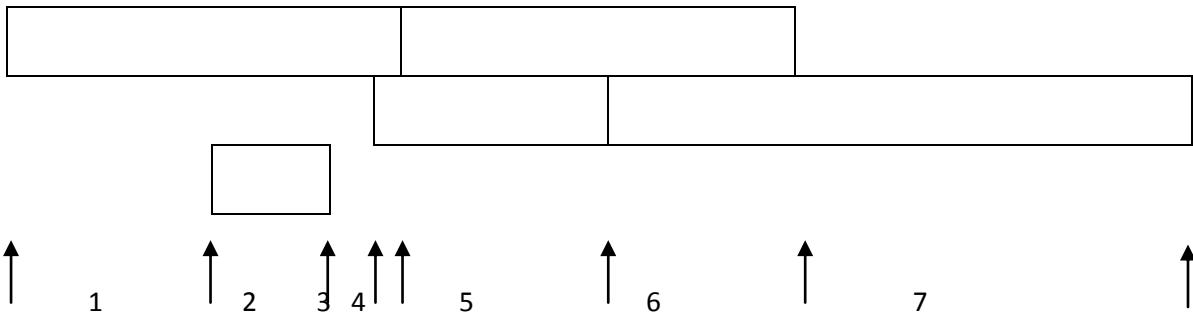
- 28     • Perform the steps above to lay out the fixed width version of the table.  
 29     • Calculate the minimum content width - the width of the cell's contents including all possible line  
   breaking locations (or the cell's width, if the width of the content is smaller), and the maximum  
   content width -the width of the cell's contents (assuming no line breaking not generated by explicit  
   line breaks).  
 31     • The minimum and maximum content width of all cells that span a single grid column is the minimum  
   and maximum content width of that column.  
 34     • For cells which span multiple grid columns, enlarge all cells which it spans as needed to meet that cell's  
   minimum width.

- If any cell in a grid column has a preferred width, the first such width overrides the maximum width of the column's contents.
- Place the text in the cells in the table, respecting the minimum content width of each cell's content. If a cell's minimum content width exceeds the cell's current width, preferences are overridden as follows:
  - First, override the column widths by making all other grid columns proportionally smaller until each it at its minimum width. This cell may then grow to any width between its own minimum and maximum width.
  - Next, override the preferred table width until the table reaches the page width.
  - Finally, force a line break in each cell's contents as needed

### 2.5.8 Complex Table Example

The properties above are best illustrated by example:

As shown above, table cells can be merged horizontally. This is represented with a single table cell whose gridSpan property defines the number of grid units consumed by that table cell for the current row. Consider the following fixed width table, which makes extensive use of resized and merged cells on what is actually just a seven-column grid. (The arrows point to each (invisible) vertical line of the grid and the numbers refer to the grid columns):



Although the table is visually complex, the standard rules apply: the first cell in the table is simply a cell which spans four grid units horizontally, as specified in the gridSpan element, and whose preferred width is 2952 twips, specified in the tcW element:

```

<w:tc>
  <w:tcPr>
    <w:tcW w:w="2952" w:type="dxa"/>
    <w:gridSpan w:val="4"/>
  </w:tcPr>
  <w:p/>
</w:tc>

```

Similarly, all cells indented from the start and end of the grid specify that indent using the gridBefore and gridAfter elements. For example, the XML for the second row in the table shows that that row starts three grid units into the table:

```

1   <w:tr>
2     <w:trPr>
3       <w:gridBefore w:val="3"/>
4       <w:wBefore w:w="2748" w:type="dxa"/>
5     </w:trPr>
6     ...
7   </w:tr>

```

8 If we take this fixed width table and introduce a long string into the single cell in row 3, we see that the  
9 presence of this text does not affect cell widths:

longtextstringwithnobreakingcharacters	

10

11 If we now turn on the AutoFit property and type into the cell in row three, which spans only grid column two,  
12 we see that the algorithm for this AutoFit table causes all cells in grid column two to increase in size,  
13 proportionally decreasing the other grid columns' size to accommodate the long non-breaking string in the last  
14 cell:

longtextstringwithnobreakingcharacters	

15

16 Each of the other grid columns was reduced, but since all columns are not at their minimum size, the table  
17 width is not increased even though the table is not yet at the page width.

### 18 2.5.9 Vertically Merged Cells

19 Although the previous examples may have implied that tables have strict definition of rows, table cells can also  
20 be merged vertically. The tcPr element may contain the vmerge element that defines the extent of vertically  
21 merged grid columns within a table. A vmerge element with its val attribute set to restart marks the start of  
22 a vertically merged cell range. A vmerge element with the val attribute set to continue (the default value)

1 marks the continuation of a vertically merged grid column. Cells between the first and last merged cell that are  
 2 part of the vertical merge each must have a vmerge element to continue the vertical merge.

3 For example, consider a table with two rows and two columns:

First cell, first row	Last cell, first row
First cell, second row	Last cell, second row

4

5 Merging the two rows in the second column will result in the following table:

First cell, first row	Last cell, first row
First cell, second row	Last cell, second row

6

7 The last cell in the first row starts a merge that is completed in the cell below it, resulting in the following  
 8 WordprocessingML:

```

9      <w:tr>
10         <w:tc>
11             <w:p>
12                 <w:r>
13                     <w:t>First cell, first row</w:t>
14                 </w:r>
15             </w:p>
16         </w:tc>
17         <w:tc>
18             <w:tcPr>
19                 <w:vmerge w:val="restart"/>
20             </w:tcPr>
21             <w:p>
22                 <w:r>
23                     <w:t>Last cell, first row</w:t>
24                 </w:r>
25             </w:p>
26             <w:p>
27                 <w:r>
28                     <w:t>Last cell, second row</w:t>
29                 </w:r>
30             </w:p>
31         </w:tc>
32     </w:tr>
```

```

1   <w:tr>
2     <w:tc>
3       <w:p>
4         <w:r>
5           <w:t>First cell, second row</w:t>
6         </w:r>
7       </w:p>
8     </w:tc>
9     <w:tc>
10       <w:tcPr>
11         <w:vmerge/>
12       </w:tcPr>
13       <w:p/>
14     </w:tc>
15   </w:tr>
```

16 As shown, the vmerge with a value of restart begins (or restarts) a merged region, and the cell with no  
 17 value is merged with the one above.

## 18 2.6 Custom Markup

19 Within a WordprocessingML document, it is often necessary for specific documents to contain semantic  
 20 information beyond the presentation information specified by this Office Open XML specification. For example,  
 21 an invoice document may wish to specify that a particular sentence of text is a customer name, in order for  
 22 that information to be easily extracted from the document without the need to parse the text using regular  
 23 expression matching or similar. For those cases, multiple facilities are provided for the insertion and round-  
 24 tripping of customer defined semantics within a WordprocessingML document.

25 There are three distinct forms in which customer-defined semantics can be inserted into a WordprocessingML  
 26 document, each with their own specific intended usage:

- 27 • Smart tags
- 28 • Custom XML markup
- 29 • Structured document tags (content controls)

30 The usage and presentation of each of these forms is described in the following sections.

### 31 2.6.1 Smart Tags

32 The first example of customer-defined semantics which can be embedded in a WordprocessingML document  
 33 are smart tags. Smart tags allow semantic information to be added around an arbitrary run or set of runs  
 34 within a document to provide information about the type of data contained within.

35 Consider the following text in a WordprocessingML document, with a smart tag around the stock symbol  
 36 'CNTS' (where the smart tag is displayed using a purple dotted underline):

This is a stock symbol: MSFT

- 1  
2 This text would translate to the following WordprocessingML markup:

```
3 <w:p w:rsidR="00672474" w:rsidRDefault="00672474">
4   <w:r>
5     <w:t xml:space="preserve">This is a stock symbol: </w:t>
6   </w:r>
7   <w:smartTag w:uri="http://schemas.openxmlformats.org/2006/smarttags"
8     w:element="stockticker">
9     <w:r>
10    <w:t>MSFT</w:t>
11    </w:r>
12  </w:smartTag>
13 </w:p>
```

14 As shown above, the smart tag is delimited by the smartTag element, which surrounds the run (or runs) which  
15 contain the text which is part of the smart tag.

16 The smart tag itself carries two required pieces of information, which together contain the customer semantics  
17 for this smart tag.

18 The first of these is the namespace for this smart tag (contained in the uri attribute). This allows the smart tag  
19 to specify a URI which should be round-tripped with this smart tag and be available to a consumer. It is  
20 intended to be used to specify a family of smart tags to which this one belongs – for example, in the sample  
21 above, the smart tag belongs to the <http://schemas.openxmlformats.org/2006/smarttags> namespace.

22 The second of these is the element name for this smart tag (contained in the element attribute). This allows  
23 the smart tag to specify a name which should be round-tripped with this smart tag and again available to a  
24 consumer. It is intended to be used to specify a unique name for this type of smart tag – for example, in the  
25 sample above, the smart tag specifies that its data is of type stockticker.

26 As well as the required information specified above, a smart tag can also contain any number of additional  
27 properties in namespace/name/value sets by adding them to the smart tag's property bag.

28 Using the example above, adding a new property called **fullCompanyName** with no namespace and value  
29 Microsoft Corporation to the smart tag would mean augmenting the output to add the smartTagPr  
30 element with this new property as follows:

```

1   <w:smartTag w:uri="http://schemas.openxmlformats.org/2006/smarttags"
2     w:element="stockticker">
3     <w:smartTagPr>
4       <w:attr w:name="fullCompanyName" w:val="Microsoft Corporation"/>
5     </w:smartTagPr>
6     <w:r>
7       <w:t>MSFT</w:t>
8     </w:r>
9   </w:smartTag>
```

- 10 The resulting XML, as seen above, simply adds an attr element which specifies the property and value for the  
 11 property bag.
- 12 A producer can embed a smart tag around any run-level content in a WordprocessingML document in order to  
 13 embed additional information about the family and type of the data contained within. This allows ‘tagging’ of  
 14 specific regions of a document with these semantics without need to provide context beyond the information  
 15 provided in the uri and element attributes.
- 16 A consumer can read this smart tag data and provide additional functionality around these  
 17 namespace/element pairs, which may or may not be specific to that smart tag type in the document. Examples  
 18 of this functionality include: the ability to add/remove this markup via a user interface, ability to provide  
 19 actions to operating in the context of this data type, etc.

## 20      **2.6.2      Custom XML Markup**

- 21 The next example of customer-defined semantics which can be embedded in a WordprocessingML document  
 22 is custom XML markup. Custom XML markup allows the application of the XML elements defined in any valid  
 23 XML Schema file to be applied to the contents of a WordprocessingML document in one of two locations:  
 24 around a paragraph or set of paragraphs (at the block level); or around an arbitrary run or set of runs within a  
 25 document (at the inline level) to provide semantics to that content within the context and structures defined  
 26 by the associated XML Schema definition file.
- 27 The distinction between custom XML markup and smart tags is based on the fact that custom XML markup  
 28 corresponds with the contents of a custom XML schema; which means that as shown below, custom XML  
 29 markup can be used at the block-level to mark up the contents of a document on levels beyond that of one or  
 30 more runs as well as on the inline (run) level. It can also be validated against a custom XML schema by a  
 31 producer at run time.
- 32 Consider a simple XML Schema which defines two elements: a root element of invoice, and a child element of  
 33 customerName - the first defining that this file's contents are an invoice, and the second specifying that the  
 34 enclosed text as a customer's name:

`<invoice>`This is an invoice.

And this is a customer name: `<customerName>(Tristan Davis)</customerName>`

- 1
- 2 This output would translate to the following WordprocessingML markup:

```

3   <w:customXml w:uri="http://www.contoso.com/2006/invoice" w:element="invoice">
4     <w:p>
5       <w:r>
6         <w:t>This is an invoice.</w:t>
7       </w:r>
8     </w:p>
9     <w:p>
10    <w:r>
11      <w:t xml:space="preserve">And this is a customer name: </w:t>
12    </w:r>
13    <w:customXml w:uri="http://www.contoso.com/2006/invoice"
14      w:element="customerName">
15      <w:r>
16        <w:t>Tristan Davis</w:t>
17      </w:r>
18      </w:customXml>
19    </w:p>
20  </w:customXml>

```

- 21 As shown above, each of the XML elements from the customer-supplied XML schema is represented within the document output as a customXml element.

23 Similar to the smart tag example above, a custom XML element in a document has two required attributes.

24 The first is the uri attribute, whose contents specify the namespace of the custom XML element in the document. In the example above, the elements each belong to the `http://www.contoso.com/2006/invoice` namespace.

27 The second is the element attribute, whose contents specify the name of the custom XML element at this location in the document. In the example above, the root element is called invoice and the child element is called customerName.

30 As well as the required information specified above, custom XML elements can also specify any number of attributes (as specified in the associated XML Schema) on the element. To add this information, the customXmlPr (properties on the custom XML element) specify one or more attr elements.

33 Using the example above, we can add a type attribute to the customerName element as follows:

```

1   <w:customXml w:uri="http://www.contoso.com/2006/invoice"
2     w:element="customerName">
3       <w:customXmlPr>
4         <w:attr w:uri="http://www.contoso.com/2006/invoice" w:name="type"
5           w:val="individual"/>
6         </w:customXmlPr>
7       <w:r>
8         <w:t>Tristan Davis</w:t>
9       </w:r>
10      </w:customXml>
```

11 The resulting XML, as seen above, simply adds an attr element which specifies the attribute for the custom  
12 XML element.

13 A producer can embed a custom XML element around or with block-level or run-level content in a  
14 WordprocessingML document in order to embed the structure of the customer-defined XML Schema within  
15 the WordprocessingML content. This allows ‘tagging’ of specific regions of a document with the semantics  
16 from this schema, while ensuring that the resulting file can be validated to the WordprocessingML schemas.

17 A consumer can read this custom XML markup and provide additional functionality around this customer-  
18 defined XML markup, which may or may not be specific to that type of XML in the document. Examples of this  
19 functionality include: the ability to add/remove this XML markup via a user interface, ability to provide actions  
20 to operating in the context of this data type, etc.

21 Each custom XML element is analogous to an XML element in the specified XML schema, and can be nested  
22 arbitrarily to any depth in the document. This facility is limited only by the XML Schema file itself, and the  
23 contents of the current document.

### 24 **2.6.3 Structured Document Tags**

25 The final example of customer-defined semantics which can be embedded in a WordprocessingML document  
26 is the structured document tag (SDT).

27 As shown above, smart tags and custom XML markup each provide a facility for embedding customer-defined  
28 semantics into the document: smart tags, via the ability to provide a basic namespace/name for a run or set of  
29 runs within a documents; and custom XML markup, via the ability to tag the document with XML elements and  
30 attributes specified by any valid XML Schema file.

31 However, each of these techniques, while they each provide a way to add the desired semantic information,  
32 does not provide a way to affect the presentation or interaction within the document. To bridge these two  
33 worlds, structured document tags allow both the specification of customer semantics as well as the ability to  
34 influence the presentation of that data in the document.

- 1 This means that the customer can define the semantics and context of the tag, but can then use a rich set of  
 2 pre-defined properties to define its behavior and appearance within the WordprocessingML document's  
 3 presentation.
- 4 Consider a region which should be tagged with the semantic of "birthday", for the user to enter their date or  
 5 birth into the document. Ideally, this region would also utilize a date picker to allow the user to enter the date  
 6 from a calendar:



- 7
- 8 This content would translate to the following WordprocessingML markup:

```

9   <w:sdt>
10  <w:sdtPr>
11    <w:alias w:val="Birthday"/>
12    <w:id w:val="8775518"/>
13    <w:placeholder>
14      <w:docPart w:val="DefaultPlaceholder_22479095"/>
15    </w:placeholder>
16    <w:showingPlcHdr/>
17    <w:date>
18      <w:dateFormat w:val="M/d/yyyy"/>
19      <w:lid w:val="EN-US"/>
20    </w:date>
21  </w:sdtPr>
22  <w:sdtContent>
23    <w:p>
24      <w:r>
25        <w:rPr>
26          <w:rStyle w:val="PlaceholderText"/>
27        </w:rPr>
28        <w:t>Click here to enter a date...</w:t>
29      </w:r>
30    </w:p>
31  </w:sdtContent>
32 </w:sdt>
```

- 33 As shown above, each of the structured document tags in the WordprocessingML file is represented using the  
 34 sdt element.

1 Within a structured document tag, there are two child elements which contain the definition and the content  
 2 of this SDT. The first of these is the sdtPr element, which contains the set of properties specified for this  
 3 structured document tag. The second is the sdtContent element, which contains all the content which is  
 4 contained within this structured document tag.

### 5 2.6.3.1 Structured Document Tag Properties

6 Within the SDT's properties, various properties can be set which affect the appearance and behavior of this  
 7 content in the document. These properties can be divided into four groups:

- 8 • Shared properties
- 9 • Locking properties
- 10 • Structured document tag type
- 11 • Type-specific properties

12 The complete set of properties for a structured document tag are found on the sdtPr element.

13 The first group is properties shared by all types of SDTs. These include, but are not limited to, the semantic  
 14 name for the SDT, a unique ID (as an integer) that is round-tripped and allows the control to be uniquely  
 15 identified across sessions, and a reference to a document building block that should be displayed as  
 16 placeholder text.

17 The next group is the locking properties for the tag – these specify whether any consumer should allow the  
 18 contents of the SDT to be edited, or the SDT itself to be deleted from the document.

19 The next group, the structured document tag's type, specifies how the content should be expressed in a  
 20 document. These types include: plain text (all contents are of one formatting), rich text, date picker, combo  
 21 box, drop-down list, and image. Each of the types provides user interface restrictions that restrict the contents  
 22 to only those specified by the type (e.g., the picture cannot contain text).

23 Finally, the type-specific properties contain properties that are sensible in the context of that type. For  
 24 example, the date format for a date picker or the drop-down list entries for a drop-down list/combo box. Type-  
 25 specific properties are stored as children of the type's element.

26 Referring to the example above, the date properties are stored underneath the date element, as follows:

```
27 <w:sdtPr>
28 ...
29 <w:date>
30   <w:dateFormat w:val="M/d/yyyy"/>
31   <w:lid w:val="EN-US"/>
32 </w:date>
33 </w:sdtPr>
```

34 This ensures that these properties are only available in the appropriate context(s).

1    2.6.3.2        Structured Document Tag Content

2    The second child of the sdt element is the sdtContent element, which contains all the content which is  
 3    contained within this structured document tag.

4    2.6.3.3        XML Mapping

5    An additional property for SDTs allows their contents to be stored in another part (in particular, in the custom  
 6    XML data storage within the file). The presence of the dataBinding element specifies that the contents of this  
 7    SDT are simply a cache of the data stored at a particular XML element in a particular custom XML data storage  
 8    part.

9    2.7        Sections

10   Within the main document story, there is also often a need for groupings of content on a basis larger than a  
 11   paragraph (for example, ensuring that a specific set of paragraphs and tables are printed in landscape view,  
 12   while ensuring that the remainder of the document is printed in portrait view). In order to group this content,  
 13   a document can be divided into multiple *sections*, each of which defines a region of content in the document  
 14   and allows the application of a set of section-level properties.

15   Consider a WordprocessingML document with two paragraphs of content, the first of which should be  
 16   displayed on a page printed in portrait view, and the second of which should be displayed on a page printed in  
 17   landscape view (the page content should be rotated 90 degrees to the left on the underlying page).

18   In order to have each of these paragraphs on different pages having different page orientation characteristics,  
 19   this document would be split into two sections. Looking at the WordprocessingML for the example above:

```

20      <w:body>
21          <w:p>
22              <w:pPr>
23                  <w:sectPr>
24                      <w:pgSz w:w="12240" w:h="15840"/>
25                      <w:pgMar w:top="1440" w:right="1800" w:bottom="1440"
26                          w:left="1800" w:header="720" w:footer="720" w:gutter="0"/>
27                      <w:cols w:space="720"/>
28                      <w:docGrid w:linePitch="360"/>
29                  </w:sectPr>
30              </w:pPr>
31              <w:r>
32                  <w:t>This is sentence one.</w:t>
33              </w:r>
34          </w:p>
```

```

1      <w:p>
2          <w:r>
3              <w:t>This is sentence two.</w:t>
4          </w:r>
5      </w:p>
6      <w:sectPr>
7          <w:pgSz w:w="15840" w:h="12240" w:orient="landscape"/>
8          <w:pgMar w:top="1800" w:right="1440" w:bottom="1800"
9              w:left="1440" w:header="720" w:footer="720" w:gutter="0"/>
10         <w:cols w:space="720"/>
11         <w:docGrid w:linePitch="360"/>
12     </w:sectPr>
13 </w:body>

```

14 This syntax defines two sections using two distinct sectPr elements: the first has a page size of 12,240  
15 twentieths of a point wide and 15,640 twentieths of a point tall; the second has a page size of 15,640  
16 twentieths of a point wide and 12,240 twentieths of a point tall, and is oriented in landscape mode.

## 17 2.7.1 Section Properties

18 As shown above, the end of a section is defined as a set of properties applied to the last paragraph in that  
19 section—converting that paragraph mark into a *section break* (i.e., a paragraph that closes a section).

20 Those properties are contained in a sectPr element, which is located within the paragraph properties (the  
21 pPr element) for the final paragraph in that section. Within the definition of section properties, the properties  
22 to be applied to that section (including, but not limited to, page size and orientation, line numbering settings,  
23 margins, and columns) are specified. The complete set of section properties is located on the definition for the  
24 sectPr element.

25 The only exception to this rule is the final set of section properties in this document. These are stored as the  
26 last child of the body element. This is done because the document's last paragraph must specify paragraph  
27 properties, and this syntax enforces that the final set of section properties are specified.

28 Going back to our example, the first section break is defined within the last paragraph for that section, but the  
29 last section properties are stored after the final paragraph.

```

1   <w:body>
2     <w:p
3       <w:pPr>
4         <w:sectPr>
5           <w:pgSz w:w="12240" w:h="15840"/>
6           <w:pgMar w:top="1440" w:right="1800" w:bottom="1440"
7             w:left="1800" w:header="720" w:footer="720" w:gutter="0"/>
8           <w:cols w:space="720"/>
9           <w:docGrid w:linePitch="360"/>
10        </w:sectPr>
11      </w:pPr>
12      <w:r>
13        <w:t>This is sentence one.</w:t>
14      </w:r>
15    </w:p>
16    <w:p>
17      <w:r>
18        <w:t>This is sentence two.</w:t>
19      </w:r>
20    </w:p>
21    <w:sectPr>
22      <w:pgSz w:w="15840" w:h="12240" w:orient="landscape"/>
23      <w:pgMar w:top="1800" w:right="1440" w:bottom="1800"
24        w:left="1440" w:header="720" w:footer="720" w:gutter="0"/>
25      <w:cols w:space="720"/>
26      <w:docGrid w:linePitch="360"/>
27    </w:sectPr>
28  </w:body>

```

## 29     2.7.2     Section Breaks

30     As well as specifying the section's properties, the type of section break is specified using the type element.  
31     WordprocessingML supports four distinct types of section breaks:

- 32       • *Next page section breaks* (the default if type is not specified), which begin the new section on the  
33        following page.
- 34       • *Odd page section breaks*, which begin the new section on the next odd-numbered page.
- 35       • *Even page section breaks*, which begin the new section on the next even-numbered page.
- 36       • *Continuous section breaks*, which begin the new section on the following paragraph. This means that  
37        continuous section breaks might not specify certain page-level section properties, since they must be  
38        inherited from the following section. These breaks, however, can specify other section properties, such  
39        as line numbering and footnote/endnote settings.

## 1    2.8    Styles

- 2    After looking at the primary elements of block-level content in a WordprocessingML file, it is now necessary to  
3    look at the information stored in the document that affects how this content is displayed.
- 4    The first such group of information is *styles*. Within a WordprocessingML file, *styles* are predefined sets of  
5    paragraph and/or character properties which can be applied to text within the document. This allows the  
6    formatting properties to be stored and managed independently from the content, allowing the look of  
7    document content to be changed in a single location (e.g., the look of all first-level headings is changed by  
8    changing the style with styleId `Heading1` rather than looking for and changing each paragraph in the  
9    document).
- 10   The Normal paragraph style in a word processing document can have any number of formatting properties,  
11   e.g., font face = Times New Roman; font size = 12pt; paragraph justification = left). All paragraphs that  
12   reference this paragraph style would automatically inherit these properties.

### 13    2.8.1    Styles Part

- 14   Style information in a WordprocessingML document is stored in the Styles part within the package, which is  
15   stored via an implicit relationship from the Main Document or Glossary Document part of relationship type  
16   <http://schemas.openxmlformats.org/wordprocessingml/2006/styles> and has a content type of  
17   `vnd-openxmlformats.officedocument.wordprocessingml-styles+xml`.
- 18   The styles part stores two types of style information for the document:
- 19        •    Style definitions  
20        •    Latent style information

### 21    2.8.2    Style Definitions

- 22   Each style defined within a WordprocessingML document requires a *style definition*. The style definition  
23   contains all of the information needed by a consumer to store and display that style within a  
24   WordprocessingML document, and is defined using the style element. The style definition for any style in  
25   WordprocessingML can be divided into three segments (Note: the complete definition of style properties can  
26   be found on the reference for the style element):
- 27        •    Common style properties  
28        •    Style ‘types’  
29        •    Type specific properties
- 30   Common style properties refer to the set of properties which can be used regardless of the type of style; for  
31   example, the style name, additional aliases for the style, a style ID (used by the document content to refer to  
32   the style), if style is hidden, if style is locked, etc.
- 33   Consider a style called Heading 1 in a document as follows:

```

1   <w:style w:type="paragraph" w:styleId="Heading1">
2     <w:name w:val="heading 1"/>
3     <w:basedOn w:val="Normal"/>
4     <w:next w:val="Normal"/>
5     <w:link w:val="Heading1Char"/>
6     <w:priority w:val="1"/>
7     <w:qformat/>
8     <w:rsid w:val="00F303CE"/>
9     ...
10    </w:style>
```

11 Above the formatting information specific to this style type are a set of common style properties which define  
12 information shared by all style types.

13 Style types refer to the property on a style that defines the type of style created with this style definition.  
14 WordprocessingML supports six types of style definitions:

- 15 • Paragraph styles
- 16 • Character styles
- 17 • Linked styles (paragraph + character)
- 18 • Table styles
- 19 • Numbering styles
- 20 • Default paragraph + character properties

21 Consider a style called Heading 1 in a document as follows:

```

22   <w:style w:type="paragraph" w:styleId="Heading1">
23     <w:name w:val="heading 1"/>
24     <w:basedOn w:val="Normal"/>
25     <w:next w:val="Normal"/>
26     <w:link w:val="Heading1Char"/>
27     <w:priority w:val="1"/>
28     <w:qformat/>
29     <w:rsid w:val="00F303CE"/>
30     ...
31   </w:style>
```

32 The type attribute has a value of *paragraph*, which indicates that the following style definition is a paragraph  
33 style.

### 34 **2.8.3 Paragraph Styles**

35 The first type of style definition, *paragraph styles* are styles that apply to the contents of an entire paragraph  
36 as well as the paragraph mark. This definition implies that the style can define both character properties  
37 (properties that apply to text within the document) as well as paragraph properties (properties which apply to

1 the positioning and appearance of the paragraph). Paragraph styles cannot be referenced by runs within a  
 2 document, they must be referenced by the pStyle element within a paragraph's paragraph properties (pPr)  
 3 element.

4 A paragraph style has three defining type-specific characteristics:

- 5 • The type attribute on the style has a value of paragraph, which indicates that the following style  
 6 definition is a paragraph style.
- 7 • The next element defines an editing behavior which supplies the paragraph style to be automatically  
 8 applied to the next paragraph when ENTER is pressed at the end of a paragraph of this style.
- 9 • The style specifies both paragraph-level and character-level properties using the pPr and rPr  
 10 elements, respectively. In this case, the run properties are the set of properties applied to each run in  
 11 the paragraph.

12 The paragraph style is then applied to paragraphs by referencing the styleId attribute value for this style in the  
 13 paragraph properties' pStyle element.

14 Consider a paragraph style titled "Test Paragraph Style" which defines: paragraph spacing = double, paragraph  
 15 indent = 1" (first line only); font = Algerian, font color = red, font size = 20 points. The resulting style definition  
 16 would be:

```
17 <w:style w:type="paragraph" w:styleId="TestParagraphStyle">
18   <w:name w:val="Test Paragraph Style"/>
19   <w:qformat/>
20   <w:rsid w:val="00F85845"/>
21   <w:pPr>
22     <w:spacing w:line="480" w:lineRule="auto"/>
23     <w:ind w:firstLine="1440"/>
24   </w:pPr>
25   <w:rPr>
26     <w:rFonts w:ascii="Algerian" w:hAnsi="Algerian"/>
27     <w:color w:val="ED1C24"/>
28     <w:sz w:val="40"/>
29   </w:rPr>
30 </w:style>
```

31 Notice that the character properties for the style are under the rPr element, and the paragraph properties are  
 32 under the pPr element.

33 The document content for a paragraph of this style would be:

```

1   <w:p>
2     <w:pPr>
3       <w:pStyle w:val="TestParagraphStyle"/>
4     </w:pPr>
5     <w:r>
6       <w:t xml:space="preserve">Here is some fancy Text</w:t>
7     </w:r>
8   </w:p>

```

9 The pStyle element links the paragraph with the style definition.

## 10 2.8.4 Character Styles

11 The next type of style definition, *character styles* are styles which apply to the contents of one or more runs of  
 12 text within a document's contents. This definition implies that the style can only define character properties  
 13 (properties which apply to text within a paragraph) because it cannot be applied to paragraphs. Character  
 14 styles can only be referenced by runs within a document, and they must be referenced by the rStyle element  
 15 within a run's run properties element.

16 A character style has two defining type-specific characteristics:

- 17 • The type attribute on the style has a value of character, which indicates that the following style  
 18 definition is a character style.
- 19 • The style specifies only character-level properties using the rPr element. In this case, the run  
 20 properties are the set of properties applied to each run which is of this style.

21 The character style is then applied to runs by referencing the styleId attribute value for this style in the run  
 22 properties' rStyle element.

23 Consider a character style titled "Test Character Style" which defines; font = Courier New, font color = yellow;  
 24 underline. The resulting style definition would be:

```

25   <w:style w:type="character" w:styleId="TestCharacterStyle">
26     <w:name w:val="Test Character Style"/>
27     <w:priority w:val="99"/>
28     <w:qformat/>
29     <w:rsid w:val="00E77BF0"/>
30     <w:rPr>
31       <w:rFonts w:ascii="Courier New" w:hAnsi="Courier New"/>
32       <w:color w:val="FFF200"/>
33       <w:u w:val="single"/>
34     </w:rPr>
35   </w:style>

```

36 Notice that the character properties applied using this style are under the rPr element. The document content  
 37 for a paragraph with a run of this style would be:

```

1   <w:p>
2     <w:r>
3       <w:t xml:space="preserve">The following text is in the </w:t>
4     </w:r>
5     <w:r>
6       <w:rPr>
7         <w:rStyle w:val="TestCharacterStyle"/>
8       </w:rPr>
9       <w:t>character style</w:t>
10    </w:r>
11    <w:r>
12      <w:t>.</w:t>
13    </w:r>
14  </w:p>

```

15 The rStyle element in the second run links that run with the style definition, inheriting the formatting  
 16 properties for that run.

### 17 2.8.5 Linked Styles

18 The next type of style definition, *linked styles* are actually a paired combination of styles which can be applied  
 19 to the contents of one or more runs of text within a document's contents or the entire contents of one or  
 20 more paragraphs in a WordprocessingML document. This definition implies that the style can define both a set  
 21 of character properties (properties which apply to text within a paragraph) as well as a set of paragraph  
 22 properties (properties which apply to the positioning and appearance of the paragraph) because it must be  
 23 possible to apply the style to paragraphs as well as characters.

24 In order to accomplish these dual uses, a linked style is actually a pairing of a paragraph style and a character  
 25 style in the WordprocessingML document. Each style exists uniquely within the styles part, but are linked by  
 26 the link element, which specifies that these styles are each half of a linked style definition and should be  
 27 treated as one style at runtime.

28 A typical example of the use of a linked style is a quote style - if the style is applied to a paragraph, the quoted  
 29 text should be indented additionally to create a block quote effect, but if the style is applied to text in a  
 30 paragraph, only the character level effects should be applied.

31 Consider the following two styles which comprise a linked style pairing:

```

32  <w:style w:type="paragraph" w:styleId="TestLinkedStyle">
33    <w:name w:val="Test Linked Style"/>
34    <w:link w:val="TestLinkedStyleChar"/>
35    <w:qformat/>
36    <w:rsid w:val="009C1646"/>

```

```

1   <w:pPr>
2     <w:spacing w:line="480" w:lineRule="auto"/>
3     <w:ind w:left="1440"/>
4   </w:pPr>
5   <w:rPr>
6     <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>
7     <w:color w:val="22B14C"/>
8   </w:rPr>
9   </w:style>
10  <w:style w:type="character" w:styleId="TestLinkedStyleChar">
11    <w:name w:val="Test Linked Style Char"/>
12    <w:link w:val="TestLinkedStyle"/>
13    <w:rsid w:val="009C1646"/>
14    <w:rPr>
15      <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>
16      <w:color w:val="22B14C"/>
17    </w:rPr>
18  </w:style>

```

19 The link element in the paragraph style specifies `TestLinkedStyleChar`, the `styleId` of the paired character  
20 style, and the link element in the character style specifies `TestLinkedStyle`, the `styleId` of the paired  
21 paragraph style, creating a linked style combination.

22 Paragraph-level instances of linked styles can only be referenced by paragraphs within a document, and they  
23 must be referenced by the `pStyle` element within the paragraph's paragraph properties element (`pPr`), which  
24 must reference the paragraph version of the linked style. Character-level instances of linked styles can only be  
25 referenced by a run's run properties element (`rPr`) within a document, and they must be referenced by the  
26 `rStyle` element within the run properties element which must reference the character version of the linked  
27 style.

28 Consider a linked style titled "Test Linked Style" which defines; font = Arial, font color = green; paragraph  
29 spacing = double, indent = 1" left. The resulting style definitions would be:

```

30  <w:style w:type="paragraph" w:styleId="TestLinkedStyle">
31    <w:name w:val="Test Linked Style"/>
32    <w:link w:val="TestLinkedStyleChar"/>
33    <w:qformat/>
34    <w:rsid w:val="009C1646"/>
35    <w:pPr>
36      <w:pStyle w:val="TestLinkedStyle"/>
37      <w:spacing w:line="480" w:lineRule="auto"/>
38      <w:ind w:left="1440"/>
39    </w:pPr>

```

```

1   <w:rPr>
2     <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>
3     <w:color w:val="22B14C"/>
4   </w:rPr>
5   </w:style>
6   <w:style w:type="character" w:styleId="TestLinkedStyleChar">
7     <w:name w:val="Test Linked Style Char"/>
8     <w:link w:val="TestLinkedStyle"/>
9     <w:rsid w:val="009C1646"/>
10    <w:rPr>
11      <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>
12      <w:color w:val="22B14C"/>
13    </w:rPr>
14  </w:style>

```

15 Notice that the linked style definition is composed of the paragraph style, which specifies both the run and  
 16 paragraph properties, and the character style, which specifies only the run properties. The document content  
 17 for a paragraph with this linked style would be:

```

18  <w:p>
19    <w:pPr>
20      <w:pStyle w:val="TestLinkedStyle"/>
21    </w:pPr>
22    <w:r>
23      <w:t xml:space="preserve">A para version of Test Linked Style.</w:t>
24    </w:r>
25  </w:p>

```

26 The pStyle element in the paragraph's properties links the paragraph with the paragraph version of the linked  
 27 style definition.

28 The document content for a paragraph with a run of this linked style would be:

```

29  <w:p>
30    <w:r>
31      <w:t xml:space="preserve">Next run is character version of </w:t>
32    </w:r>
33    <w:r>
34      <w:rPr>
35        <w:rStyle w:val="TestLinkedStyleChar"/>
36      </w:rPr>
37      <w:t>Test Linked Style</w:t>
38    </w:r>

```

```

1   <w:r>
2     <w:t>.</w:t>
3   </w:r>
4 </w:p>
```

5 The rStyle element in the second run's properties links the run with the character version of the linked style  
6 definition.

## 7 **2.8.6 Numbering Styles**

8 Numbering styles are style definitions which specify common style properties for a multi-level numbering  
9 format within a document. This means that a numbering style defines only a single paragraph property: a  
10 reference to a numbering definition stored in the document's numbering part, using the numPr element.

11 Unlike paragraph and character styles, numbering styles are never directly referenced by content in the  
12 document – instead, an abstract numbering definition (covered in the numbering topic of this section)  
13 specifies that it is actually the underlying numbering information for a numbering style.

14 Consider a numbering style “Test Numbering Style”:

```

15 <w:style w:type="numbering" w:styleId="TestNumberingStyle">
16   <w:name w:val="Test Numbering Style" />
17   <w:priority w:val="99" />
18   <w:rsid w:val="0045009F" />
19   <w:pPr>
20     <w:numPr>
21       <w:numId w:val="1" />
22     </w:numPr>
23   </w:pPr>
24 </w:style>
```

25 The only information specified in the numbering style definition is a reference to the numbering definition for  
26 the numbering information which is defined by this numbering style.

## 27 **2.8.7 Table Styles**

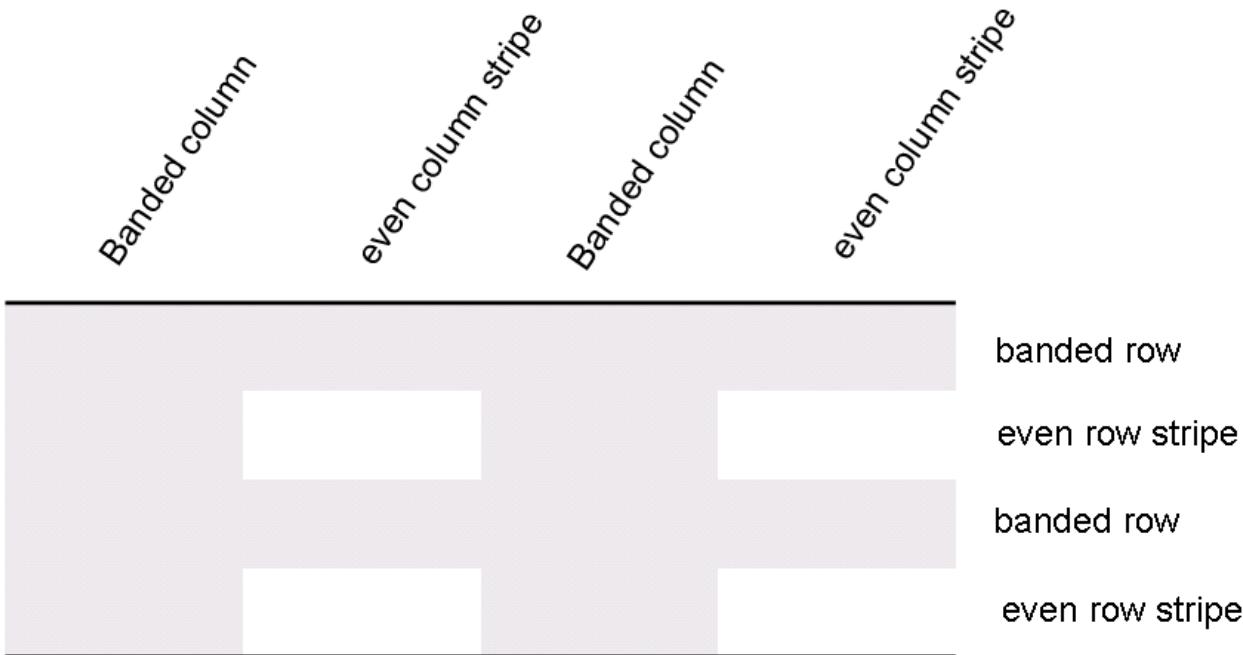
28 The last conventional type of style definition, table styles are styles which apply to the contents of zero or  
29 more tables within a document. This definition implies that the style can only define table properties  
30 (properties which apply to the table and its constituent rows and cells), however a table style can also define  
31 paragraph properties (properties which apply to the positioning and appearance of paragraphs) as well as  
32 character properties (properties which apply to runs) for all of the paragraphs and runs within the specified  
33 table in the document. Table styles can only be referenced by tables within a document, and they must be  
34 referenced by the tblStyle element within a table's table properties (tblPr) element.

- 1 Like the style definitions discussed above, table styles specify all of the properties that can be applied to a table, as well as paragraph and character properties for the table's contents. However, unlike other style definitions, table styles allow for the definition of conditional formats for different regions of the table.
- 4 These table conditional formats are applied to different regions of the table as follows:

Top left cell	Header row	Top right cell
First column	Table body	Last column
Bottom left cell	Footer row	Bottom right cell

5

- 6 All rows in the table can also have conditional formatting on an alternating row/column basis as well as follows:



8

1 When specified, these conditional formats are applied in the following order (therefore subsequent formats  
 2 override properties on previous formats):

- 3 • Whole table
- 4 • Banded columns, even column banding
- 5 • Banded rows, even row banding
- 6 • First row, last row
- 7 • First column, last column
- 8 • Top left, top right, bottom left, bottom right

9 Consider a table style “Test Table Style” defined as follows: all cells with 1pt table borders on all sides, 0.1” cell  
 10 margins on left and right of cells, and 0” cell margins on top and bottom of cells, as well as header row specific  
 11 formatting of: red shading, bold text as follows:

```

12   <w:style w:type="table" w:styleId="TestTableStyle">
13     <w:name w:val="Test Table Style"/>
14     <w:basedOn w:val="TableNormal"/>
15     <w:priority w:val="99"/>
16     <w:rsid w:val="00340CC4"/>
17     <w:tblPr>
18       <w:tblBorders>
19         <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>
20         <w:left w:val="single" w:sz="4" w:space="0" w:color="auto"/>
21         <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>
22         <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>
23         <w:insideH w:val="single" w:sz="4" w:space="0" w:color="auto"/>
24         <w:insideV w:val="single" w:sz="4" w:space="0" w:color="auto"/>
25       </w:tblBorders>
26       <w:tblCellMar>
27         <w:top w:w="0" w:type="dxa"/>
28         <w:left w:w="108" w:type="dxa"/>
29         <w:bottom w:w="0" w:type="dxa"/>
30         <w:right w:w="108" w:type="dxa"/>
31       </w:tblCellMar>
32     </w:tblPr>
33     <w:tblStylePr w:type="firstRow">
34       <w:rPr>
35         <w:b/>
36       </w:rPr>
37       <w:tcPr>
38         <w:shd w:val="clear" w:color="auto" w:fill="ED1C24"/>
39       </w:tcPr>
40     </w:tblStylePr>
41   </w:style>
```

1 The `tblPr` element holds the formatting which is applied to the entire table, and the `tblStylePr` element with a  
 2 type attribute value of `firstRow` holds the formatting for the first table row, specifically the bold run  
 3 property and red cell shading.

4 An individual instance of a table defines an association with a table style using the `tblStyle` element in the  
 5 table's properties (`tblPr`), as discussed above. However, individual tables can choose whether to apply the  
 6 following aspects of the table's conditional formats individually:

- 7 • First row
- 8 • Last row
- 9 • First column
- 10 • Last column
- 11 • Row banding
- 12 • Column banding

13 The use or omission conditional formats are specified using the `tblLook` element, which contains a bitmask  
 14 representing which properties are applied and omitted.

15 Consider two tables using the table style "Style2"; one which specifies that it should only use the header row  
 16 and footer row conditional formatting properties from the table style, and the other which specifies that it  
 17 should use the header row, footer row, and banded row conditional formatting:

```

18 <w:tbl>
19   <w:tblPr>
20     <w:tblStyle w:val="Style2"/>
21     <w:tblW w:w="0" w:type="auto"/>
22     <w:tblLook w:val="0660"/>
23   </w:tblPr>
24   ...
25 </w:tbl>
26 ...
27 <w:tbl>
28   <w:tblPr>
29     <w:tblStyle w:val="Style2"/>
30     <w:tblW w:w="0" w:type="auto"/>
31     <w:tblLook w:val="0460"/>
32   </w:tblPr>
33   ...
34 </w:tbl>
```

35 The tables each specify the appropriate set of conditional formats using the `tblLook` element, as seen by the  
 36 identical table styles in the `tblStyle` element, and different `tblLook` values.

## 1    2.8.8     Default Document Paragraph and Character Properties

2    The final type of style in a WordprocessingML document is the default paragraph and character properties for  
 3    the document. Although this is not a style in the strict sense of the word (because this property set cannot  
 4    directly be applied to text) it defines the basic set of formatting properties which are inherited by paragraphs  
 5    and runs in the document.

6    The following section, entitled Style Inheritance, explains exactly how the default document paragraph and  
 7    character properties influence the appearance of all content in the document.

## 8    2.8.9     Style Inheritance

9    In order to compile the complete set of paragraph and character properties specified by any given style (as  
 10   appropriate), a consumer must follow the rule of style inheritance to determine each property in that set.

11   Style inheritance states that styles of any given type may inherit from other styles of that type, and therefore a  
 12   consumer must ‘build up’ the style information by following the inheritance tree. This inheritance is defined via  
 13   the basedOn element, which specifies the styleId of the parent style.

14   The “Tristan Test” paragraph style can inherit properties from the “Heading 1” paragraph style, which itself can  
 15   inherit properties from the “Normal” paragraph style.

16   To build up the resulting style, a consumer must trace the hierarchy (following each basedOn value) back to a  
 17   style which has no basedOn element (is not based on another style). The resulting style is then constructed by  
 18   following each level in the tree, applying the specified paragraph and/or character properties as appropriate.  
 19   When properties conflict, they are overridden by each subsequent level (this includes turning OFF a property  
 20   set at an earlier level). Properties which are not specified simply do not change those specified at earlier levels.

21   Consider a character style “Green” which specifies only that the text color is green, but inherits from another  
 22   character style “Base” which defines a font face of Arial, as well as bold:

```

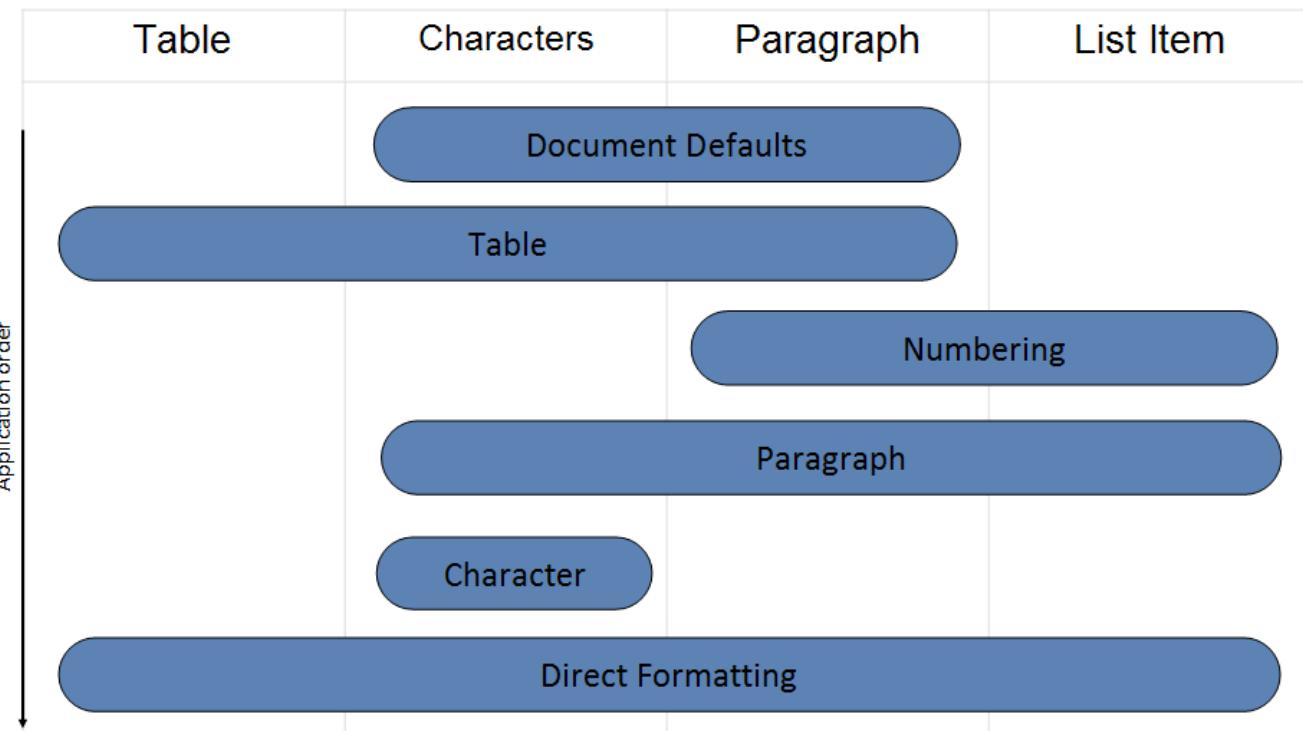
23      <w:style w:type="character" w:styleId="Green">
24          <w:name w:val="Green" />
25          <w:basedOn w:val="Base" />
26          <w:rPr>
27              <w:color w:val="22B14C" />
28          </w:rPr>
29      </w:style>
30      ...
31      ./Local Settings/Temp/styles.xml<w:style w:type="character" w:styleId="Base">
32          <w:name w:val="Base" />
33          <w:rPr>
34              <w:rFonts w:ascii="Arial" w:hAnsi="Arial" />
35              <w:b />
36          </w:rPr>
37      </w:style>
```

- 1 The definition of the Green character style has a basedOn element which specifies the Base style. This means  
2 that any use of the Green style is defined as bold, green, Arial text.
- 3 Conversely, a producer should not output any property on a style which has already been set by a previous  
4 level of the style hierarchy, as well as those which match the document defaults. This means that if the  
5 document defaults or any previous level in a style's hierarchy specify a property which is unchanged at this  
6 level, that property should not be part of the style definition in the resulting WordprocessingML. Adding a  
7 property at multiple levels in the style hierarchy is not invalid, but unnecessarily duplicative as the setting is  
8 already applied to the text, resulting in an unnecessary increase to file size.
- 9 If the document default font is Bauhaus 93 and the Heading 1 style also specifies the Bauhaus 93 font, then  
10 a producer should not output any rFonts element for the Heading 1 style definition, because that formatting  
11 is inherited from the document defaults.

### 12 2.8.10 Style Application

13 With the various flavors of styles available, multiple style types can be applied to the same content within a  
14 file, which means that properties must be applied in a specific deterministic order. As with inheritance, the  
15 resulting formatting properties set by one type can be unchanged, removed, or altered by following types.

16 The following table illustrates the order of application of these defaults, and which properties are impacted by  
17 each:



1 This process can be described as follows: First, the document defaults are applied to all runs and paragraphs in  
 2 the document. Next, the table style properties are applied to each table in the document, following the  
 3 conditional formatting inclusions and exclusions specified per table. Next, numbered item and paragraph  
 4 properties are applied to each paragraph formatted with a numbering style. Next, paragraph and run  
 5 properties are applied to each paragraph as defined by the paragraph style. Next, run properties are applied to  
 6 each run with a specific character style applied. Finally, we apply direct formatting (paragraph or run  
 7 properties not from styles).

### 8 **2.8.11 Latent Styles**

9 The final piece of information stored in the styles part in the document, aside from style definition  
 10 information, is *latent style* information.

11 *Latent styles* are all styles contained in a document's template which have not yet been instantiated (used) in  
 12 the current instance of the document.

13 In WordprocessingML, there are often properties which must be set on all styles in a document template  
 14 regardless of whether they are being used: for example, whether or not the style can be applied in the current  
 15 document (locked state), UI sorting priority, whether the style should be shown in the user interface, etc. In  
 16 order for the document to function correctly, it is essential that this information is stored within the  
 17 document, so that a consumer can determine the necessary style information from the document alone  
 18 (without access to the template). However, it would be grossly inefficient for the document to store all style  
 19 information for all styles simply to store this information, so latent styles are used to store just the necessary  
 20 style properties without caching all style information in the document.

21 In order to do this efficiently, the document declares a `latentStyles` element in the `styles` part which defines  
 22 the default properties applied to all latent styles in the document. All styles whose properties do not match the  
 23 default for the set of style properties which must be defined for all styles are explicitly defined using the  
 24 `lsdException` element.

25 Consider the following latent style information stored in a document's `styles` part:

```
26 <w:latentStyles w:defLockedState="off" w:defPriority="99"
27   w:defSemiHidden="on" w:defUnhideWhenUsed="on" w:defQFormat="off"
28   w:count="180">
29   <w:lsdException w:name="Normal" w:unhideWhenUsed="off"
30     w:qformat="on"/>
31   <w:lsdException w:name="heading 1" w:semiHidden="off" w:priority="1"/>
32   <w:lsdException w:name="heading 2" w:priority="1"
33     w:unhideWhenUsed="on"/>
34   <w:lsdException w:name="heading 3" w:semiHidden="off"/>
35   <w:lsdException w:name="heading 4" w:priority="1" w:qformat="on"/>
36   <w:lsdException w:name="heading 5" w:priority="1" w:qformat="on"/>
```

```

1   <w:lsdException w:name="heading 6" w:priority="1" w:qformat="on"/>
2   <w:lsdException w:name="heading 7" w:priority="1" w:qformat="on"/>
3   <w:lsdException w:name="heading 8" w:priority="1" w:qformat="on"/>
4   <w:lsdException w:name="heading 9" w:priority="1" w:qformat="on"/>
5   <w:lsdException w:name="Normal Indent" w:priority="6" w:qformat="on"/>
6   </w:latentStyles>
```

7 The attributes on the latentStyles element define the properties applied to all latent styles for this document.  
8 All styles whose properties do not match the default latent styles properties are explicitly defined using the  
9 values on the lsdException elements.

## 10 2.9 Fonts

### 11 2.9.1 Font References

12 Within a WordprocessingML document, font face information can be referenced by any set of run properties,  
13 both as part of a style definition or direct formatting on one or more runs in the document's contents. This  
14 reference is established by referencing the primary name of the font face that is used in the rFonts element of  
15 the run properties, linking that run with the desired font face.

16 For example, consider a run of text that has been directly formatted to use the Arial Black font face. This  
17 setting would be specified as follows on the run's properties:

```

18   <w:r>
19     <w:rPr>
20       <w:rFonts w:ascii="Arial Black" w:hAnsi="Arial Black" />
21     </w:rPr>
22     <w:t>This run of text uses the Arial Black font face.</w:t>
23   </w:r>
```

24 The rFonts element specifies that the run should be formatted using the Arial Black font face.  
25 Applications can then look up and use the font with primary name of Arial Black when formatting this run.

### 26 2.9.2 Font Reference Types

27 In the example above, two attributes were present, both referring to the font face with primary name Arial  
28 Black. This simple case illustrates the ability for a WordprocessingML document to store up to four fonts  
29 which may be used on the contents of a run, as follows:

- 30 • ASCII font
- 31 • High ANSI font
- 32 • East Asian font
- 33 • Complex Script font

34 Each of these font faces is used to format the characters in the run that fall under their purview:

- 1 The *ASCII font* formats all characters in the ASCII range (character values 0–127). This font is specified using the  
2 `ascii` attribute on the `rFonts` element.
- 3 The *East Asian font* formats all characters that belong to Unicode sub ranges for East Asian languages. This font  
4 is specified using the `eastAsia` attribute on the `rFonts` element.
- 5 The *complex script font* formats all characters that belong to Unicode sub ranges for complex script languages.  
6 This font is specified using the `cs` attribute on the `rFonts` element.
- 7 The *high ANSI font* formats all characters that belong to Unicode sub ranges other than those explicitly  
8 included by one of the groups above. This font is specified using the `hAnsi` attribute on the `rFonts` element.
- 9 For example, consider a run of text defined as follows:

```
10    <w:r>
11        <w:rPr>
12            <w:rFonts w:ascii="Arial Black" w:hAnsi="Arial Black" w:cs="Arial"
13                w:eastAsia="SimSun"/>
14        </w:rPr>
15        ...
16    </w:r>
```

17 The `rFonts` element specifies that the contents of this run are formatted as follows:

- 18 • Complex script characters used the `Arial` font  
 19 • East Asian characters used the `SimSun` font  
 20 • All other characters used the `Arial Black` font

### 21 **2.9.3 Ambiguous Characters**

22 When classifying characters into one of the four slots defined above, it is likely that the classification of some  
23 characters will be ambiguous (the resulting classification would be equally applicable for one or more font  
24 slots).

25 To handle this, the font face information can also include a `hint`, which specifies how ambiguous mappings are  
26 resolved into a font slot. This information is stored on the `hint` attribute on the `rFonts` element, and specifies  
27 the bucket into which these ambiguous characters fall.

28 For example, if the `hint` attribute has a value of `eastAsia`, then all ambiguous characters shall be formatted  
29 using the East Asian font face.

### 30 **2.9.4 Font Table**

31 Within a document, the *font table* contains information about the fonts used in the document to allow:

- 32 • Applications to perform substitution with the most appropriate possible font when the desired font  
33 face is not available on the system. Since some fonts are commercially distributed, it is possible for a

1 document to be formatted with one or more fonts that are not available depending on the machine  
 2 opening the current system. This information allows the application that cannot locate the desired  
 3 font to perform the most appropriate possible match.

- 4 • Embedding of fonts in the document to prevent the need for font substitution

5 The font table part is stored via an implicit relationship from either the main document part or the glossary  
 6 document part, and has a relationship type of  
 7 <http://schemas.openxmlformats.org/wordprocessingml/2006/fontTable>, and a content type of vnd-  
 8 openxmlformats.officedocument.wordprocessingml-fontTable+xml.

## 9 **2.9.5 Font Substitution Data**

10 The first classification of data stored in the font table are an optional set of font metrics which are queried  
 11 from the font and stored in the document such that future applications can utilize them when the desired font  
 12 is not available. If a particular font face cannot be located on the current system, then this data is used to  
 13 substitute a font that most appropriately matches its characteristics.

14 For example, consider the font substitution data stored for the `Arial Black` font:

```
15 <w:font w:name="Arial Black">
16   <w:panose1 w:val="020B0A04020102020204" />
17   <w:charset w:val="00" />
18   <w:family w:val="swiss" />
19   <w:pitch w:val="variable" />
20   <w:sig w:usb0="00000287" w:usb1="00000000" w:usb2="00000000"
21     w:usb3="00000000" w:csb0="0000009F" w:csb1="00000000" />
22 </w:font>
```

23 This data is linked to the font face with a name of `Arial Black` via the `name` attribute, and stores the  
 24 following information about the font (see the reference material on fonts for more details):

- 25 • The font's Panose-1 number
- 26 • The character set of the font
- 27 • The font's family
- 28 • The font's pitch
- 29 • The code pages and Unicode sub ranges supported by the font

## 30 **2.9.6 Font Embedding**

31 As well as providing information about the font's metrics, applications may be directed to embed the contents  
 32 of a font (partially or as a whole) into a document, a process known as font embedding. *Font embedding*  
 33 literally embeds an obfuscated version of the font into the file so that it may be retrieved and used to view the  
 34 contents of this document - but the obfuscation ensures that the font cannot be extracted and used for any  
 35 other document (as it may have a commercial license).

36 Within the font table, when a font is embedded there are explicit relationships to each font form needed:

- 1     • Regular
  - 2     • Bold
  - 3     • Italic
  - 4     • Bold + Italic
- 5    Each form is obfuscated using the mechanism described in the reference material on this subject.

## 6    **2.9.7      Theme Fonts**

7    As well as storing standard font face information, run properties may store an abstraction for font face  
 8    information known as theme fonts. *Theme fonts* are values that specify that the font face information for a run  
 9    is not stored in the attribute value using the appropriate font face name, but is rather a reference into the  
 10   document's theme part, allowing font face information to be stored and managed centrally as part of the  
 11   theme data. It is appropriate to think of theme fonts as a "style for fonts" in the same way in which a style is a  
 12   reference to the formatting that is stored centrally in another part.

13   Theme fonts are specified using the theme attribute variants in the rFonts element, rather than storing the  
 14   actual font face name.

15   For example, consider a run of text defined as follows:

```
16       <w:r>
17        <w:rPr>
18        <w:rFonts w:asciiTheme="minorHAnsi" w:hAnsiTheme="minorHAnsi" />
19        </w:rPr>
20       ...
21       </w:r>
```

22   The rFonts element's attribute values of asciiTheme and hAnsiTheme both store a reference to a theme font  
 23   stored in the document's theme part (i.e., there is no font with the primary name `minorHAnsi`).

24   Once this information has been established, it is combined with the theme language data stored in the  
 25   document's settings to resolve the appropriate theme fonts from the theme part. The syntax and format of the  
 26   theme part are stored in the DrawingML syntax and discussed in that section.

## 27   **2.10   Numbering**

28   *Numbering* in WordprocessingML refers to symbols—Arabic numerals, Roman numerals, symbol characters  
 29   ("bullets"), text strings, etc.—that are used to label individual paragraphs of text.

30   The following two paragraphs each contain numbering as defined by WordprocessingML: the first uses an  
 31   Arabic numeral, the second a symbol character:

- 32     1. This is a paragraph with numbering information.
- 33     • This is also a paragraph with numbering information.

## 2.10.1 Numbering Part

Numbering information in a WordprocessingML document is stored in the Numbering part within the package, which is stored via an implicit relationship from the Main Document part or Glossary Document part of relationship type <http://schemas.openxmlformats.org/wordprocessingml/2006/numbering> and has a content type of vnd-openxmlformats.officedocument.wordprocessingml-numbering+xml.

## 2.10.2 Numbering Definitions

The specification of a specific set of numbering information is called a *numbering definition*. Numbering definitions are stored in two components:

- Abstract numbering definitions
- Numbering definition instances

As shown below, their relationship is (essentially) that of an abstract and an inherited class.

## 2.10.3 Abstract Numbering Definitions

An *abstract numbering definition* is the basis for all numbering information in a WordprocessingML document, as it defines the appearance and behavior of a specific set of numbered paragraphs in a document, and is defined using the abstractNum element. Although abstract numbering definitions contain all of the numbering information for one type of numbering, they cannot be directly referenced by content (hence their abstract designation), they must be inherited by a numbering definition instance, which itself can be referenced by content. A specific abstract numbering definition in WordprocessingML can be divided into two parts:

- Common numbering properties
- Numbering levels

The complete definition of all abstract numbering properties can be found in the reference for the abstractNum element.

Common numbering properties refer to the properties that can be specified by all abstract numbering definitions regardless of their contents. Examples of common numbering properties include: a numbering ID (which uniquely identifies a numbering definition), the numbering definition type (single level, multi-level, multi-level hybrid), the numbering name, and optional numbering style references, as discussed in detail later in this subclause.

Consider the following example of an abstract numbering definition in a WordprocessingML document:

```

<w:abstractNum w:abstractNumId="4">
  <w:nsid w:val="1DE04504" />
  <w:multiLevelType w:val="hybridMultilevel" />
  <w:lvl w:ilvl="0" w:tplc="0409000F">
    ...
  </w:lvl>
```

```

1   <w:lvl w:ilvl="1" w:tplc="04090019">
2     ...
3   </w:lvl>
4   <w:lvl w:ilvl="2" w:tplc="04090019">
5     ...
6   </w:lvl>
7   <w:lvl w:ilvl="3" w:tplc="0409000F">
8     ...
9   </w:lvl>
10    ...
11 </w:abstractNum>
```

12 This numbering definition specifies two common properties: a numbering ID (using the nsid element) of  
 13 1DE04504, and a list type (using the multiLevelType element) of hybridMultilevel, which specifies that this  
 14 abstract numbering definition is more than one level and contains multiple numbering formats.

15 The other part of an abstract numbering definition is the specification of one or more numbering levels, each  
 16 of which defines a unique set of formatting properties for one level in this numbering definition.

17 Consider three numbered paragraphs that reference the same numbering definition, but each, in turn,  
 18 reference a different level within that list:

- 1. One
  - a. Two
    - i. Three

19  
 20 Although the paragraphs each reference the same abstract numbering definition (which is discussed later),  
 21 each refers to a separate level within that abstract numbering definition, and therefore each has a unique set  
 22 of paragraph and numbering properties.

23 It is important to note that the concept of levels in an abstract numbering definition refers to the levels as  
 24 defined in the file format, and in no way the logical indentation of numbered paragraphs within a  
 25 WordprocessingML document.

26 Consider another set of numbered paragraphs in WordprocessingML, where each subsequent paragraph is a  
 27 different level but references the same abstract numbering definition:

- 1} Level one
  - a} Level two
    - i} Level three
      - {1} Level four
        - {a} Level five

- 1 In this example, the properties of each level of the numbering definition is such that the paragraphs for each  
 2 level are indented arbitrarily. However, this is still a completely valid numbering definition, and the paragraphs  
 3 each represent subsequent levels of the same numbering definition.
- 4 Within an abstract numbering level definition, each numbering level is represented by an `lvl` element that  
 5 defines a single level of numbering information. Numbering levels specify the following properties: starting  
 6 number value, a number format presentation code (e.g., 1 vs. the string literal One), an associated paragraph  
 7 style, which previous level should cause this numbering level to restart, the numbering text, number  
 8 justification, a paragraph properties indentation for this level, etc. The complete definition of all numbering  
 9 level properties can be found on the reference for the `lvl` element.

10 Consider the following numbering level definition in WordprocessingML:

```

11   <w:lvl w:iilvl="1">
12     <w:start w:val="4"/>
13     <w:nfc w:val="3"/>
14     <w:pStyle w:val="Heading1"/>
15     <w:lvlText w:val="BEFORE %2 AFTER %1 END"/>
16     <w:lvlJc w:val="left"/>
17     <w:pPr>
18       <w:tabs>
19         <w:tab w:val="num" w:pos="2880"/>
20       </w:tabs>
21       <w:ind w:left="288" w:firstLine="1152"/>
22     </w:pPr>
23   </w:lvl>
```

24 This particular numbering level defines the following information:

- 25 • This is level 1 (the second level) for this numbering definition
- 26 • Start at number 4
- 27 • Use number format 3 (which translates to 1, 2, 3, and so on)
- 28 • When this level is used, apply the `Heading1` style
- 29 • Use the following level text for the number: BEFORE %2 AFTER %1 END
- 30 • Left justify the number
- 31 • Set a left indent of 288 twentieths of a point, and a first line indent of 1152 twentieths of a point

32 This information is used to display the number for paragraphs of level 1 the reference this numbering  
 33 definition.

34 Of particular significance is the `lvlText` element, which defines the content of the number text for each  
 35 numbering level. Its syntax allows any string literal to be placed in the number (e.g., the ARTICLE in ARTICLE I,  
 36 ARTICLE II, ARTICLE III, and so on), as well as the current value of the number for this or any previous level in  
 37 the list.

1 Referring to the numbering level definition above, the lvlText is defined as follows:

2 <w:lvlText w:val="BEFORE %2 AFTER %1 END"/>

3 This level text specifies three literal strings (BEFORE, AFTER, END) mixed with the current numbering value  
4 from level 1 and level 0 in the document. Therefore, assuming level 0 is just a simple number, when inserted it  
5 would read:

6 1  
7     BEFORE 1 AFTER 1 END  
8     BEFORE 2 AFTER 1 END  
9     BEFORE 3 AFTER 1 END

10 2  
11    BEFORE 1 AFTER 2 END  
12    BEFORE 2 AFTER 2 END  
13 ...

14 The %1 and %2 values correspond to the value for level 0 and 1 of this list, respectively.

## 15 2.10.4 Numbering Definition Instances

16 A numbering definition instance is a specific instantiation of numbering information that can be referenced by  
17 zero or more paragraphs within the document. A numbering definition instance is defined using the  
18 num element. A specific numbering definition instance in WordprocessingML can be divided into two parts:

- 19 • An abstract numbering reference
- 20 • (Optional) level overrides

21 The definition of all numbering definition instance properties can be found on the reference for the num  
22 element.

23 The required piece of information in a numbering definition instance, the instance must reference an abstract  
24 numbering definition using the abstractNumId element. This element specifies the value of the  
25 abstractNumId attribute for the inherited abstract numbering definition information.

26 Consider the WordprocessingML for a document with four numbering definition instances, two of which  
27 reference the same underlying abstract numbering definition:

```
28 <w:numbering>
29 ...
30 <w:num w:numId="2">
31   <w:abstractNumId w:val="0" />
32 </w:num>
33 <w:num w:numId="3">
34   <w:abstractNumId w:val="1" />
35 </w:num>
```

```

1   <w:num w:numId="4">
2     <w:abstractNumId w:val="4" />
3   </w:num>
4   <w:num w:numId="5">
5     <w:abstractNumId w:val="4" />
6   </w:num>
7 </w:numbering>
```

8 As shown above, the first two numbering definition instances reference abstractNumId values of 0 and 1  
9 respectively, and the last two both reference the abstract numbering definition with an abstractNumId of 4.

10 The second (and optional) piece of information for a numbering definition instance is one or more numbering  
11 level overrides using the lvlOverride element. This element specifies a set of optional overrides applied to  
12 zero or more levels from the abstract numbering definition inherited by this instance.

13 Consider a numbering definition instance that inherits its information from the abstract numbering definition  
14 with abstractNumId of 4, but wishes to use a different set of properties for level 0 of the numbering  
15 definition. The resulting WordprocessingML would look like:

```

16   <w:num w:numId="6">
17     <w:abstractNumId w:val="4" />
18     <w:lvlOverride w:ilvl="0">
19       <w:lvl w:ilvl="0">
20         <w:start w:val="4" />
21         <w:lvlText w:val="%1)" />
22         <w:lvlJc w:val="left" />
23         <w:pPr>
24           <w:ind w:left="360" w:hanging="360" />
25         </w:pPr>
26       </w:lvl>
27     </w:lvlOverride>
28   </w:num>
```

29 This level overrides level 0 of the list with the specified set of numbering properties, replacing those in the  
30 abstract numbering definition.

### 31 2.10.5 Applying Numbering to Paragraphs

32 Once numbering information is defined in the numbering part, this information must be associated with  
33 paragraphs within the document in order to display numbering on one or more paragraphs of content.

34 To accomplish this, numbered paragraphs are identified by the numPr element within the paragraph's  
35 properties element (the pPr element). The numbering properties within a paragraph are specified using two  
36 specific elements that specify the numbering definition information to use:

- 37 • A numbering definition instance reference

- 1        • A numbering level reference
- 2        The numbering definition instance reference is specified using the numId element. This element contains a  
3        reference to the numId attribute in a specific numbering definition instance within the numbering part, which  
4        links this paragraph to that numbering definition instance.
- 5        The numbering level reference is specified using the ilvl element. This element contains a reference to the ilvl  
6        attribute in the specified numbering definition instance's level information, which specifies the numbering  
7        level within the referenced numbering definition instance to be used by this numbered paragraph.
- 8        Consider the following numbered paragraphs in a WordprocessingML document:

- 1. Level one item one**
  - a. Level two item one**
- 2. Level one item two**
  - a. Level two item one**

- 9
- 10
- 11        These four numbered paragraphs, all referencing the same numbering definition, produce the following  
12        WordprocessingML:

```
13      <w:p>
14          <w:pPr>
15              <w:numPr>
16                  <w:ilvl w:val="0" />
17                  <w:numId w:val="5" />
18              </w:numPr>
19          </w:pPr>
20          <w:r>
21              <w:t>Level one item one</w:t>
22          </w:r>
23      </w:p>
```

```

1   <w:p>
2     <w:pPr>
3       <w:numPr>
4         <w:ilvl w:val="1" />
5         <w:numId w:val="5" />
6       </w:numPr>
7     </w:pPr>
8     <w:r>
9       <w:t>Level two item one</w:t>
10      </w:r>
11    </w:p>
12    <w:p>
13      <w:pPr>
14        <w:numPr>
15          <w:ilvl w:val="0" />
16          <w:numId w:val="5" />
17        </w:numPr>
18      </w:pPr>
19      <w:r>
20        <w:t>Level one item two</w:t>
21      </w:r>
22    </w:p>
23    <w:p>
24      <w:pPr>
25        <w:numPr>
26          <w:ilvl w:val="1" />
27          <w:numId w:val="5" />
28        </w:numPr>
29      </w:pPr>
30      <w:r>
31        <w:t>Level two item one</w:t>
32      </w:r>
33    </w:p>
```

34 In these numbered paragraphs, level 0 and 1 of the numbering definition are referenced through the `ilvl`  
 35 element with a `val` attribute of 0 or 1, respectively, however, the `numId` element always references the  
 36 numbering definition instance with a `val` of 5.

37 The numbering at any particular numbering level is restarted when a paragraph in the current document from  
 38 the same numbering definition uses the level specified in the `lvlRestart` element for this numbering level.

39 Consider a set of numbered paragraphs in a WordprocessingML document where:

- 40 • Level 1 is set to restart after each level 0 (`lvlRestart` of 1)

- 1        • Level 2 is set to never restart (lvlRestart of 0)
- 1) Level one
- a) Level two – restarts after each level one
    - i) Level three – never restarts
  - b) Level two – restarts after each level one
    - ii) Level three – never restarts
- 2) Level one
- a) Level two – restarts after each level one
    - iii) Level three – never restarts
  - b) Level two – restarts after each level one
    - iv) Level three – never restarts

2

- 3        As the example shows, the numbering at level 1 (a, b, c, and so on) restarts after each level 0 is used, but  
 4        level 2 (i, ii, iii, and so on) never restarts.

## 5        2.10.6     The Complete Story

6        To summarize the use of numbering information in a document, the paragraph properties specify a numPr  
 7        element, which references a numbering definition instance via the numId element. The numbering definition  
 8        instance specifies an inherited abstract numbering definition via the abstractNumId element. The paragraph  
 9        then also specifies the list level from the numbering definition instance using the ilvl element.

10      Consider the following WordprocessingML for a numbered paragraph:

```

11      <w:p>
12          <w:pPr>
13              <w:numPr>
14                  <w:ilvl w:val="0" />
15                  <w:numId w:val="5" />
16              </w:numPr>
17          </w:pPr>
18          <w:r>
19              <w:t>Numbered paragraph</w:t>
20          </w:r>
21      </w:p>
```

22      Based on the numId of 5, the paragraph uses the numbering definition instance with a numId of 5:

```

1   <w:numbering>
2     ...
3       <w:num w:numId="5">
4         <w:abstractNumId w:val="4" />
5       </w:num>
6   </w:numbering>
```

7 Based on the abstractNumId of 4, this instance inherits the abstract numbering definition with an  
8 abstractNumId of 4:

```

9   <w:numbering>
10    <w:abstractNum w:abstractNumId="4">
11      <w:nsid w:val="FFFFFFFFFF7F" />
12      <w:multiLevelType w:val="singleLevel" />
13      <w:lvl w:ilvl="0">
14        <w:start w:val="1" />
15        <w:lvlText w:val="%1." />
16        <w:lvlJc w:val="left" />
17        <w:pPr>
18          <w:tabs>
19            <w:tab w:val="num" w:pos="720" />
20          </w:tabs>
21          <w:ind w:left="720" w:hanging="360" />
22        </w:pPr>
23      </w:lvl>
24    </w:abstractNum>
25    ...
26  </w:numbering>
```

27 Since the numbering definition instance does not specify an override for ilvl 0, the definition for the  
28 corresponding level from the abstract numbering definition is applied to the text.

## 29 **2.10.7 Numbering Styles**

30 As stated earlier in the styles subclause (§2.8), numbering styles are style definitions which specify common  
31 formatting properties for a multi-level numbering format within a document. This means that a numbering  
32 style definition in the styles part defines only a single property: a reference to a numbering definition instance  
33 stored in the document's numbering part, using the numId element within the numPr element. That  
34 numbering definition instance specifies an abstract numbering style, which contains the numbering level  
35 information for the numbering style. It also specifies that it is the basis for the numbering style by back-  
36 referencing the numbering style's styleId attribute via the styleLink element.

37 Unlike paragraph and character styles, numbering styles are never directly referenced by content in the  
38 document—instead, an abstract numbering definition specifies that it contains the underlying numbering

1 information for a numbering style, and one or more numbering definition instances reference that abstract  
 2 numbering definition.

### 3 **2.10.8 Referencing Numbering Styles**

4 To use a numbering style in a document, the paragraph properties for one or more paragraphs again specify a  
 5 numPr element, which references a numbering definition instance via the numId element. The numbering  
 6 definition instance itself again specifies an inherited abstract numbering definition via the abstractNumId  
 7 element.

8 At this stage, the abstract numbering definition specifies that it is based on a numbering style via either of the  
 9 following:

- 10 • The abstract numbering style contains no level data, and simply specifies a reference to the numbering  
 11 style's styleId attribute via the numStyleLink element.
- 12 • The abstract numbering style contains the numbering level information for the numbering style, and  
 13 specifies that it is the basis for the numbering style by referencing the numbering style's styleId  
 14 attribute via the styleLink element.

15 Although the result of each method is identical, the following two examples illustrate each of the syntaxes:

16 Consider the first numbering style syntax, in which the numbering on a paragraph is based on an abstract  
 17 numbering definition which simply references the numbering style via numStyleLink. The contents of the  
 18 paragraph would consist of the following:

```
19 <w:p>
20   <w:pPr>
21     <w:numPr>
22       <w:ilvl w:val="0" />
23       <w:numId w:val="6" />
24     </w:numPr>
25   </w:pPr>
26   <w:r>
27     <w:t>This paragraph references a numbering style via numStyleLink.</w:t>
28   </w:r>
29 </w:p>
```

30 The numId element references a numbering definition instance with a value of 6, located in the numbering  
 31 part:

```
32   <w:num w:numId="6">
33     <w:abstractNumId w:val="0" />
34   </w:num>
```

35 Based on the abstractNumId of 0, this instance inherits the abstract numbering definition with an  
 36 abstractNumId of 0:

```

1   <w:abstractNum w:abstractNumId="0">
2     <w:nsid w:val="38901FA4" />
3     <w:multiLevelType w:val="multilevel" />
4     <w:numStyleLink w:val="TestNumberingStyle" />
5   </w:abstractNum>

6 This abstract numbering definition contains no numbering information - it simply notes that it inherits the
7 numbering information from the numbering style TestNumberingStyle by referencing the styleId
8 attribute on that style:
```

```

9   <w:style w:type="numbering" w:styleId="TestNumberingStyle">
10    <w:name w:val="Test Numbering Style" />
11    <w:uiPriority w:val="99" />
12    <w:rsid w:val="00DB3C4B" />
13    <w:pPr>
14      <w:numPr>
15        <w:numId w:val="4" />
16      </w:numPr>
17    </w:pPr>
18  </w:style>
```

19 The style references a numbering definition instance, again via the numId element:

```

20  <w:num w:numId="4">
21    <w:abstractNumId w:val="2" />
22  </w:num>
```

23 Based on the abstractNumId of 2, this instance inherits the abstract numbering definition with an  
24 abstractNumId of 2:

```

1   <w:abstractNum w:abstractNumId="2">
2     <w:nsid w:val="46364EB7" />
3     <w:multiLevelType w:val="multilevel" />
4     <w:styleLink w:val="TestNumberingStyle" />
5     <w:lvl w:ilvl="0">
6       <w:lvlText w:val="%1 %1 %1" />
7       <w:lvlJc w:val="left" />
8       <w:pPr>
9         <w:tabs>
10           <w:tab w:val="num" w:pos="360" />
11         </w:tabs>
12         <w:ind w:left="0" w:firstLine="0" />
13       </w:pPr>
14     </w:lvl>
15   ...
16 </w:abstractNum>
```

17 This abstract numbering definition defines the properties for each level of the numbering format (levels 1  
18 through 9 omitted for brevity). Since neither of the numbering definition instances specified overrides for  
19 level 0, the properties from abstract numbering format 2 are applied to level 0 in the resulting numbering  
20 definition instance and are applied to the text via the ilvl element.

21 Consider the second numbering style syntax, in which the numbering on a paragraph is based on an abstract  
22 numbering definition which defines the numbering information and references the numbering style via  
23 styleLink. The contents of the paragraph would consist of the following:

```

24   <w:p>
25     <w:pPr>
26       <w:numPr>
27         <w:ilvl w:val="0" />
28         <w:numId w:val="4" />
29       </w:numPr>
30     </w:pPr>
31     <w:r>
32       <w:t>This paragraph references a numbering style via styleLink.</w:t>
33     </w:r>
34   </w:p>
```

35 The numId element references a numbering definition instance with a value of 4, located in the numbering  
36 part:

```

37   <w:num w:numId="4">
38     <w:abstractNumId w:val="2" />
39   </w:num>
```

1 Based on the abstractNumId of 2, this instance inherits the abstract numbering definition with an  
 2 abstractNumId of 2:

```

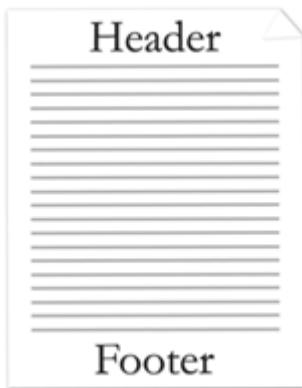
3   <w:abstractNum w:abstractNumId="2">
4     <w:nsid w:val="46364EB7" />
5     <w:multiLevelType w:val="multilevel" />
6     <w:styleLink w:val="TestNumberingStyle" />
7     <w:lvl w:ilvl="0">
8       <w:lvlText w:val="%1 %1 %1" />
9       <w:lvlJc w:val="left" />
10      <w:pPr>
11        <w:tabs>
12          <w:tab w:val="num" w:pos="360" />
13        </w:tabs>
14        <w:ind w:left="0" w:firstLine="0" />
15      </w:pPr>
16    </w:lvl>
17    ...
18  </w:abstractNum>
```

19 This abstract numbering definition defines the properties for each level of the numbering format (levels 1  
 20 through 9 omitted for brevity) and specifies that it is the underlying numbering information for a numbering  
 21 format by referencing the styleId of that numbering style via the styleLink element. Since the numbering  
 22 definition instances specified no override for level 0, the properties from abstract numbering format 2 are  
 23 applied to level 0 in the resulting numbering definition instance and are applied to the text via the ilvl  
 24 element.

## 25 **2.11 Headers and Footers**

26 *Headers* and *footers* refer to text, graphics, or data (such as page number, date, document title, and so on)  
 27 that can appear at the top or bottom of each page in a WordprocessingML document.

28 A *header* appears in the top margin (above the main document content on the page), while a *footer* appears in  
 29 the bottom margin of a document page (below the main document content on the page); for example:



1

2 Since WordprocessingML is a flow-based format, headers and footers are applied by specifying the headers  
 3 and footers for all pages in a particular section of a document.

#### 4 **2.11.1 Header Part**

5 Header information in a WordprocessingML document is stored in a header part within the package, which is  
 6 stored via an implicit relationship from the Main Document part or the Glossary Document part of relationship  
 7 type `http://schemas.openxmlformats.org/wordprocessingml/2006/header` and has a content  
 8 type of `vnd-openxmlformats.officedocument.wordprocessingml-header+xml`.

#### 9 **2.11.2 Footer Part**

10 Footer information in a WordprocessingML document is stored in a footer part within the package, which is  
 11 stored via an implicit relationship from the Main Document part or the Glossary Document part of relationship  
 12 type `http://schemas.openxmlformats.org/wordprocessingml/2006/footer` and has a content  
 13 type of `vnd-openxmlformats.officedocument.wordprocessingml-footer+xml`.

#### 14 **2.11.3 Headers and Footers**

15 As described above, header and footer information is stored in one or more header or footer parts within the  
 16 package.

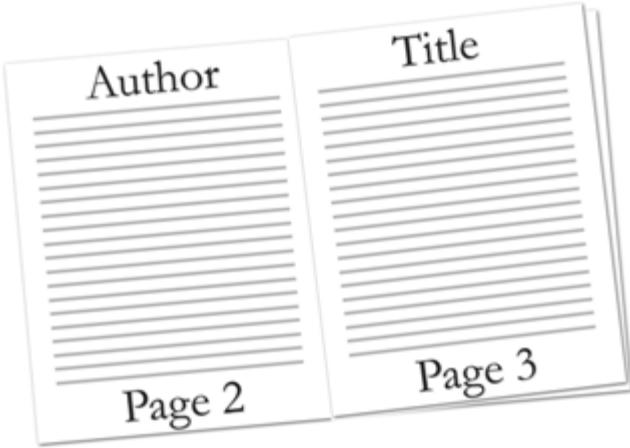
17 The `hdr` element defines a single header for the document, while the `ftr` element defines a single footer for  
 18 the document. Headers and footers are just another document story in WordprocessingML. Within the root  
 19 element of the header or footer, the content of the element is similar to the content of the `body` element, and  
 20 contains what is referred to as *block-level markup* —markup that can exist as a sibling element to paragraphs  
 21 in a WordprocessingML document.

22 Within each section of a document there can be up to three different types of headers and footers:

- 23 • First page header/footer
- 24 • Odd page header/footer
- 25 • Even page header/footer

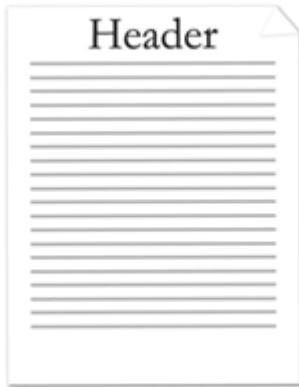
26 *First page headers and footers* specify a unique header or footer that shall appear on the first page of a  
 27 section. *Odd page headers and footers* specify a unique header and footer that shall appear on all odd  
 28 numbered pages for a given section. *Even page headers and footers* specify a unique header and footer that  
 29 shall appear on all even numbered pages in a given section.

30 Different headers or footers can be useful for bounded documents like books, as shown in the figure below.



1

2 Consider the following simple one-page document with one header:



3

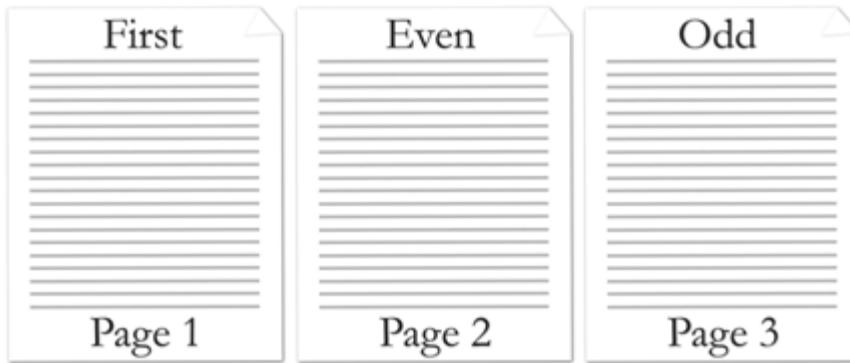
4 This document defines one header with the text Header. The header's content is stored in a unique Header part. The resulting header is represented by the following WordprocessingML:

```

6   <w:hdr>
7     <w:p>
8       <w:r>
9         <w:t>Header</w:t>
10        </w:r>
11      </w:p>
12    </w:hdr>
```

13 Since headers are containers of block level contents, all block level contents can be used within them. In this  
14 particular example, the content is a single paragraph.

15 Consider a more complex three-page document with different first, odd, and even page headers defined:



1  
 2 This document defines three headers stored in three different header parts. The resulting headers are  
 3 represented by the following WordprocessingML:

4 First page header part:

```
5   <w:hdr>
 6     <w:p>
 7       <w:r>
 8         <w:t>First</w:t>
 9       </w:r>
10     </w:p>
11   </w:hdr>
```

12 Even page header part:

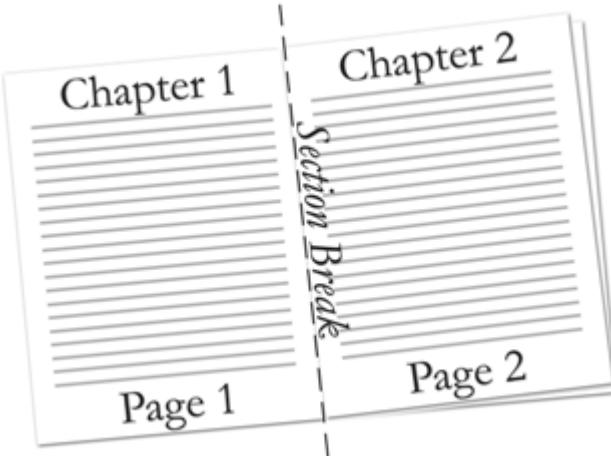
```
13   <w:hdr>
14     <w:p>
15       <w:r>
16         <w:t>Even</w:t>
17       </w:r>
18     </w:p>
19   </w:hdr>
```

20 Odd page header part:

```
21   <w:hdr>
22     <w:p>
23       <w:r>
24         <w:t>Odd</w:t>
25       </w:r>
26     </w:p>
27   </w:hdr>
```

## 2.11.4 Multiple Sections

- Documents are capable of having multiple sections, where each section can define up to three headers and footers. By default, sections other than the first section inherit the previous header and footer references, unless that section specifies header and footer references.



- Consider a two-page, two-section document with only the first section header defined. This document defines one header that is referenced in the first section. The document is represented by the following WordprocessingML:

```

9   <w:body>
10  ...
11  <w:p>
12    <w:pPr>
13      <w:sectPr>
14        <w:headerReference r:id="rId6" />
15      ...
16    </w:sectPr>
17  </w:pPr>
18  ...
19  </w:p>
20  ...
21  <w:sectPr>
22  ...
23  </w:sectPr>
24  </w:body>
```

- The second section does not explicitly reference a header. Instead, the second section inherits the header from the previous section.

## 1    2.11.5    Empty Header or Footer

2    Not specifying a header and footer reference in a section, other than the first section, causes the document to  
 3    inherit the previous section's header and footer references. In order to declare an empty header or footer, a  
 4    header or footer reference must be made to a null header or footer relationship, as follows:

5        <Relationship Id="rId2" Type="http:// .../header" Target="null" />

6    The null attribute value specifies that the header or footer shall not be inherited from the previous section,  
 7    and a blank header or footer shall explicitly be used.

## 8    2.12    Footnotes and Endnotes

9    *Footnotes* and *endnotes* are separate text stories used in documents and books to show the source of  
 10   borrowed material or to enter explanatory or supplementary information that does not interrupt the normal  
 11   reading flow of the document.

12   *Footnotes* are typically located at the bottom of a page or beneath text being referenced, and *endnotes* are  
 13   typically placed at the end of a document or at the end of a section. If document has been divided up into one  
 14   or more sections, each section of a document can contain endnotes.

15   Both footnotes and endnotes consist of two parts:

- 16     • A note reference mark in the body text to indicate that additional information is in a footnote or  
 17        endnote, with a numbering system used for each to tell readers whether to look for the note at the  
 18        end of the page or the end of the document or section.
- 19     • The actual footnote or endnote story content.

20   Here's an example of a footnote applied to text in a document:



21

22

- 1 The note reference mark follows the noted text and specifies that there is associated footnote information; the
- 2 footnote itself is at the bottom of the current page.
- 3 Consider the following example of an endnote applied to text in a document:



- 4
- 5
- 6 The note reference mark follows the noted text and specifies that there is associated endnote information; the
- 7 endnote itself is at the end of the current section.

### 8 **2.12.1 Footnote Part**

9 Footnote information in a WordprocessingML document is stored in the footnotes part within the package,  
10 which is stored via an implicit relationship from the Main Document part or Glossary Document part of  
11 relationship type <http://schemas.openxmlformats.org/wordprocessingml/2006/footnotes> and  
12 has a content type of vnd-openxmlformats.officedocument.wordprocessingml-footnotes+xml.

### 13 **2.12.2 Endnote Part**

14 Endnote information in a WordprocessingML document is stored in the Endnotes part within the package,  
15 which is stored via an implicit relationship from the Main Document part or Glossary Document part of

1 relationship type <http://schemas.openxmlformats.org/wordprocessingml/2006/endnotes> and  
 2 has a content type of `vnd-openxmlformats.officedocument.wordprocessingml-endnotes+xml`.

### 3 **2.12.3 Footnotes and Endnotes**

4 As described above, footnote and endnote information is stored in the corresponding footnotes and endnotes  
 5 part within the package. The footnotes element below specifies three or more footnotes, each identified by  
 6 the footnote element, for the document. The endnotes element specifies three or more endnotes, each  
 7 identified by the endnote element, for the document. Each footnote or endnote element is associated with a  
 8 unique ID, specified by the attribute id.

9 Consider three different types of footnotes, each identified by a footnote element, defined in the Footnotes  
 10 part. The use of each type of footnote is defined in the next subclause:

```
11   <w:footnotes ...>
12     <w:footnote w:type="separator" w:id="0">
13       ...
14     </w:footnote>
15     <w:footnote w:type="continuationSeparator" w:id="1">
16       ...
17     </w:footnote>
18     <w:footnote w:id="2">
19       ...
20     </w:footnote>
21   </w:footnotes>
```

22 Similarly consider three different types of endnotes, each identified by an endnote element, defined in the  
 23 Endnotes part. The use of each type of endnote is defined in the next subclause:

```
24   <w:endnotes ...>
25     <w:endnote w:type="separator" w:id="0">
26       ...
27     </w:endnote>
28     <w:endnote w:type="continuationSeparator" w:id="1">
29       ...
30     </w:endnote>
31     <w:endnote w:id="2">
32       ...
33     </w:endnote>
34   </w:endnotes>
```

35 Footnotes and endnotes are just another kind of paragraph in WordprocessingML. Within the footnote or  
 36 endnote element, the footnote or endnote may contain any valid block-level content.

## 2.12.4 Footnote and Endnote Types

There are four different types of footnotes and endnotes:

- Normal – contain the text of any footnote (or endnote) in the document.
- Separator – define the separator used to separate the footnote (or endnote) from the document text.
- Continuation separator – define the separator used to separate the footnote (or endnote) from the document text when the footnote or endnote is a continuation from the previous page.
- Continuation notice – define the notice text to let readers know that the footnote (or endnote) has continued on the next page.

The attribute type specifies the type of footnote or endnote. Normal footnotes or endnotes are specified by a type of normal or by omitting type. In conjunction to a normal type, a footnote reference mark, specified by footnoteRef element, or endnote reference mark, specified by endnoteRef element, must be present within the footnote or endnote definition.



13

Consider the following page in a document, where some text is referenced by a footnote at the end of a page:

The footnote text at the bottom of the page is a normal type footnote represented by the following WordprocessingML:

```

17   <w:footnote w:id="2">
18     <w:p>
19       <w:pPr>
20         <w:pStyle w:val="FootnoteText" />
21       </w:pPr>

```

```

1      <w:r>
2          <w:rPr>
3              <w:rStyle w:val="FootnoteReference" />
4          </w:rPr>
5          <w:footnoteRef />
6      </w:r>
7      <w:r>
8          <w:t>Cool reference</w:t>
9      </w:r>
10     </w:p>
11 </w:footnote>
```

12 Not specifying any type attribute in the footnote element defaults to being a `normal` type of footnote. In this  
 13 example, the footnote has a unique ID of 2. The text of the footnote is contained in the text run. Like any  
 14 paragraph, footnotes can be associated with a particular style, and, in this example, the paragraph uses the  
 15 `FootnoteText` paragraph style. Similarly, like any run, footnotes can be associated with a particular style,  
 16 and, in this example, the run uses the `FootnoteReference` run style.

17 Separator footnotes or endnotes are specified by separator. These types of footnotes or endnotes define the  
 18 look of the separator used to separate document text from footnotes or endnotes. In conjunction to  
 19 separator type, a footnote or endnote separator reference mark, specified by a separator element must be  
 20 present within the footnote or endnote definition.

21 Consider the following page in a document, where some text is referenced by a footnote at the end of a page:



22

23

24 The line separating the document text from the footnote is represented by the following WordprocessingML:

```

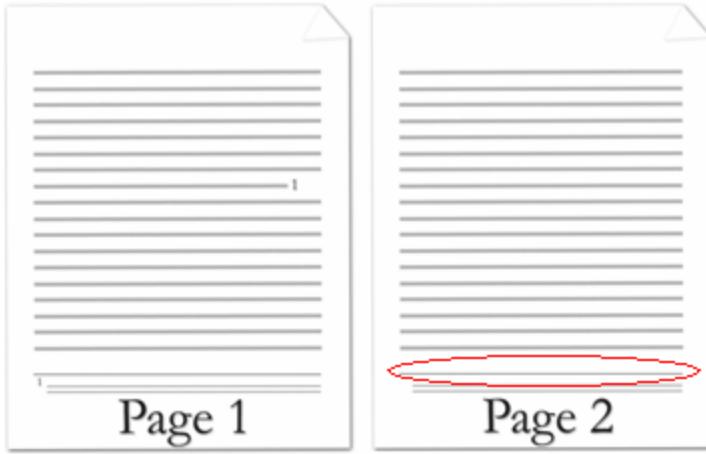
1   <w:footnote w:type="separator" w:id="0">
2     <w:p>
3       <w:pPr>
4         <w:spacing w:after="0" w:line="240" w:lineRule="auto" />
5       </w:pPr>
6       <w:r>
7         <w:separator />
8       </w:r>
9     </w:p>
10   </w:footnote>

```

11 In this example, the footnote has a unique ID of 0. The vertical spacing after the line separator is 0 twentieths  
 12 of a point. The vertical spacing between the line separator and text is 240 twentieths of a point.

13 Continuation separator footnotes or endnotes are specified by continuationSeparator. These types of  
 14 footnotes or endnotes define the look of the separator used to separate document text from footnotes or  
 15 endnotes when the footnote or endnote continues the next page. In conjunction to a  
 16 continuationSeparator type, a footnote or endnote continuation separator reference mark, specified by  
 17 continuationSeparator element must be present within the footnote or endnote definition.

18 Consider the following two pages in a document, where some text is referenced by a footnote that extends to  
 19 the next page:



20

21

22 The line separating the document text from the footnote that is continued on the next page (circled in red in  
 23 the image above) is the continuation separator footnote, and is represented by the following  
 24 WordprocessingML:

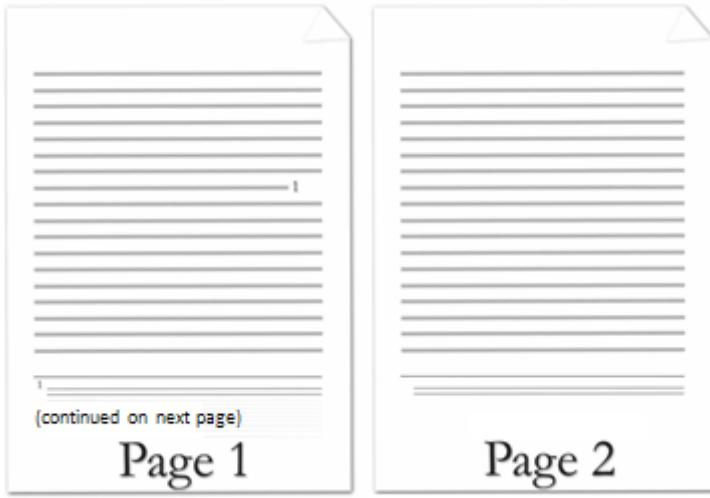
```

1   <w:footnote w:type="continuationSeparator" w:id="1">
2     <w:p>
3       <w:pPr>
4         <w:spacing w:after="0" w:line="240" w:lineRule="auto" />
5       </w:pPr>
6     <w:r>
7       <w:continuationSeparator />
8     </w:r>
9   </w:p>
10  </w:footnote>
```

11 In this example, the footnote has a unique ID of 1. The vertical spacing after the line separator is 0 twentieths  
 12 of a point. The vertical spacing between the line separator and text is 240 twentieths of a point.

13 Continuation notice footnotes or endnotes are specified by continuationNotice. These types of footnotes or  
 14 endnotes specify the text to let readers know that the footnote or endnote is continued on the next page.

15 Consider the following two pages in a document, where some text is referenced by a footnote that extends to  
 16 the next page. A continuation notice is given to readers to indicate that the footnote extends to the next page:



17 The continuation notice text is at the bottom of the footnote indicating that the footnote is continued to the  
 18 next page (which reads continued on next page above) and is represented by the following  
 19 WordprocessingML:  
 20

```

21   <w:footnote w:type="continuationNotice" w:id="3">
22     <w:p>
23       <w:pPr>
24         <w:spacing w:after="0" w:line="240" w:lineRule="auto" />
25       </w:pPr>
```

```

1      <w:r>
2          <w:t>(continued on next page)</w:t>
3      </w:r>
4  </w:p>
5  </w:footnote>
```

6 In this example, the footnote has a unique ID of 3. The text that shows up after the footnote text is  
7 (continued on next page).

### 8 2.12.5 Footnote and Endnote Reference

9 Once footnote or endnote information is defined in the footnotes or endnotes part, this information must be  
10 associated with document text within the document in order to display the footnotes or endnotes. Each  
11 footnote or endnote is identified by a unique ID that references footnote or endnote definitions specified in  
12 the footnotes or endnotes part. Footnote or endnote references are identified by the footnoteReference or  
13 endnoteReference element within the text run's element (the r element). The footnoteReference or  
14 endnoteReference element points to the footnote or endnote ID defined in the footnotes or endnotes part.

15 Consider the following one-page document, where some text is referenced by a footnote at the end of the  
16 document page:



17

18

19 The footnote references text and is represented by the following WordprocessingML:

```

20      <w:p>
21          <w:r>
22              <w:t>Some referenced text</w:t>
23          </w:r>
```

```

1   <w:r>
2     <w:rPr>
3       <w:rStyle w:val="FootnoteReference" />
4     </w:rPr>
5     <w:footnoteReference w:id="2" />
6   </w:r>
7 </w:p>

```

8 The footnote references the footnote in the footnotes part with ID equals to 2. Like any run, footnotes can be  
9 associated with a particular style, and, in this example, the run uses the `FootnoteReference` run style. The  
10 style of the footnote defines the look and numbering of the footnote.

## 11 **2.13 Glossary Document**

12 The introduction to a WordprocessingML document formally introduced the concept of stories, individual  
13 ranges of a word-processing document containing block-level content like paragraphs and tables. Some  
14 examples of stories in a WordprocessingML document include the following: the main document, headers,  
15 footers, comments, footnotes, and endnotes.

16 At that time, a story was defined by two characteristics:

- 17 • It is a unique region containing block-level content
- 18 • All document stories shared the same set of properties (e.g., style definitions, numbering definitions,  
19 and settings)

20 The glossary document, although it follows the first rule, actually defies the second.

21 Within a WordprocessingML file, the *glossary document* is a supplemental storage location for additional  
22 document content which shall travel with the document, but which shall not be displayed for printed as part of  
23 the main document until it is explicitly added to that document by deliberate action.

24 The glossary document shall also be afforded a separate instance of all of the relationships that are provided  
25 on the main document part - this means that the glossary document shall have its own style definitions,  
26 numbering definitions, comments, headers, footers, etc. within the WordprocessingML document.

27 Consider a document that shall include ten optional clauses that may be inserted through a user interface. It is  
28 clearly not desirable to have these ten clauses appear in the main document story's contents before they are  
29 explicitly inserted, therefore each of them may be stored in the glossary document and inserted via the user  
30 interface as needed.

31 Within the glossary document, each distinct region of document content is referred to as a *glossary document*  
32 *entry*, and is defined via the `docPart` element. These document parts may contain any block-level  
33 WordprocessingML element, and may also have a set of classifications and behaviors applied to them via the  
34 glossary document entry's properties.

1 Consider the following definition for the contents of a glossary document part within a WordprocessingML  
 2 document:

```

3   <w:glossaryDocument>
4     <w:docParts>
5       <w:docPart>
6         <w:docPartPr>
7           ...
8           </w:docPartPr>
9         <w:docPartBody>
10        <w:p>
11          <w:r>
12            <w:t>Sample entry.</w:t>
13          </w:r>
14        </w:p>
15      </w:docPartBody>
16    </w:docPart>
17    <w:docPart>
18      ...
19      </w:docPart>
20    </w:docParts>
21  </w:glossaryDocument>
```

22 The glossaryDocument element defines the contents of the glossary document part. Within the glossary  
 23 document, each docPart element contains the definition for one glossary document entry: in this case, there  
 24 are two entries in the glossary document, the first of which contains a single paragraph with a single run of  
 25 text.

26 Each glossary document entry consists of two components:

- 27 • The entry's properties, specified using the docPartPr element  
 28 • The entry's contents, specified using the docPartBody element

29 The first specifies information about the entry (e.g., its classification) for when it is inserted, the latter stores  
 30 the block level content which constitutes the entry.

## 31 **2.14 Annotations**

### 32 **2.14.1 Introduction**

33 An *annotation* is one of various kinds of supplementary markup, which may be stored inside or around a  
 34 region of text within the document's contents. The kinds of supplementary information stored within a  
 35 document can include comments (§2.14.5), revisions (§2.14.7), spelling and/or grammatical errors (§2.14.10),  
 36 bookmark information (§2.14.8), and optional editing permissions (§2.14.9).

37 Within a document's contents, annotations are stored in one of three different forms:

- 1       • Inline
- 2       • Cross-Structure
- 3       • Properties

4     These three forms are needed in order to maintain compatibility with both the legacy annotations  
 5     functionality of current word-processing applications and the requirements of an XML-based format (i.e., well-  
 6     formedness of the resulting XML markup). These three forms are referenced within the individual annotation  
 7     types described in subclauses §2.14.2 through §2.14.4.

## 8     **2.14.2     Inline Annotations**

9     An *inline annotation* is a form of annotation that does not require special handling in order to maintain the  
 10    XML well-formedness requirements of the resulting WordprocessingML output. In these cases, a single XML  
 11    element shall encapsulate the entire contents of the document content which is being annotated.

12    Consider the following WordprocessingML markup for a paragraph that reads The quick brown fox  
 13    jumps over the jet lagged dog., where jet lagged replaced the previous text lazy when the  
 14    editing application was tracking revisions:

```
15    <w:p>
16      <w:r>
17        <w:t xml:space="preserve">The quick brown fox jumps over the </w:t>
18      </w:r>
19      <w:del ... >
20        <w:r>
21          <w:delText>lazy</w:delText>
22          </w:r>
23        </w:del>
24        <w:ins ... >
25          <w:r>
26            <w:t>jet lagged</w:t>
27            </w:r>
28        </w:ins>
29        <w:r>
30          <w:t xml:space="preserve"> dog.</w:t>
31          </w:r>
32      </w:p>
```

33    The del and ins elements each fully encapsulate the extent of their respective annotations (a marked deletion  
 34    and insertion, respectively), as they are inline annotations.

## 35    **2.14.3     Cross-Structure Annotations**

36    A *cross-structure annotation* is a form of annotation that can span portions of WordprocessingML markup.  
 37    (Cross-structure annotations may span parts of multiple paragraphs, one half of a custom XML markup  
 38    element's contents, and so on.) In these cases, the annotation's region is delimited by two elements, a start

1 element and an end element, which mark the start and end points of the annotated content, respectively, but  
 2 do not contain it. Matching start and end markers have the same id attribute value.

3 Consider the following WordprocessingML markup for two paragraphs, each reading Example Text, where a  
 4 bookmark has been added spanning the second word in paragraph one, and the first word in paragraph two:

```

5   <w:p>
6     <w:r>
7       <w:t>Example</w:t>
8     </w:r>
9     <w:bookmarkStart w:id="0" w:name="sampleBookmark" />
10    <w:r>
11      <w:t xml:space="preserve"> text.</w:t>
12    </w:r>
13  </w:p>
14  <w:p>
15    <w:r>
16      <w:t>Example</w:t>
17    </w:r>
18    <w:bookmarkEnd w:id="0" />
19    <w:r>
20      <w:t xml:space="preserve"> text.</w:t>
21    </w:r>
22  </w:p>
```

23 The bookmarkStart and bookmarkEnd elements specify the location where the bookmark starts and ends,  
 24 but cannot contain that bookmark because it spans parts of two paragraphs. They are part of one group  
 25 because the id attribute value specifies 0 for both.

#### 26 **2.14.4 Property Annotations**

27 A *property annotation* is a form of annotation that is stored as a property on an object (Property annotations  
 28 may appear on paragraph properties, run properties, table rows, and so on.) In these cases, the annotation's  
 29 semantics are defined by the property, as they can affect content and/or formatting.

30 Consider the following WordprocessingML markup for a paragraph reading Example Text, where the first  
 31 word had the bold property applied when the editing application was tracking revisions:

```

1   <w:p>
2     <w:r>
3       <w:rPr>
4         <w:b/>
5         <w:rPrChange ... >
6           <w:rPr/>
7         </w:rPrChange>
8       </w:rPr>
9       <w:t>Example</w:t>
10    </w:r>
11    <w:r>
12      <w:t xml:space="preserve"> text.</w:t>
13    </w:r>
14  </w:p>

```

15 The `rPrChange` element contains the set of previously applied revision properties associated with a particular  
16 author at a particular time. It is stored itself as a property on the parent run which was modified.

## 17 2.14.5 Comments

18 A *comment* is an annotation that is anchored to a region of document content, but which contains an arbitrary  
19 amount of block-level content stored in its own separate document story. Within a WordprocessingML  
20 document, comments are stored in a separate comments part within the document package.

21 A comment in a WordprocessingML document is divided into two components:

- 22 • The comment anchor (the text to which the comment applies)
- 23 • The comment content (the contents of the comment)

24 The *comment anchor* is the cross-structure annotation that defines the region of text on which the comment is  
25 anchored. The *comment content* is the text of the comment.

26 Consider a paragraph in a WordprocessingML document whose second word is annotated with a comment:

27 

28 The first component to this comment is the document content, which defines the extents of the comment and  
29 references the specific comment in the comments part:

```

30 <w:p>
31   <w:r>
32     <w:t xml:space="preserve">Some </w:t>
33   </w:r>

```

```

1   <w:commentRangeStart w:id="0" />
2   <w:r>
3     <w:t>text.</w:t>
4   </w:r>
5   <w:commentRangeEnd w:id="0" />
6   <w:r>
7     <w:commentReference w:id="0" />
8   </w:r>
9   </w:p>

```

10 The commentRangeStart and commentRangeEnd elements delimit the run content to which the comment  
 11 with an id of 0 applies (in this case, the single run of text). The commentReference element that follows links  
 12 the preceding run content with a comment in the comments part having an id of 0. Without all three of these  
 13 elements, the range and comment cannot be linked (although the first two elements are optional, in which  
 14 case the comment shall be anchored at the comment reference mark)

15 The second component to this comment is the comment content, which defines the text in the comment:

```

16   <w:comment w:id="0" w:author="Joe Smith"
17     w:date="2006-04-06T13:50:00Z" w:initials="User">
18   <w:p>
19     <w:pPr>
20       <w:pStyle w:val="CommentText" />
21     </w:pPr>
22     <w:r>
23       <w:rPr>
24         <w:rStyle w:val="CommentReference" />
25       </w:rPr>
26       <w:annotationRef />
27     </w:r>
28     <w:r>
29       <w:t>comment</w:t>
30     </w:r>
31   </w:p>
32 </w:comment>

```

33 In this example, the comment specifies that it was inserted by author Joe Smith with the initials User via the  
 34 author and date attributes. It is linked to the run content via the id attribute, which matches the value of 0  
 35 specified using the commentReference element above. The block-level content of the comment specifies that  
 36 its text is comment and the style of the comment content is based off of the character style with the name  
 37 CommentReference.

## 1    2.14.6    Comments Part

2    Comment information in a WordprocessingML document is stored in the Comments part within the package,  
 3    which is stored via an implicit relationship from the Main Document or Glossary Document part of relationship  
 4    type <http://.../comments> and has a content type of vnd-  
 5    openxmlformats.officedocument.wordprocessingml-comments+xml.

## 6    2.14.7    Revisions

7    A *revision* provides a mechanism for storing information about the evolution of the document (i.e., the set of  
 8    modifications made to a document by one or more authors). When an application adds revisions to the  
 9    content of a WordprocessingML document, depending on the revision type they are specifying this by storing  
 10   either:

- 11     • The current state of the document (a deletion stores the current state of the text as deleted, and  
 12       implies that its original state was the content that used to exist)
- 13     • The initial state of the document (a run's initial properties are explicitly stored in a previous run  
 14       properties block, as the current run properties are always those that are the child of the rPr element

15   A revision consists of two required pieces of information:

- 16     • The revision type (specified via the name of the revision element)
- 17     • A unique revision identifier (used to uniquely identify revisions)

18   As well as optional information:

- 19     • The author of the revision
- 20     • The date and time of the revision

21   A revision is stored using the inline annotation format or the property annotation format.

22   Consider a paragraph of text in a WordprocessingML document in which one word has been inserted, as  
 23   follows:

24     Some text

25   This paragraph has the word **text** marked inserted as a revision, and is represented as the following  
 26   WordprocessingML:

```
27     <w:p>
28       <w:r>
29         <w:t>Some</w:t>
30       </w:r>
31       <w:ins w:id="0" w:author="Joe Smith" w:date="2006-03-31T12:50:00Z">
32         <w:r>
33           <w:t>text</w:t>
```

```

1      </w:r>
2      </w:ins>
3      </w:p>

4 The ins element contains all of the content that shall be treated as revision marked as inserted (i.e., the word
5 text).

6 This means that it contains both required pieces of information: the revision type, specified by the name of the
7 revision element (ins); and a unique revision identifier of 0.

8 The element also stores the optional information about the revision: the word text was inserted by Joe
9 Smith on March 31, 2006 at 12:50 pm.

10 Within a WordprocessingML document, the following types of revisions can be used to track the changes to a
11 document (each annotation's form in parentheses):

12 • Insertions (inline annotations for run content, property annotations for tables and paragraphs)
13 • Deletions (inline annotations for run content, property annotations for tables and paragraphs)
14 • Moves (inline annotations)
15 • Changes to run/paragraph/table/numbering/section properties (property annotations)
16 • Changes to custom XML markup (property annotations)

```

## 2.14.8 Bookmarks

A *bookmark* refers to an arbitrary region of content that is bounded and has a unique name associated with it.

Because bookmarks are a legacy word-processing function that predates the concepts of XML and well-formedness, they can start and end at any location within a document's contents and, therefore, must use the cross-structure annotation format described in §2.14.3.

Consider the following WordprocessingML markup for two paragraphs, each reading Example Text, where a bookmark has been added spanning the second word in paragraph one and the first word in paragraph two:

```

24 <w:p>
25   <w:r>
26     <w:t>Example</w:t>
27   </w:r>
28   <w:bookmarkStart w:id="0" w:name="sampleBookmark" />
29   <w:r>
30     <w:t xml:space="preserve"> text.</w:t>
31   </w:r>
32 </w:p>

```

```

1   <w:p>
2     <w:r>
3       <w:t>Example</w:t>
4     </w:r>
5     <w:bookmarkEnd w:id="0" />
6     <w:r>
7       <w:t xml:space="preserve"> text.</w:t>
8     </w:r>
9   </w:p>

```

- 10 The bookmarkStart and bookmarkEnd elements specify the location where the bookmark starts and ends,  
 11 but cannot contain it using a single tag because it spans parts of two paragraphs. However, the two tags are  
 12 part of one group because the id attribute value specifies 0 for both.

### 13 2.14.9 Range Permissions

14 A *range permission* refers to a special type of bookmark used to control which subset(s) of users may edit a  
 15 particular region of a document. Range permissions specify the user, or set of users, that are allowed to edit all  
 16 content between them whenever the document protection specified by the documentProtection element is  
 17 enabled and set to `readOnly` or `comments`.

18 Like bookmarks, range permissions are a legacy word-processing function that predates the concepts of XML  
 19 and well-formedness, so they can start and end at any location within a document's contents and, therefore,  
 20 must use the cross-structure annotation format described in §2.14.3.

21 Consider the following WordprocessingML markup for a single paragraph, where a range permission has been  
 22 added spanning the words `range permission`:

```

23   <w:p>
24     <w:r>
25       <w:t xml:space="preserve">This is a </w:t>
26     </w:r>
27     <w:permStart w:id="0" w:edGrp="everyone"/>
28     <w:r>
29       <w:t>range permission</w:t>
30     </w:r>
31     <w:permEnd w:id="0"/>
32     <w:r>
33       <w:t>. </w:t>
34     </w:r>
35   </w:p>

```

36 The permStart and permEnd elements specify the location where the range permission starts and ends. The  
 37 two tags are part of one group because the id attribute value specifies 0 for both.

1 If document protection was enabled, then no content in this document shall be editable except for this range  
 2 permission, which is editable by all users that open the document (specified using an editor group of  
 3 everyone).

#### 4 **2.14.10 Spelling and Grammar**

5 A *spelling and grammar error* is an annotation used to specify the locations of an existing spelling and/or  
 6 grammatical error within the contents of a document. Spelling and grammar errors use the cross-structure  
 7 annotation format.

8 **Rationale:** When a WordprocessingML document is saved, applications may choose to save currently flagged  
 9 spelling and grammar errors, for two reasons:

- 10 • In order to increase the performance subsequent loads of the document (as those load operations can  
 11 rely on the persisted proofing state of the document)
- 12 • In order to store words which shall not be marked as proofing errors regardless of how they would  
 13 normally be flagged by the proofing tools engine (i.e., to store spelling and grammar exceptions).

14 Consider the following paragraph consisting of two misspelled words, where the second word has been  
 15 explicitly flagged as not being a spelling error. This paragraph would consist of the following  
 16 WordprocessingML markup:

```
17 <w:p>
18   <w:proofErr w:val="spellStart"/>
19   <w:r>
20     <w:t>erqwt</w:t>
21   </w:r>
22   <w:proofErr w:val="spellEnd"/>
23   <w:r>
24     <w:t xml:space="preserve"> werewr</w:t>
25   </w:r>
26 </w:p>
```

27 The proofErr elements, with a val attribute value of spellStart and spellEnd, respectively, delimit the start  
 28 and end of the content in this paragraph that is stored as a spelling error. Since the second word is not  
 29 included in that range, it is not stored as a spelling error.

#### 30 **2.15 Mail Merge**

31 Mail merge refers to a process by which a WordprocessingML document is connected to and populated with  
 32 external data by a conforming hosting application and/or data source access application. A WordprocessingML  
 33 document that contains the necessary data to connect to an external data source during a Mail Merge is  
 34 known as a source document. In other words, a source document is a WordprocessingML document containing  
 35 the elements and attributes necessary to enable the document to connect to an external data source, but not  
 36 yet merged with any data.

1 Applications leverage source documents to generate new documents containing the static content contained  
2 in the merged document as well as data from the specified external data source. The documents that result  
3 from importing external data into a source document are known as merged documents. How source  
4 documents and merged documents are specified is explained in the following sections.

## 5 **2.15.1 Mail Merge, WordprocessingML, and Hosting Applications**

6 The two key parts of the mail merge process are:

- 7     1. Connecting to an external data source  
8     2. Populating mail merge fields with external data

9 It is important to note that aspects of the mail merge process outside of connecting to an external data source  
10 and populating mail merge fields with external data, are at the discretion of the hosting application.

11 As an additional example, WordprocessingML provides an element to be used as a flag by hosting applications  
12 to specify action to be taken on the merged documents that are generated by a mail merge. In other words,  
13 performing actions such as:

- 14     • creating a new document for each merged document  
15     • generating and sending emails containing merged document  
16     • printing merged documents

17 may be specified through WordprocessingML, but what if any specific action is taken on merged documents is  
18 determined by the application.

## 19 **2.15.2 Connecting Documents to an External Data Source**

20 As mentioned, a source document is the single WordprocessingML document that contains the data necessary  
21 to be connected to an external data source by a conforming hosting application and/or data source access  
22 application. The presence and parameters of this connection are specified within the mailMerge element. This  
23 element enables WordprocessingML documents to be connected to an external data source by specifying the  
24 following data:

- 25     • Where the external data is located (e.g., file path)  
26     • What type of data the external data source contains (e.g., database and spreadsheet)  
27     • How the data will be accessed

28 Consider a document containing static WordprocessingML constructs such as paragraphs in addition to two  
29 WordprocessingML mail merge fields calling for `Courtesy Title` and `Last Name` data.

Dear {MERGEFIELD "Courtesy Title" \m}  
{MERGEFIELD "Last Name" \m},

Sample text. Sample text. Sample text.  
Sample text. Sample text. Sample text. Sample  
text. Sample text. Sample text. Sample text.  
Sample text. Sample text. Sample text. Sample  
text. Sample text. Sample text. Sample text.  
Sample text. Sample text. Sample text. Sample  
text. Sample text. Sample text. Sample text.  
Sample text.

Sincerely,

1

2 If the following WordprocessingML was added to this document, this document would become a source  
3 document rather than just a standard WordprocessingML document, as the mailMerge element specifies the  
4 elements and attributes necessary to enable the hosting applications and/or data source access applications to  
5 connect the document to an external data source.

```
6   <w:mailMerge>
7     ...
8       <w:dataType w:val="database" />
9       <w:query w:val="SELECT * FROM Table1" />
10      <w:dataSource r:id="rId1" />
11      ...
12   </w:mailMerge>
```

13 Here, the dataType and dataSource elements specify that the given document shall be connected to the  
14 external data source referenced by the r:id attribute's value of rId1. While connected to the external data  
15 source, the merged document together with the hosting application and/or data source access application may  
16 extract data from the external data source as specified by the connectionString and query elements.

### 17 **2.15.3 Populating Merged Documents with External Data**

18 Before the hosting application can populate merged documents with external data, mail merge fields must be  
19 inserted into the *merged* document and mapped to the external data. How external data is mapped to given  
20 mail merge fields is determined by the WordprocessingML element fieldMapData.

1 Consider the example merged document from the previous example which contained the two mail merge  
 2 fields calling for **Courtesy Title** and **Last Name**. The WordprocessingML below demonstrates how  
 3 mapping of the external data to the merged document's mail merge fields occurs:

```

4   <w:fieldMapData>
5     <w:type w:val="dbColumn" />
6     <w:name w:val="Customer Title" />
7     <w:mappedName w:val="Courtesy Title" />
8     <w:column w:val="9" />
9   </w:fieldMapData>
10  <w:fieldMapData>
11    <w:type w:val="dbColumn" />
12    <w:name w:val="Customer Last Name" />
13    <w:mappedName w:val="Last Name" />
14    <w:column w:val="10" />
15  </w:fieldMapData>
```

16 Within the first fieldMapData element, the child elements column, name, type, and mappedName specify  
 17 that the data contained within tenth column titled 'Customer Title', in the specified external database, is to be  
 18 mapped to the mail merge field calling for 'Courtesy Title' data, respectively. Within the second fieldMapData  
 19 element, the child elements column, name, type, and mappedName specify that the data contained within  
 20 eleventh column in the specified external database is to be mapped to the merge field titled **Customer Last**  
 21 **Name** or the predefined merge field name **Last Name**.

22 Once a merged document's mail merge fields have been mapped to external data, the hosting application  
 23 and/or data source access application may populate the respective fields with applicable external data.

24 Consider a conforming hosting application and/or data source access application that wishes to populate the  
 25 mail merge fields within the merged document from the previous example with applicable external data. In  
 26 addition, consider that the specified external data source contains two records--one for Mr. John Doe and one  
 27 for Ms. Jane Smith. With external data from the **Customer Title** column mapped to the Mail Merge field  
 28 calling for **Courtesy Title** data, and the **Customer Last Name** column mapped to the Mail Merge field  
 29 calling for **Last Name** data to populate the fields within this merged document with external data.

30 The mail merge process will then run through the specified external database and populate the mail merge  
 31 fields with the data from the **Customer Title** and **Customer Last Name** columns in the specified  
 32 database, and generate two of merged documents containing the specified external data as well as the static  
 33 contents of the source document (illustrated in the table below):

Source Document	Merged document populated with first external data source entry	Merged document populated with second external data source entry
-----------------	---	--

Source Document	Merged document populated with first external data source entry	Merged document populated with second external data source entry
<p>Dear{MERGEFIELD "Courtesy Title"\m} {MERGEFIELD "Last Name"\m},</p> <p>Sample text. Sample text.</p> <p>Sincerely,</p>	<p>Dear Mr. Doe:</p> <p>Sample text. Sample text.</p> <p>Sincerely,</p>	<p>Dear Ms. Smith:</p> <p>Sample text. Sample text.</p> <p>Sincerely,</p>

## 2.16 Settings

A *setting* specifies a stored preference that shall be used when processing the contents of the document. In other words, settings refer to specified behaviors that shall be applied to WordprocessingML documents on a document by document basis. Just like paragraphs and text runs have properties specified that apply to their contents, entire WordprocessingML documents leverage settings to specify properties and behaviors that apply to the entire document.

These settings are typically divided into three categories:

- *Document Settings* — Settings that influence the appearance and behavior of the current document, as well as storing document-level state.
- *Compatibility Settings* — Settings that tell applications to perform behaviors which are designed to maintain visual output of previous word-processing applications.
- *Web Settings* — Settings that affect how a document shall be handled when it is saved as HTML.

### 2.16.1 Document Settings

A *document setting* specifies a document-level property that affects the handling of a given document, and influences the appearance and behavior of the current document, as well as the stored document-level state. All document settings are found in the Document Settings part.

Consider a document in which the document setting `doNotHyphenateCaps` is applied. As a document setting this element specifies whether words comprised of all capitalized letters shall be hyphenated or not throughout the given document.. Specifically, if words in ALL CAPITAL LETTERS shall not be hyphenated, this requirement would be specified by adding the following WordprocessingML to the settings part:

```
<w:doNotHyphenateCaps w:val="true"/>
```

Specifying that words comprised of ALL CAPITAL LETTERS shall be hyphenated, as illustrated below:

THIS IS A HYPHENATION EXAMPLE. THIS IS A SHORT HYPHENATION EXAMPLE. THIS IS A SHORT HYPHENATION EXAMPLE.

1

- 2 If this element is omitted, then words in ALL CAPITAL LETTERS shall be hyphenated when the document is  
 3 hyphenated, as illustrated below:

THIS IS A HYPHENATION EXAMPLE. THIS IS A SHORT HYPHENATION EXAMPLE. THIS IS A SHORT HYPHENATION EXAMPLE.

4

## 5 2.16.2 Compatibility Settings

6 A *compatibility setting* is an optional setting used to mimic behavior of documents created in earlier word-  
 7 processing applications. It is recommended that new WordprocessingML documents contain no compatibility  
 8 settings. If compatibility settings are needed, they are stored in the Document Settings part (§2.16.1).

9 Consider a document in which the compatibility setting `ww11IndentRules` is applied. As a compatibility  
 10 setting, this element specifies an indentation behavior to be applied throughout the given document to  
 11 preserve visual fidelity with an earlier word processing application. Specifically, if the indentation applied to  
 12 numbering when positioned next to a wrapped object shall not be suppressed, this requirement would be  
 13 specified by adding the following WordprocessingML to the settings part

```
14 <w:compat>
15   <w:ww11IndentRules />
16 </w:compat>
```

17 Specifying that indentation applied to numbering when positioned next to a wrapped object shall not be  
 18 suppressed, as illustrated below:

1. Example



2. Example  
 3. Example  
 4. Example  
 5. Example

19

20 If this element is omitted, then indentation applied to numbering when positioned next to a wrapped object  
 21 shall be suppressed, as illustrated below:

1. Example



2. Example  
 3. Example  
 4. Example  
 5. Example

22

### 1    2.16.3    Web Settings

- 2    A *web setting* is a setting used to specify a document-level property that is applicable when saving a web page  
 3    as a WordprocessingML document, or when saving a WordprocessingML document as a webpage. Thus, if a  
 4    given WordprocessingML document was not created from a web page, and will never become a web page, no  
 5    web settings are needed within the document. If they are needed, web settings are stored in the Web Settings  
 6    part.
- 7    Consider a document in which the web setting allowPNG is applied. As a web setting this element specifies if  
 8    the PNG graphics format will be used for persisting images when saving the document as a web page.  
 9    Specifically, if the PNG graphics format will be used when saving a document as a web page, this requirement  
 10   would be specified by adding the following WordprocessingML to the settings part:

```
11    <w:webSettings>
12      <w:allowPNG />
13    </w:webSettings>
```

- 14   If this element is omitted, then the JPEG graphics format will be used for persisting images when saving the  
 15   document as a web page.

## 16   2.17   Fields and Hyperlinks

### 17   2.17.1   Fields

- 18   Most text in a word processing document is static; that is, unless it is directly changed as the result of editing,  
 19   its contents remain the same, no matter how the rest of the document might change. However, certain useful  
 20   pieces of information can change value over the life of a document. Consider the case of a reference to a page  
 21   number, as in "For more information on this topic, see page 56." Clearly, hard coding the page number as 56  
 22   means that that number will need to be manually replaced as the document's size or layout is changed. Even a  
 23   simple change to any margin, line spacing, or font size can invalid such references.

- 24   Fields provide a mechanism for placeholders, such as page reference numbers, that can be added to a  
 25   document such that those placeholders are replaced by their corresponding values when the document is  
 26   rendered for display or print. Other applications for fields include, but are not limited to, automatic numbering  
 27   of tables and figures, document creation and current date and time, document author information, and the  
 28   computation of totals for a table column.

- 29   A *field* is a set of codes that instructs a WordprocessingML consumer to insert text, graphics, page numbers,  
 30   and other material into a document automatically. (The DATE field causes the current date to be inserted.) The  
 31   text or graphics inserted into a document when a consumer carries out a field's codes is referred to as the *field*  
 32   *result* for that field. The act of carrying out a field's codes is referred to as a *field update*. As to how or when  
 33   any field is updated is outside the scope of this standard.

## 1    2.17.2    Hyperlinks

2    As well as allowing for dynamic run content using fields, a WordprocessingML document may contain one or  
 3    more *hyperlinks*, which allow for the linking of two disparate regions of WordprocessingML content (analogous  
 4    to hyperlinks in HTML pages). WordprocessingML hyperlinks can be any of the following:

- 5       • Intradocument: A hyperlink can target any bookmark contained within the current WordprocessingML  
 6       document.
- 7       • Interdocument: A hyperlink can target another WordprocessingML package, as well as specify a  
 8       bookmark within that package.
- 9       • Other destinations: A hyperlink can target any other valid URI location.

## 10    2.18    Miscellaneous Topics

### 11    2.18.1    Text Boxes

12    All VML-based drawing objects (except for connectors) support the addition of rich WordprocessingML content  
 13    within their extents. When WordprocessingML contents have been added to a VML drawing object, the  
 14    resulting text is contained within a *text box*.

15    When WordprocessingML content is contained within a text box, it is represented within the object by  
 16    specifying the VML textbox element, which contains within it a single txbxContent element that contains all of  
 17    the desired WordprocessingML content. Text box content cannot contain references to other document  
 18    stories, nor can it contain other txbxContent elements. That is, nested shapes cannot have rich content.

### 19    2.18.2    Subdocuments

20    Within a WordprocessingML document, it is sometimes necessary to break a large document into two or more  
 21    separate WordprocessingML document files, allowing each of these files to be distributed, edited, and handled  
 22    independently.

23    A book might consist of five chapters, each edited by a separate author. The editor for the book would  
 24    therefore desire to create six WordprocessingML documents - one for each author to work on their chapter,  
 25    and a main document which collates the content of the five chapters appropriately.

26    When a WordprocessingML document is composed of other WordprocessingML documents in this way, the  
 27    resulting documents are a master document and its subdocuments.

- 28       • A *master document* is a document which incorporates one or more subdocuments (as well as optional  
 29       WordprocessingML content) to create a larger document
- 30       • A *subdocument* is a WordprocessingML document—there is no specific information in a document  
 31       which classifies it as such

32    Consider a WordprocessingML document, which is being used to write a book:

# My Book

---

Once upon a time...

## Chapter One

It was a cold, stormy night...

## Chapter Two

Still cold...

1

- 2 To allow this document to be written by multiple authors, each chapter in the book is placed in a separate file
- 3 (the sections highlighted in red below):

# My Book

---

Once upon a time...

## Chapter One

It was a cold, stormy night...

## Chapter Two

Still cold...

4

- 5 The result is three WordprocessingML documents:

- 6
  - A master document (containing the title of the book, the first paragraph, and references to the subdocuments for each chapter)
  - Two subdocuments (one for each chapter)

### 2.18.3 Importing External Content

- 10 When generating WordprocessingML documents, it is sometimes necessary to include existing document content (henceforth called *external content*) within the document. External content in a document is typically

1 included because it was stored in a format other than the WordprocessingML format defined by this Office  
 2 Open XML specification.

3 In order to facilitate the inclusion of such content without requiring its conversion as a prerequisite to its  
 4 inclusion in a document, WordprocessingML includes the facility for applications to implement the import of  
 5 external content in any format as part of a WordprocessingML document. This functionality, called *external*  
 6 *content import*, allows the inclusion of content of an arbitrary content type within the WordprocessingML  
 7 package, which can then be opened and merged into the main document when the package is consumed by  
 8 applications which understand that content type.

9 Consider a WordprocessingML document which is being created based on the following existing HTML  
 10 content:

```
11 <html ... >
12   <body style="margin-left:200px;margin-top:50px">
13     <p>Paragraph one.</p>
14     <blockquote style="border:5px solid #00FFFF">Paragraph in a
15       blockquote.</blockquote>
16     <p>Paragraph two.</p>
17   </body>
18 </html>
```

19 This content can be converted to its WordprocessingML equivalents using the XML syntax defined by this  
 20 Office Open XML specification, or a more basic tool can use the external content import to include the HTML  
 21 document within a WordprocessingML package, allowing a subsequent consumer of that content to import the  
 22 resulting HTML. When the resulting WordprocessingML package is opened, the HTML document it could be  
 23 read (if it is an alternate format understood by the consuming application) and migrated into the appropriate  
 24 location in the main WordprocessingML document.

#### 25 **2.18.4 Roundtripping Alternate Content**

26 Office Open XML defines a mechanism for the storage of content which is not defined by this Office Open XML  
 27 specification, for example extensions developed by future software applications which leverage the Open XML  
 28 formats. This mechanism allows for the storage of a series of alternative representations of content, of which  
 29 the consuming application may use the first alternative whose requirements are met.

30 Consider an application which creates a new paragraph property intended to make the colors of its text change  
 31 randomly when it is displayed. This functionality is not defined in this Office Open XML specification, and so  
 32 the application might choose to create an alternative representation setting a different manual color on each  
 33 character for clients which do not understand this extension using an *AlternateContent* block as follows:

```

1   <ve:AlternateContent xmlns:ve="...">
2     <ve:Choice Requires="colors" xmlns:colors="urn:randomTextColor">
3       <w:p>
4         <w:pPr>
5           <colors:random colors:val="true" />
6         </w:pPr>
7         <w:r>
8           <w:t>Random colors!</w:t>
9         </w:r>
10        </w:p>
11      </ve:Choice>
12      <ve:Fallback>
13        <w:p>
14          <w:r>
15            <w:rPr>
16              <w:color w:val="FF0000" />
17            </w:rPr>
18            <w:t>R</w:t>
19          </w:r>
20          <w:r>
21            <w:rPr>
22              <w:color w:val="00FF00" />
23            </w:rPr>
24            <w:t>a</w:t>
25          </w:r>
26          ...
27        </w:p>
28      </ve:Fallback>
29    </ve:AlternateContent>
```

30 The Choice element that requires the new color extensions uses the random element in its namespace, and  
 31 the Fallback element allows clients that do not support this namespace to see an appropriate alternative  
 32 representation.

33 These alternate content blocks may occur at any location within a WordprocessingML document, and  
 34 applications shall handle and process them appropriately (taking the appropriate choice).

35 However, WordprocessingML does not explicitly define a set of locations where applications shall attempt to  
 36 store and roundtrip all non-taken choices whenever possible.

37 If an application does not understand the colors extension, the resulting file (if alternate choices are to be  
 38 preserved would appear as follows:

```
1  <ve:AlternateContent xmlns:ve="...">
2      <ve:Choice Requires="colors" xmlns:colors="urn:randomTextColor">
3          ...
4      </ve:Choice>
5      <ve:Fallback>
6          ...
7      </ve:Fallback>
8  </ve:AlternateContent>
9
```

10 The file would then appear as follows after the choice is processed:

```
11 <w:p>
12     <w:r>
13         <w:rPr>
14             <w:color w:val="FF0000" />
15         </w:rPr>
16         <w:t>R</w:t>
17     </w:r>
18     <w:r>
19         <w:rPr>
20             <w:color w:val="00FF00" />
21         </w:rPr>
22         <w:t>a</w:t>
23     </w:r>
24     ...
25 </w:p>
```

26 **End of informative text.**

# 3. Introduction to SpreadsheetML

2 **This clause is informative.**

3 This clause contains a detailed introduction to the structure of a SpreadsheetML document.

## 4 **3.1 Workbook**

### 5 **3.1.1 Overview**

6 A *workbook* is composed of book-level properties and a collection of one or more *sheets*. The sheets are the  
 7 central working surface for a spreadsheet application. The workbook part and corresponding properties  
 8 comprise data used to set application- and workbook-level operational state. The workbook also serves to bind  
 9 all the sheets and child objects into an organized single file. The workbook properties include information  
 10 about what application last saved the file, where and how the windows of the workbook were positioned, and  
 11 an enumeration of the worksheets in the workbook.

### 12 **3.1.2 Minimum Workbook Scenario**

13 For the sake of simplicity, it is important to minimize the required set of workbook properties that must be  
 14 present to compose a valid workbook. The smallest possible (blank) workbook must contain the following:

- 15 • A single sheet
- 16 • A sheet ID
- 17 • A relationship Id that points to the location of the sheet definition

18 For example:

```
19 <workbook>
20   <sheets>
21     <sheet name="Sheet1" sheetId="1" r:id="rId1"/>
22   </sheets>
23 </workbook>
```

### 24 **3.1.3 Example Workbook Properties**

25 Consider the following graphical representation of a workbook:

The screenshot shows a Microsoft Excel spreadsheet titled "Workbook And Sheet Properties.xlsx". The formula bar at the top displays "=B2+1". The first two rows contain text: "External Link:" with a value of "1" and "Formula:" with a value of "2". Below these is a data table with columns labeled "Category", "Num1", "Num2", "Num3", and "Total". The data is as follows:

Category	Num1	Num2	Num3	Total
A	0.184607	0.934631	0.586478	1.705714922
A	0.504252	0.251189	0.269182	1.024622503
A	0.600602	0.183192	0.122543	0.906337605
A	0.78015	0.7816	0.067448	1.629198103
B	0.636081	0.356358	0.671221	1.663660406
B	0.333273	0.22565	0.579399	1.138321964

A large merged cell spans from C13 to H13, containing the text "Merged Cells".

- 1  
2  
3 The above example will have the following workbook properties definition:

```

4 <workbook>
5   <fileVersion lastEdited="4" lowestEdited="4" rupBuild="3814"/>
6   <workbookPr backupFile="1" saveExternalLinkValues="0" updateLinks="never"/>
7   <calcPr calcId="122211" calcMode="manual" iterate="1"/>
8   <bookViews>
9     <workbookView showHorizontalScroll="0" showVerticalScroll="0"
10    showSheetTabs="0" xWindow="45" yWindow="15" windowHeight="9420"
11    windowHeight="5460" tabRatio="701"/>
12   </bookViews>
13   <sheets>
14     <sheet name="Sheet1" sheetId="1" sh:id="rId1"/>
15     <sheet name="Sheet2" sheetId="2" sh:id="rId2"/>
16     <sheet name="Sheet3" sheetId="3" sh:id="rId3"/>
17   </sheets>
18 </workbook>

```

- 19 The elements and attributes used here are discussed in more detail in the following subclauses.

### 3.1.4 fileVersion

This contains file versioning properties, as follows.

- lastEdited – The version of the application that last saved the file.
- lowestEdited – The earliest version of the application that saved the file. This value is reset any time an application that understands all data in the file saves the file.
- rupBuild – An incremental public release of the application (e.g., RTM version or SP1 version).
- workbookPr – A group of various workbook properties.
- backupFile – A flag that indicates whether the application should create a backup of the file in question during a save operation.
- saveExternalLinkValues – A flag that indicates whether the application should cache values retrieved from other workbooks via an externally linking formula during save. If yes, a supporting part is written out containing a cached cell table from the external workbook.
- updateLinks – A flag that dictates how external links are handled upon opening the file. In this example, never means don't ask the user if they want to refresh the cached values from an external workbook, and in fact, don't ever do it until the user initiates the action.
- calcPr – Various calculation properties grouped together.
- calcId – The version of the calculation engine used to calculate values in the workbook. When a newer version of the application opens a file with an older calcId value, the application performs a full calculation of all formulas immediately after opening the workbook, to ensure proper calculation results.
- calcMode – A flag that indicates when the application should calculate formulas:
  - Manual means to wait for the user to initiate the action.
  - Automatic means to perform only the needed calculations whenever a cell value changes.
- Iterate – When formula references are circular (i.e., they refer back on themselves for required input), the iterate flag specifies that this is an intended and valid state. Further properties not discussed here control the number of iterative calculations to perform before stopping calculation.
- bookViews – A collection of views.

### 3.1.5 workbookView

A single view definition represented using the following flags:

- showHorizontalScroll – Controls visibility of the horizontal scroll bar of the application. In the example above, it is set to not being visible.
- showVerticalScroll – Controls visibility of the vertical scroll bar of the application. In the example above, it is set to not being visible.
- showSheetTabs – Controls visibility of the worksheet tabs in the application. In the example above, they are set to not being visible.
- xWindow – Specifies the x coordinate (in twips) of the upper right corner of the workbook window.
- yWindow – Specifies the y coordinate (in twips) of the upper right corner of the workbook window.

- 1     • windowWidth – Specifies the width of the workbook window.
- 2     • windowHeight – Specifies the height of the workbook window.
- 3     • tabRatio – Specifies the ratio between the workbook tabs bar and the horizontal scroll bar.
- 4     • Sheets – A collection of worksheets in the workbook.
- 5     • Sheet – A single sheet definition (book-level).
- 6     • Name – The name of the worksheet. These must be unique within the workbook.
- 7     • sheetId – The internal Id of the sheet. These must be unique within the workbook.
- 8     • Id - The relationship Id that points to the sheet part definition.

## 9     **3.2     Sheets**

10    Sheets are the central structures within a workbook, and are where a user does most of his spreadsheet work.  
 11    The most common type of sheet is the *worksheet*, which is represented as a grid of cells. Worksheet cells can  
 12    contain text, numbers, dates, and formulas. Cells can also be formatted. A workbook usually contains more  
 13    than one sheet. To aid in the analysis of data and the making of informed decisions, spreadsheet applications  
 14    often implement features and objects which help calculate, sort, filter, organize, and graphically display  
 15    information. Since these features are often connected very tightly with the spreadsheet grid, these are also  
 16    included in the sheet definition on disk.

17    Other types of sheets include *chart sheets* and *dialog sheets*.

### 18    **3.2.1     Minimum Worksheet Scenario**

19    The smallest possible (blank) sheet is as follows:

```
20       <worksheet>
21        <sheetData/>
22      </worksheet>
```

23    The empty sheetData collection represents an empty grid; this element is required. As defined in the schema,  
 24    some optional sheet property collections can appear before sheetData, and some can appear after. To simplify  
 25    the logic required to insert a new sheetData collection into an existing (but empty) sheet, the sheetData  
 26    collection is required, even when empty.

### 27    **3.2.2     Example Sheet**

28    Consider the following graphical representation of a worksheet:

	A	B	C	D	E	F	G	H
1								
2	External Link:	1						
3	Formula:		2					
4								
5								
7			A		0.184607	0.934631	0.586478	1.705714922
8			A		0.504252	0.251189	0.269182	1.024622503
9			A		0.600602	0.183192	0.122543	0.906337605
10			B		0.78015	0.7816	0.067448	1.629198103
11			B		0.636081	0.356358	0.671221	1.663660406
12			B		0.333273	0.22565	0.579399	1.138321964
13								
14								
15								

1

2

3 Notice that cells A2 and A3 contain text. Cell B1 contains a formula linking to another workbook, whose value  
 4 is 1. Cell B2 contains a formula as well; this formula appears in the formula bar (top of picture) because it is the  
 5 active cell. Cells D5:H5 contain bold-faced text that serves as headers for the table of data residing in D6:H11.  
 6 The table of data has a filter feature applied to it (evidenced by drop down arrows in the header row), and  
 7 columns G and H have different types of conditional formatting applied. Finally, cells D13:H14 are part of a  
 8 merged cell feature, where a series of cells behave together as a single, larger cell.

9 When saved, the above example will have the syntax below written out in the corresponding sheet part. Sheet  
 10 information is organized into three main sections:

- 11     1. Top-level sheet properties (everything before sheetData)  
 12     2. The cell table (sheetData)  
 13     3. Supporting sheet features (everything after sheetData)

14 Therefore, the XML for the above example would look like this, broken into three sections:

### 15     3.2.3     Sheet Properties

```
16 <worksheet>
17   <sheetPr filterMode="1"/>
18   <dimension ref="A2:H14"/>
```

```

1   <sheetViews>
2     <sheetView tabSelected="1" workbookViewId="0">
3       <selection activeCell="B3" sqref="B3"/>
4     </sheetView>
5   </sheetViews>
6   <sheetFormatPr defaultRowHeight="15"/>
7   <cols>
8     <col min="1" max="1" width="12.85546875" bestFit="1" customWidth="1"/>
9     <col min="3" max="3" width="3.28515625" customWidth="1"/>
10    <col min="4" max="4" width="11.140625" bestFit="1" customWidth="1"/>
11    <col min="8" max="8" width="17.140625" style="1" customWidth="1"/>
12  </cols>

```

### 3.2.4 Sheet Data

sheetData, which represents the cell table, directly after the cols collection:

```

15  <sheetData>
16    <row r="2" spans="1:2" customFormat="1">
17      <c r="A2" s="1" t="s">
18        <v>0</v>
19      </c>
20      <c r="B2">
21        <f>[1]Sheet1!$A$1</f>
22        <v>1</v>
23      </c>
24    </row>
25    <row r="3" spans="1:8" customFormat="1">
26      <c r="A3" s="1" t="s">
27        <v>1</v>
28      </c>
29      <c r="B3">
30        <f>B2+1</f>
31        <v>2</v>
32      </c>
33      <c r="H3" s="1"/>
34    </row>
35    <row r="4" spans="1:8">
36      <c r="H4"/>
37    </row>
38    <row r="5" spans="4:8">
39      <c r="D5" s="1" t="s">
40        <v>4</v>
41      </c>

```

```

1   <c r="E5" s="1" t="s">
2     <v>5</v>
3   </c>
4   <c r="F5" s="1" t="s">
5     <v>6</v>
6   </c>
7   <c r="G5" s="1" t="s">
8     <v>7</v>
9   </c>
10  <c r="H5" s="1" t="s">
11    <v>8</v>
12  </c>
13 </row>
14 <row r="6" spans="4:8">
15   <c r="D6" t="s">
16     <v>2</v>
17   </c>
18   <c r="E6">
19     <v>0.18460660235998017</v>
20   </c>
21   <c r="F6">
22     <v>0.93463071023892952</v>
23   </c>
24   <c r="G6">
25     <v>0.58647760893211043</v>
26   </c>
27   <c r="H6" s="1">
28     <f ce="1">SUM(E6:G6)</f>
29     <v>1.7057149215310201</v>
30   </c>
31 </row>
32 <row r="7" spans="4:8">
33   <c r="D7" t="s">
34     <v>2</v>
35   </c>
36   <c r="E7">
37     <v>0.50425224796279555</v>
38   </c>
39   <c r="F7">
40     <v>0.25118866081991786</v>
41   </c>

```

```

1   <c r="G7">
2     <v>0.26918159410869791</v>
3   </c>
4   <c r="H7" s="1">
5     <f t="shared" ref="H7:H11" ce="1" si="0">SUM(E7:G7)</f>
6     <v>1.0246225028914113</v>
7   </c>
8 </row>
9 <row r="8" spans="4:8">
10  <c r="D8" t="s">
11    <v>2</v>
12  </c>
13  <c r="E8">
14    <v>0.6006019062877066</v>
15  </c>
16  <c r="F8">
17    <v>0.18319235857964333</v>
18  </c>
19  <c r="G8">
20    <v>0.12254334000604317</v>
21  </c>
22  <c r="H8" s="1">
23    <f t="shared" ce="1" si="0">SUM(E8:G8)</f>
24    <v>0.9063376048733931</v>
25  </c>
26 </row>
27 <row r="9" spans="4:8" hidden="1">
28  <c r="D9" t="s">
29    <v>2</v>
30  </c>
31  <c r="E9">
32    <v>0.78015011938458589</v>
33  </c>
34  <c r="F9">
35    <v>0.78159963723670689</v>
36  </c>
37  <c r="G9">
38    <v>6.7448346870105036E-2</v>
39  </c>

```

```

1   <c r="H9" s="1">
2     <f t="shared" ce="1" si="0">SUM(E9:G9)</f>
3       <v>1.6291981034913978</v>
4   </c>
5 </row>
6 <row r="10" spans="4:8" hidden="1">
7   <c r="D10" t="s">
8     <v>3</v>
9   </c>
10  <c r="E10">
11    <v>0.63608141933645479</v>
12  </c>
13  <c r="F10">
14    <v>0.35635845012920608</v>
15  </c>
16  <c r="G10">
17    <v>0.67122053637107193</v>
18  </c>
19  <c r="H10" s="1">
20    <f t="shared" ce="1" si="0">SUM(E10:G10)</f>
21    <v>1.6636604058367328</v>
22  </c>
23 </row>
24 <row r="11" spans="4:8" hidden="1">
25   <c r="D11" t="s">
26     <v>3</v>
27   </c>
28   <c r="E11">
29     <v>0.33327331908137214</v>
30   </c>
31   <c r="F11">
32     <v>0.2256497329592122</v>
33   </c>
34   <c r="G11">
35     <v>0.5793989116090501</v>
36   </c>
37   <c r="H11" s="1">
38     <f t="shared" ce="1" si="0">SUM(E11:G11)</f>
39     <v>1.1383219636496344</v>
40   </c>
41 </row>

```

```

1      <row r="13" spans="4:8">
2          <c r="D13" s="2" t="s">
3              <v>9</v>
4          </c>
5          <c r="E13" s="3"/>
6          <c r="F13" s="3"/>
7          <c r="G13" s="3"/>
8          <c r="H13" s="4"/>
9      </row>
10     <row r="14" spans="4:8">
11         <c r="D14" s="5"/>
12         <c r="E14" s="6"/>
13         <c r="F14" s="6"/>
14         <c r="G14" s="6"/>
15         <c r="H14" s="7"/>
16     </row>
17     </sheetData>

```

### 3.2.5 Supporting Features

The supporting feature definitions follow the cell table data:

```

20     <sheetProtection objects="0" scenarios="0"/>
21     <autoFilter ref="D5:H11">
22         <filterColumn colId="0">
23             <filters>
24                 <filter val="A"/>
25             </filters>
26         </filterColumn>
27         <filterColumn colId="1">
28             <customFilters and="1">
29                 <customFilter operator="greaterThan" val="0"/>
30                 <customFilter operator="lessThan" val="0.7"/>
31             </customFilters>
32         </filterColumn>
33     </autoFilter>
34     <mergeCells>
35         <mergeCell ref="D13:H14"/>
36     </mergeCells>
37     <conditionalFormatting sqref="H6:H11">
38         <cfRule type="dataBar" priority="3" stopIfTrue="0">
39             <formula>MAX(IF(ISBLANK(H6:H11), "", IF(ISERROR(H6:H11), "", H6:H11)))</formula>
40             <formula>MIN(IF(ISBLANK(H6:H11), "", IF(ISERROR(H6:H11), "", H6:H11)))</formula>
41
42

```

```

1   <dataBar minLength="10" maxLength="90" showValue="1">
2     <cfvo type="min" val="0"/>
3     <cfvo type="max" val="0"/>
4     <color rgb="FF63C384"/>
5   </dataBar>
6 </cfRule>
7 </conditionalFormatting>
8 <conditionalFormatting sqref="G6:G11">
9   <cfRule type="cellIs" dxfId="0" priority="1" stopIfTrue="0"
10    operator="greaterThan">
11    <formula>0.5</formula>
12  </cfRule>
13 </conditionalFormatting>
14 <printOptions/>
15 <pageMargins left="0.7" right="0.7" top="0.75" bottom="0.75"
16   header="0.3" footer="0.3"/>
17 <pageSetup orientation="portrait" horizontalDpi="300" verticalDpi="300"/>
18 <headerFooter/>
```

19 These elements are discussed in more detail in the following subclauses.

### 20 **3.2.6 Sheet Properties**

21 Referring back to §3.2.3, note that several sheet-level properties are expressed before the sheetData cell table  
22 is encountered.

23 sheetPr indicates that an AutoFilter has been applied on this sheet. Dimension indicates the used range on  
24 this sheet. There should be no data or formulas outside this range. The sheetViews collection indicates which  
25 cell is active on the sheet, and indicates whether this particular sheet is the active sheet in the workbook.

26 A collection of column-level settings appears in the cols collection.

27 Finally, within sheetFormatPr, a default row height is set.

### 28 **3.2.7 sheetData Cell Table**

29 The cell table is the core structure of a worksheet. It consists of all the text, numbers, and formulas in the grid.

### 30 **3.2.8 Row**

```

31 <row r="2" spans="1:2" customFormat="1">
32   <c r="A2" s="1" t="s">
33     <v>0</v>
34   </c>
```

```

1      <c r="B2">
2          <f>[1]Sheet1!$A$1</f>
3          <v>1</v>
4      </c>
5  </row>
```

6 The cells in the cell table are organized by row. Each row has an index (attribute r) so that empty rows need  
7 not be written out. Each row indicates the number of cells defined for it, as well as their relative position in the  
8 sheet. In this example, the first row of data is row 2.

### 9 3.2.9 Cell

```

10     <c r="B3">
11         <f>B2+1</f>
12         <v>2</v>
13     </c>
```

14 The cell itself is expressed by the c collection. Each cell indicates its location in the grid using A1-style  
15 reference notation. A cell can also indicate a style identifier (attribute s) and a data type (attribute t). The cell  
16 types include string, number, and Boolean. In order to optimize load/save operations, default data values are  
17 not written out.

#### 18 3.2.9.1 Cell Values

19 Cells contain values, whether the values were directly typed in (e.g., cell A2 in our example has the value  
20 External Link:) or are the result of a calculation (e.g., cell B3 in our example has the formula B2+1).

21 String values in a cell are not stored in the cell table unless they are the result of a calculation. Therefore,  
22 instead of seeing External Link: as the content of the cell's v node, instead you see a zero-based index  
23 into the shared string table where that string is stored uniquely. This is done to optimize load/save  
24 performance and to reduce duplication of information. To determine whether the 0 in v is a number or an  
25 index to a string, the cell's data type must be examined. When the data type indicates string, then it is an index  
26 and not a numeric value.

#### 27 3.2.9.2 Formulas

28 Cells can contain formulas, which calculate results. Formulas are expressed in the file the same way the user  
29 sees them at runtime of the application. This is specifically a design choice meant to aid in creation and  
30 processing of workbook contents.

31 A formula can have attributes on it indicating how to handle calculation of the cell.

1    3.2.9.2.1                Shared Formulas

```

2    <row r="7" spans="4:8">
3     <c r="H7" s="1">
4       <f t="shared" ref="H7:H11" ce="1" si="0">SUM(E7:G7)</f>
5       <v>1.0246225028914113</v>
6     </c>
7   </row>
8   <row r="8" spans="4:8">
9     <c r="H8" s="1">
10      <f t="shared" ce="1" si="0">SUM(E8:G8)</f>
11      <v>0.9063376048733931</v>
12     </c>
13 </row>
```

14   Just as strings in cells can be extremely pervasive and redundant in a sheet (and therefore must be optimized),  
15   formulas are also extremely pervasive in a sheet, and often can be optimized. Consider the table in the above  
16   example, where column H contains a formula that sums the numbers in columns E through G, for each row.  
17   The only difference between the formulas in H6:H12 is that the reference increases by 1 row from one row to  
18   the next. Therefore, an optimization is created where only the formula in H6 needs to be written out, with  
19   some additional information indicating how far to propagate the formula once loaded. This enables the loading  
20   application to load and parse only the first of the shared formulas, and then more quickly apply the necessary  
21   transforms to produce the additional related formulas in subsequent cells.

22   Note that while formulas can be shared, it is desirable to enable easy access to the contents of a cell.  
23   Therefore, it is allowed that all formulas may be written out, but only the primary formula in a shared formula  
24   need be loaded and parsed.

25   3.2.9.2.2                External Referencing Formulas

```

26   <c r="B2">
27     <f>[1]Sheet1!$A$1</f>
28     <v>1</v>
29   </c>
```

30   In the above example, cell B2 contains a formula that references a cell in another workbook, namely  
31   ‘C:\[ExternalBook.xlsx]Sheet1’ !\$A\$1. This formula is referencing ExternalBook.xlsx located  
32   at c:\. Furthermore, the formula is requesting the value of cell A1 on Sheet1 of that particular workbook.

33   Instead of writing ‘C:\[ExternalBook.xlsx]Sheet1’ !\$A\$1 directly in the formula, it is desirable to  
34   make all external references much more accessible, especially given the potentially enormous size of a cell  
35   table. Therefore, the URL and file location is persisted using the relationships semantic, in a relationship file,  
36   and then referenced inline with the formula: [1]Sheet1!\$A\$1. In this way, external resource files can more  
37   easily be determined and updated if needed.

1 Note that whenever a workbook contains a formula referencing another workbook, some values from that  
 2 external workbook are also cached with the referencing workbook. This is done so that if a recalculation of the  
 3 workbook is needed and the workbook isn't accessible, a cached value may be used to complete the  
 4 calculation.

### 5 **3.2.10 Supporting Sheet Features**

### 6 **3.2.11 Defined Names**

```
7 <definedNames>
8   <definedName name="FMLA">Sheet1!$B$3</definedName>
9   <definedName name="SheetLevelName" comment="This name is scoped to Sheet1"
10    localSheetId="0">Sheet1!$B$3</definedName>
11 </definedNames>
```

12 Defined names can be used in place of cell references in formulas. For example, instead of using B3+1 to add 1  
 13 to the value that's in B3, one could define a name, as in FMLA, and assign it to B3. Then FMLA+1 can be used to  
 14 perform the calculation.

15 Names can be defined and assigned to a cell location or range or to a formula or constant value. Names can be  
 16 referenced in formulas. Names can be scoped to either the entire workbook (default) or just the local sheet.  
 17 Names scoped to the local sheet cannot be referenced from other sheets. Names scoped to the workbook can  
 18 be referenced from any sheet.

19 Defined names are actually stored in the workbook part, but are discussed here in the context of the sheet  
 20 because they are so closely related to cells and formulas.

### 21 **3.2.12 AutoFilter**

```
22 <autoFilter ref="D5:H11">
23   <filterColumn colId="0">
24     <filters>
25       <filter val="A"/>
26     </filters>
27   </filterColumn>
28   <filterColumn colId="1">
29     <customFilters and="1">
30       <customFilter operator="greaterThan" val="0"/>
31       <customFilter operator="lessThan" val="0.7"/>
32     </customFilters>
33   </filterColumn>
34 </autoFilter>
```

35 *AutoFilters* specify criteria for which cells in a table should be displayed. In this example, the first column (zero-  
 36 based index colId) in the table (cells D5:D11), has a criteria specifying that only rows in the table whose value  
 37 in column D are equal to A will be shown. The rest of the rows are *hidden*.

- 1 A second criterion is specified as well, on the 2nd column, E: only rows whose values in column E are greater than 0 and less than 0.7.
- 2
- 3 The resulting grid could be rendered like this:

The screenshot shows a Microsoft Excel spreadsheet titled "Workbook And Sheet Properties.xlsx". The formula bar at the top displays "=B2+1". The spreadsheet contains several rows of data and a table starting from row 5. Row 5 has columns labeled Category, Num1, Num2, Num3, and Total. Rows 6, 7, and 8 show data for category 'A'. Row 13 is merged with row 14, both spanning from column D to H and containing the text "Merged Cells".

	A	B	C	D	E	F	G	H
1								
2	External Link:	1						
3	Formula:	2						
4								
5		Category	Num1	Num2	Num3	Total		
6		A	0.184607	0.934631	0.586478	1.705714922		
7		A	0.504252	0.251189	0.269182	1.024622503		
8		A	0.600602	0.183192	0.122543	0.906337605		
12								
13	<b>Merged Cells</b>							
14								
15								
16								
17								
18								
19								

### 3.2.13 Merged Cells

```

6   <mergeCells>
7     <mergeCell ref="D13:H14"/>
8   </mergeCells>

```

In the example, cells D13:H14 have been merged into a single, larger cell. Note that the cell table itself doesn't reflect this merge, but it does reflect the data content and formatting. Specifically, the top-left cell in a merged collection of cells contains the value, and all the cells reflect the various border formatting.

### 3.2.14 Conditional Formatting

```

13   <conditionalFormatting sqref="H6:H11">
14     <cfRule type="dataBar" priority="3" stopIfTrue="0">
15       <formula>MAX(IF(ISBLANK(H6:H11), "", IF(ISERROR(H6:H11), "", H6:H11)))</formula>
16       <formula>MIN(IF(ISBLANK(H6:H11), "", IF(ISERROR(H6:H11), "", H6:H11)))</formula>
17     </cfRule>
18   </conditionalFormatting>

```

```

1   <dataBar minLength="10" maxLength="90" showValue="1">
2     <cfvo type="min" val="0"/>
3     <cfvo type="max" val="0"/>
4     <color rgb="FF63C384"/>
5   </dataBar>
6   </cfRule>
7 </conditionalFormatting>
8 <conditionalFormatting sqref="G6:G11">
9   <cfRule type="cellIs" dxId="0" priority="1" stopIfTrue="0"
10 operator="greaterThan">
11     <formula>0.5</formula>
12   </cfRule>
13 </conditionalFormatting>
```

14 There are two conditional formats applied: one to the table of data in column H and the other to the table of  
15 data in column G.

16 In column G, a red fill is applied to any cell whose value is greater than 0.5. Notice that sqref specifies the  
17 range to which the rule applies. The formatting is specified by dxId, which is a reference to a formatting  
18 expression in the central styles part.

19 In column H there is a dataBar formatting rule, which applies a variable length bar to the cell background,  
20 where the length of the bar depends on the relative value of the cell.

## 21 **3.3 Shared String Table**

### 22 **3.3.1 Overview**

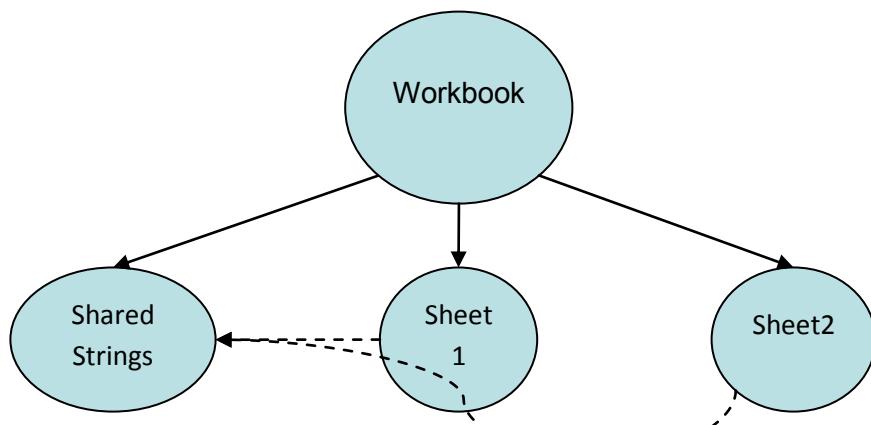
23 A workbook may contain thousands of cells containing string (non-numeric) data. Furthermore, this data is  
24 very likely to be repeated across many rows or columns. The goal of implementing a single string table that is  
25 shared across the workbook is to improve performance in opening and saving the file by only reading and  
26 writing the repetitive information once.

27 For example, consider a workbook summarizing information for cities within various countries. There may be a  
28 column for the name of the country, a column for the name of each city in that country, and a column  
29 containing the data for each city:

	A	B	C	D
1				
2		<b>Country</b>	<b>City</b>	<b>Data</b>
3		United States	Seattle	469.83
4		United States	Denver	36.16
5		United States	New York	114.38
6		United States	Philadelphia	540.95
7		United States	Houton	649.07
8		United States	San Diego	258.4
9		United States	San Francisco	686.05
10		Argentina	Buenos Aires	14.08
11		Argentina	San Juan	28.17
12		Argentina	Salta	757.31
13		Argentina	Rosario	246.23
14		Argentina	La Plata	947.63
15		Japan	Tokyo	597.55
16		Japan	Nagoya	619.01
17		Japan	Yokohama	525.14
18		Japan	Sendai	611.74
19				

- 2 In this case, the country name is repetitive, being duplicated in many cells. In many cases, the repetition is  
 3 extensive, and a tremendous savings is realized by making use of a shared string table when saving the  
 4 workbook.

### 3.3.2 File Architecture



- 6  
 7  
 8 There is a single shared strings part for all the strings in a workbook. This part is related to the workbook. Each  
 9 cell (in sheet1.xml, for example) containing a string value refers by index to a string expressed in the shared

1 strings part. The solid arrows represent relationships among the parts and the dotted arrows represent  
 2 references by index to a string in the shared strings part.

### 3 3.3.3 Example: Plain Text

4 This first example demonstrates plain text in cells. Note that in this example, some of the cells are formatted  
 5 (e.g., the column headers "Country", "City", and "Data" are bold faced). Since the formatting is applied at the  
 6 cell level, cell styles and formatting is used to describe the formatting, rather than using text formatting on the  
 7 text itself.

8 A later example demonstrates how to handle a variety of text formatting (rich text) within a single cell.

### 9 3.3.4 Illustration

10 Consider the example in the introduction:

	A	B	C	D
1				
2	Country	City	Data	
3	United States	Seattle	469.83	
4	United States	Denver	36.16	
5	United States	New York	114.38	
6	United States	Philadelphia	540.95	
7	United States	Houton	649.07	
8	United States	San Diego	258.4	
9	United States	San Francisco	686.05	
10	Argentina	Buenos Aires	14.08	
11	Argentina	San Juan	28.17	
12	Argentina	Salta	757.31	
13	Argentina	Rosario	246.23	
14	Argentina	La Plata	947.63	
15	Japan	Tokyo	597.55	
16	Japan	Nagoya	619.01	
17	Japan	Yokohama	525.14	
18	Japan	Sendai	611.74	
19				

12 In this example, the country names in the column titled 'Country'—"United States", "Argentina", and "Japan"—  
 13 would appear a single time in the shared strings part. Additionally, all the city names (in the column titled  
 14 'City') would appear a single time in the shared strings part, as would the column titles themselves in B2:D2.  
 15 The numeric values in the 'Data' column would be expressed inline with the cell table definition (e.g., in  
 16 Sheet1.xml).

### 17 3.3.5 The XML

18 The shared string table XML for this example looks like this:

```
1   <sst ... count="35" uniqueCount="22">
2     <si>
3       <t>United States</t>
4     </si>
5     <si>
6       <t>Seattle</t>
7     </si>
8     <si>
9       <t>Denver</t>
10    </si>
11    <si>
12      <t>New York</t>
13    </si>
14    <si>
15      <t>Philadelphia</t>
16    </si>
17    <si>
18      <t>Houton</t>
19    </si>
20    <si>
21      <t>San Diego</t>
22    </si>
23    <si>
24      <t>San Francisco</t>
25    </si>
26    <si>
27      <t>Argentina</t>
28    </si>
29    <si>
30      <t>Buenos Aires</t>
31    </si>
32    <si>
33      <t>San Juan</t>
34    </si>
35    <si>
36      <t>Salta</t>
37    </si>
38    <si>
39      <t>Rosario</t>
40    </si>
41    <si>
42      <t>La Plata</t>
43    </si>
```

```

1   <si>
2     <t>Japan</t>
3   </si>
4   <si>
5     <t>Tokyo</t>
6   </si>
7   <si>
8     <t>Nagoya</t>
9   </si>
10  <si>
11    <t>Yokohama</t>
12  </si>
13  <si>
14    <t>Sendai</t>
15  </si>
16  <si>
17    <t>Country</t>
18  </si>
19  <si>
20    <t>City</t>
21  </si>
22  <si>
23    <t>Data</t>
24  </si>
25  </sst>

```

26 The cell table for this example looks like this:

```

27  <sheetData>
28    <row r="2" spans="2:4" customFormat="1">
29      <c r="B2" s="1" t="s">
30        <v>19</v>
31      </c>
32      <c r="C2" s="1" t="s">
33        <v>20</v>
34      </c>
35      <c r="D2" s="1" t="s">
36        <v>21</v>
37      </c>
38    </row>
39    <row r="3" spans="2:4" customFormat="1">
40      <c r="B3" t="s">
41        <v>0</v>
42      </c>

```

```
1  <c r="C3" t="s">
2      <v>1</v>
3  </c>
4  <c r="D3">
5      <f t="shared" ref="D3:D18" ca="1" si="0">ROUND(RAND()*1000,2)</f>
6      <v>374.9</v>
7  </c>
8 </row>
9 <row r="4" spans="2:4" customFormat="1">
10     <c r="B4" t="s">
11         <v>0</v>
12     </c>
13     <c r="C4" t="s">
14         <v>2</v>
15     </c>
16     <c r="D4">
17         <f t="shared" ca="1" si="0"/>
18         <v>452.82</v>
19     </c>
20 </row>
21 <row r="5" spans="2:4" customFormat="1">
22     <c r="B5" t="s">
23         <v>0</v>
24     </c>
25     <c r="C5" t="s">
26         <v>3</v>
27     </c>
28     <c r="D5">
29         <f t="shared" ca="1" si="0"/>
30         <v>632.1</v>
31     </c>
32 </row>
33 <row r="6" spans="2:4" customFormat="1">
34     <c r="B6" t="s">
35         <v>0</v>
36     </c>
37     <c r="C6" t="s">
38         <v>4</v>
39     </c>
```

```

1   <c r="D6">
2     <f t="shared" ca="1" si="0"/>
3       <v>886.37</v>
4   </c>
5 </row>
6 <row r="7" spans="2:4" customFormat="1">
7   <c r="B7" t="s">
8     <v>0</v>
9   </c>
10  <c r="C7" t="s">
11    <v>5</v>
12  </c>
13  <c r="D7">
14    <f t="shared" ca="1" si="0"/>
15    <v>291.14</v>
16  </c>
17 </row>
18 <row r="8" spans="2:4" customFormat="1">
19   <c r="B8" t="s">
20     <v>0</v>
21   </c>
22   <c r="C8" t="s">
23     <v>6</v>
24   </c>
25   <c r="D8">
26     <f t="shared" ca="1" si="0"/>
27     <v>114.97</v>
28   </c>
29 </row>
30 <row r="9" spans="2:4" customFormat="1">
31   <c r="B9" t="s">
32     <v>0</v>
33   </c>
34   <c r="C9" t="s">
35     <v>7</v>
36   </c>
37   <c r="D9">
38     <f t="shared" ca="1" si="0"/>
39     <v>291.99</v>
40   </c>
41 </row>

```

```

1   <row r="10" spans="2:4" customFormat="1">
2     <c r="B10" t="s">
3       <v>8</v>
4     </c>
5     <c r="C10" t="s">
6       <v>9</v>
7     </c>
8     <c r="D10">
9       <f t="shared" ca="1" si="0"/>
10      <v>335.42</v>
11    </c>
12  </row>
13  <row r="11" spans="2:4" customFormat="1">
14    <c r="B11" t="s">
15      <v>8</v>
16    </c>
17    <c r="C11" t="s">
18      <v>10</v>
19    </c>
20    <c r="D11">
21      <f t="shared" ca="1" si="0"/>
22      <v>664.72</v>
23    </c>
24  </row>
25  <row r="12" spans="2:4" customFormat="1">
26    <c r="B12" t="s">
27      <v>8</v>
28    </c>
29    <c r="C12" t="s">
30      <v>11</v>
31    </c>
32    <c r="D12">
33      <f t="shared" ca="1" si="0"/>
34      <v>992.62</v>
35    </c>
36  </row>
37  <row r="13" spans="2:4" customFormat="1">
38    <c r="B13" t="s">
39      <v>8</v>
40    </c>
41    <c r="C13" t="s">
42      <v>12</v>
43    </c>

```

```
1 <c r="D13">
2   <f t="shared" ca="1" si="0"/>
3     <v>148.5</v>
4   </c>
5 </row>
6 <row r="14" spans="2:4" customFormat="1">
7   <c r="B14" t="s">
8     <v>8</v>
9   </c>
10  <c r="C14" t="s">
11    <v>13</v>
12  </c>
13  <c r="D14">
14    <f t="shared" ca="1" si="0"/>
15    <v>193.53</v>
16  </c>
17 </row>
18 <row r="15" spans="2:4" customFormat="1">
19   <c r="B15" t="s">
20     <v>14</v>
21   </c>
22   <c r="C15" t="s">
23     <v>15</v>
24   </c>
25   <c r="D15">
26     <f t="shared" ca="1" si="0"/>
27     <v>849.36</v>
28   </c>
29 </row>
30 <row r="16" spans="2:4" customFormat="1">
31   <c r="B16" t="s">
32     <v>14</v>
33   </c>
34   <c r="C16" t="s">
35     <v>16</v>
36   </c>
37   <c r="D16">
38     <f t="shared" ca="1" si="0"/>
39     <v>765.46</v>
40   </c>
41 </row>
```

```

1   <row r="17" spans="2:4" customFormat="1">
2     <c r="B17" t="s">
3       <v>14</v>
4     </c>
5     <c r="C17" t="s">
6       <v>17</v>
7     </c>
8     <c r="D17">
9       <f t="shared" ca="1" si="0"/>
10      <v>350.26</v>
11    </c>
12  </row>
13  <row r="18" spans="2:4" customFormat="1">
14    <c r="B18" t="s">
15      <v>14</v>
16    </c>
17    <c r="C18" t="s">
18      <v>18</v>
19    </c>
20    <c r="D18">
21      <f t="shared" ca="1" si="0"/>
22      <v>979.22</v>
23    </c>
24  </row>
25 </sheetData>
```

### 26      3.3.6      Shared String Table

```

27  <sst ... count="35" uniqueCount="22">
28    <si>
29      <t>United States</t>
30    </si>
31    <si>
32      <t>Seattle</t>
33    </si>
34    <si>
35      <t>Denver</t>
36    </si>
```

37 Examining the XML for the shared string part, it can be found that the first entry in the string table is "United  
38 States", residing in position 0. The value "Seattle" can be found in position 1 and "Denver" can be found in  
39 position 2.

### 3.3.7 Cell Table

```

1   <row r="2" spans="2:4" customFormat="1">
2     <c r="B2" s="1" t="s">
3       <v>19</v>
4     </c>
5     <c r="C2" s="1" t="s">
6       <v>20</v>
7     </c>
8     <c r="D2" s="1" t="s">
9       <v>21</v>
10    </c>
11  </row>
```

13 The first cell in our spreadsheet that contains data is B2. The XML indicates that it is of type 'string' (`t="s"`).  
 14 This indicates that the numeric value found inside the `<v>` element is an index to a string in the string table  
 15 rather than an actual number in the spreadsheet. The value for cell B2 is '19'. The 19th entry in the shared  
 16 string table (counting the first entry as 0) has a value of "Country". Therefore, cell B2 contains the word  
 17 "Country".

```

18   <row r="3" spans="2:4" customFormat="1">
19     <c r="B3" t="s">
20       <v>0</v>
21     </c>
22     <c r="C3" t="s">
23       <v>1</v>
24     </c>
25     <c r="D3">
26       <f t="shared" ref="D3:D18" ca="1" si="0">ROUND(RAND()*1000,2)</f>
27       <v>374.9</v>
28     </c>
29  </row>
```

30 Cell B3 (`<c @r="B3" ...>`) is also of type string, and the '0' inside the `v` element refers to the 0th item in the  
 31 string table, which corresponds to the string value "United States". Cell C3 is a string type of cell and  
 32 references the shared string found in position 1 in the string table, corresponding to the value "Seattle".  
 33 Cell D3 contains an `f` element, indicating a formula.

```

34   <row r="4" spans="2:4" customFormat="1">
35     <c r="B4" t="s">
36       <v>0</v>
37     </c>
38     <c r="C4" t="s">
39       <v>2</v>
40     </c>
```

```

1   <c r="D4">
2     <f t="shared" ca="1" si="0"/>
3     <v>452.82</v>
4   </c>
5 </row>
```

6 Examining the cell table entries for the data in row 4 of the spreadsheet, we see that cell B4 also contains the  
7 string value "United States". This is the 2nd occurrence of the value "United States" in this example. Since this  
8 value only occurs once in the string table, again the cell is using an index of 0 to reference the string item in the  
9 string table. Cell C4 is of type string and an index value of '2' indicates that "Denver" is the value of this cell.

### 10 3.3.8 Example: Rich Text

11 In this example, a single string cell value has multiple types of text formatting applied to various parts of the  
12 text.

### 13 3.3.9 Illustration

	F
	This string has a variety of <u>formatting</u> applied

### 14 15 3.3.10 Shared String Table

16 The main difference between plain text and rich text is seen in the string table itself. The si element is capable  
17 of containing rich text expressions:

```

18 <si>
19   <r>
20     <t xml:space="preserve">This </t>
21   </r>
22   <r>
23     <rPr>
24       <b/>
25       <sz val="11"/>
26       <color theme="1"/>
27       <rFont val="Calibri"/>
28       <family val="2"/>
29       <scheme val="minor"/>
30     </rPr>
31     <t xml:space="preserve">string </t>
32   </r>
```

```
1   <r>
2     <rPr>
3       <sz val="11"/>
4       <color rgb="FFFF0000"/>
5       <rFont val="Calibri"/>
6       <family val="2"/>
7       <scheme val="minor"/>
8     </rPr>
9     <t>has</t>
10   </r>
11   <r>
12     <rPr>
13       <sz val="11"/>
14       <color theme="1"/>
15       <rFont val="Calibri"/>
16       <family val="2"/>
17       <scheme val="minor"/>
18     </rPr>
19     <t xml:space="preserve"> a </t>
20   </r>
21   <r>
22     <rPr>
23       <i/>
24       <sz val="11"/>
25       <color rgb="FF00B050"/>
26       <rFont val="Calibri"/>
27       <family val="2"/>
28       <scheme val="minor"/>
29     </rPr>
30     <t>variety</t>
31   </r>
32   <r>
33     <rPr>
34       <sz val="11"/>
35       <color theme="1"/>
36       <rFont val="Calibri"/>
37       <family val="2"/>
38       <scheme val="minor"/>
39     </rPr>
40     <t xml:space="preserve"> of </t>
41   </r>
42   <r>
```

```

1      <rPr>
2          <u/>
3          <sz val="11"/>
4          <color theme="1"/>
5          <rFont val="Calibri"/>
6          <family val="2"/>
7          <scheme val="minor"/>
8      </rPr>
9      <t>formatting</t>
10     </r>
11     <r>
12         <rPr>
13         <sz val="11"/>
14         <color theme="1"/>
15         <rFont val="Calibri"/>
16         <family val="2"/>
17         <scheme val="minor"/>
18     </rPr>
19     <t xml:space="preserve"> applied</t>
20     </r>
21 </si>
```

22 Reading the string from left to right as it appears in the cell, each word represents a change in formatting. This  
 23 change in formatting corresponds to separate run elements r to separate the text with different formatting.  
 24 Every word is expressed using a run element r, which expresses the properties of the text rPr and the text  
 25 itself t.

26 Since there are no properties associated with the first word "This", the text inherits the default formatting for  
 27 the cell.

28 The rich text expression for the second string "string" contains a bold faced font element indicator b in the run  
 29 properties rPr, therefore this text will have bold face applied. While other text formatting properties are  
 30 expressed, they are the same as the cell formatting. This additional information is expressed for the sake of  
 31 clarity and completeness of expression.

32 The rich text expression for the third string "has" contains a color element indicator color in the run  
 33 properties rPr. Therefore, the color of the text associated with this set of run properties will be red, according  
 34 to the color value expressed.

35 The formatting for the remaining words in this rich text string can be deduced in a similar manner, such that  
 36 "a" has default formatting applied, "variety" is both italicized and green, "of" has default formatting applied,  
 37 "formatting" is underlined, and "applied" has default formatting applied.

## 1    3.4    Tables

### 2    3.4.1    Overview

3    A table helps organize and provide structure to lists of information in a worksheet. Tables have clearly labeled  
 4    columns, rows, and data regions. Tables enable users to sort, analyze, format, manage, add, and delete  
 5    information. Here's an example of what a table can look like:

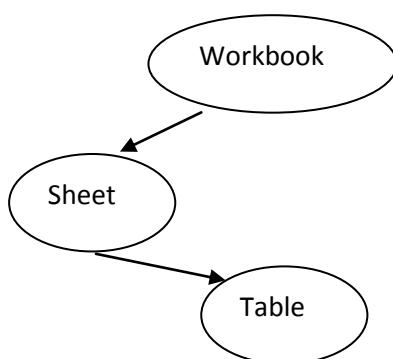
	A	B	C	D
1				
2	State	City	Zipcode	
3	WA	Seattle	98101	
4	WA	Seattle	98102	
5	WA	Tacoma	98467	
6	OR	Portland	97204	
7	OR	Portland	97205	
8	ID	Post Falls	83854	
9	Count		6	

7    Notice that this table has column headings "State", "City", and "Zipcode". There is a row summarizing the data,  
 8    in this case a count of zip codes. The formatting helps make clear where the column headings are (Bold faced,  
 9    bordered on top and bottom all the way across), where the data region is (banded row stripes), and where the  
 10   totals row is (double border separating data from totals, bold face totals label).

11   Because the table feature has been applied to this data, special behaviors can be applied which help the user  
 12   perform useful actions. For example, if the user types additional data in row 10, the table can expand and  
 13   automatically add that data to the data region of the table. Similarly, adding a column is as easy as typing a  
 14   new column heading to the right or left of the current column headings. Filter and sort abilities are  
 15   automatically surfaced to the user via the drop down arrows. Special calculated columns can be created which  
 16   summarize or calculate data in the table. These columns have the ability to expand and shrink according to size  
 17   of the table, and maintain proper formula referencing.

18   Tables can be made from data already present in the worksheet. Tables can also be the result of an external  
 19   data query. Finally, tables can be the result of mapping a collection of repeating XML elements to a worksheet  
 20   range.

### 21    3.4.2    File Architecture



1

2

3 Each table is referred to by a relationship from a sheet to the table. The relationship is found in the sheet's  
 4 \_rels directory. The sheet XML also references the ID of this relationship, because there can be more than one  
 5 table in a sheet.

6 The sheet XML stores the numeric and textual data. The table XML records the various attributes for the  
 7 particular table object.

### 8 3.4.3 Example: Table

9 This example demonstrates a table created from data that was previously entered in the sheet. (See §3.15 for  
 10 a discussion of Tables with XML data bindings.)

### 11 3.4.4 Illustration

12 Consider the example provided in §3.4.1 above:

	A	B	C	D	
1					
2					
3	State	City	Zipcode		
4	WA	Seattle	98101		
5	WA	Seattle	98102		
6	WA	Tacoma	98467		
7	OR	Portland	97204		
8	OR	Portland	97205		
9	ID	Post Falls	83854		
10	Count		6		
11					

13  
 14 Notice that this table has column headings "State", "City", and "Zipcode". There is a row summarizing the data,  
 15 in this case a count of Zip codes. In the "State" column abbreviations of United States State names are listed. In  
 16 the City column are listed names of Cities within those states. Finally, within the Zipcode column are postal  
 17 codes residing within those cities.

18 The table has a style applied, which provides unique formatting for:

- 19
- The column heading area, with bold facing and top and bottom borders

20

  - The data area, with banded striping

21

  - The total row area, with a top double border and bold facing

22

  - The last column area, with a solid background fill of blue.

### 3.4.5 The Sheet XML

The sheet XML for this example references the table definition part:

```

3 ...
4     <tableParts count="1">
5         <tablePart r:id="rId1"/>
6     </tableParts>
7 </worksheet>
```

### 3.4.6 The Table XML

The tableParts collection appears after the sheetData section of the sheet. This sheet references a table whose relationship Id r:id value is rId1.

The Table definition XML for this example:

```

12 <table xmlns= ... id="8" name="Table19" displayName="Table19" ref="B3:D10"
13     totalsRowCount="1">
14     <autoFilter ref="B3:D10"/>
15     <tableColumns count="3">
16         <tableColumn id="1" name="State" totalsRowLabel="Count"
17             totalsRowDxfId="0"/>
18         <tableColumn id="2" name="City"/>
19         <tableColumn id="3" name="Zipcode" totalsRowFunction="count"/>
20     </tableColumns>
21     <tableStyleInfo name="TableStyleMedium16" showFirstColumn="0"
22         showLastColumn="1"
23         showRowStripes="1" showColumnStripes="0"/>
24 </table>
```

name indicates that the Table's name is Table19, and the ref value indicates that it occupies the range B3:D10 on the relevant sheet. totalsRowCount value of 1 indicates that this Table's total row is visible.

The autoFilter element indicates that the autoFilter feature is applied to the range B3:D10. The tableColumns collection indicates there are 3 columns in the table, whose names are "State", "City", and "Zipcode". Furthermore, the column titled "State" has a label in the total row, whose caption is "Count", and the column titled "Zipcode" has a total row function applied, whose function is "count".

The tableStyleInfo element indicates various attributes of this Table's style and formatting. In this example, name indicates that the Table style named "TableStyleMedium16" has been applied. Additionally, even though formatting has been defined by this table style to indicate uniquely the first column of the table, since showFirstColumn is set to 0 (false), this first column formatting will not be applied to the table. The same is true for column stripes. Since showColumnStripes is set to 0 (false), even though formatting for column stripes is defined by the table style, it is not applied to this table. However, both row striping and last column formatting is set to be applied to this table, as indicated by showRowStripes and showLastColumn.

## 1    3.5    Calculation Chain

### 2    3.5.1    Overview

3    The Calculation Chain part specifies the order in which cells in the workbook were last calculated. It only  
 4    records information about cells containing formulas. It does *not* include any information about the formula-  
 5    dependency calculation tree. In other words, the Calculation Chain part does not indicate the dependencies  
 6    that formulas have on other cell values; it only indicates the order in which the cells were last calculated.

7    Any particular calculation event can cause the calculation chain order to be rearranged or altered. For  
 8    example, adding more formulas to the workbook will add references in the Calculation Chain part.

9    Another example of how the calculation order can be updated involves the idea of partial calculation. *Partial*  
 10   *calculation* is an optimization a spreadsheet application can implement to calculate only those cells that are  
     dependent on other cells whose values have changed, and to ignore other formulas in the workbook. This  
     helps to avoid redundantly recalculating results that are already known. Therefore, if a set of formulas that  
     were previously ignored during a calculation become required for calculation (due to a cell's value changing),  
     then these formulas will move to "first" on the calculation chain so they can be evaluated.

15   While calculation chain information can be loaded by a spreadsheet application, it is not required. A  
 16   calculation chain can be constructed in memory at load-time based on the formulas and their  
     interdependence, if the spreadsheet application finds this information useful. The order expressed in the  
     Calculation Chain part does not force or dictate to the implementing application the order in which  
     calculations must be performed at runtime.

### 20    3.5.2    Example

21   Consider the following set of formulas in a workbook:

	A		B		C		D	
1	1	1	11	11				
2	=A1+1	2	=B1+1	12				
3	=A2+1	3	=B2+1	13				
4	=A3+1	4	=B3+1	14				
5	=A4+1	5	=B4+1	15				
6	=A5+1	6	=B5+1	16				
7	=A6+1	7	=B6+1	17				
8	=A7+1	8	=B7+1	18				
9	=A8+1	9	=B8+1	19				
10	=A9+1	10	=B9+1	20	=B10+A10	30	=C10+10	40
11								

22  
23

24   Note that the content of each cell is displayed on the left side of the cell, and the evaluated value is  
 25   superimposed on the right side of each cell.

1 Cell A1 contains the numeric constant 1. Cell A2 contains the formula =A1+1, and this formula is filled down  
 2 to A10. Cell B1 contains the numeric constant 11. Cell B2 contains the formula =B1+1, and this formula is filled  
 3 down to B10. C10 contains the formula =B10+A10, whose current value is 30. D10 contains the formula  
 4 =C10+10, whose current value is 40.

5 Because dependencies among formulas do affect calculation order, dependencies will be discussed briefly  
 6 here. The formula in D10 depends on the result from C10. The formula in C10 depends on the results from  
 7 both A10 and B10. The formulas in column A each depend on the cell above them, ultimately depending on  
 8 the constant value in A1. The formulas in column B each depend on the cell above them, ultimately depending on  
 9 the constant value in B1.

10 This example was created by first entering the values in A1 then B1. Next, typing the formula in A2, and filling  
 11 that across to B2. Then the formulas in A2 and B2 were concurrently filled down to A10:B10. Next, the  
 12 formula was typed into C10, and finally the formula in D10 was entered. The application was in  
 13 automatic/partial calculation mode when this information was entered.

#### 14 3.5.2.1 Partial Calculation

15 The calculation chain might be saved after initially entering the data and saving the workbook, as follows:

```
16 <calcChain xmlns="...">
17   <c r="D10" i="1"/>
18   <c r="C10"/>
19   <c r="A3"/>
20   <c r="B3"/>
21   <c r="A4"/>
22   <c r="B4"/>
23   <c r="A5"/>
24   <c r="B5"/>
25   <c r="A6"/>
26   <c r="B6"/>
27   <c r="A7"/>
28   <c r="B7"/>
29   <c r="A8"/>
30   <c r="B8"/>
31   <c r="A9"/>
32   <c r="B9"/>
33   <c r="A10"/>
34   <c r="B10"/>
35   <c r="B2"/>
36   <c r="A2"/>
37 </calcChain>
```

38 Every c element represents a cell containing a formula. The first cell calculated appears first (top-to-bottom),  
 39 and so on. The reference attribute r indicates the cell's address in the sheet. The index attribute i indicates the

1 index of the sheet with which that cell is associated. The sub-chain attribute s (not present in this first  
 2 example) indicates that this cell can be treated as a sub chain of the preceding cell. Sub-chains can be useful  
 3 when calculation can be multi-threaded or calculated concurrently. Whenever a cell does not contain an i or s  
 4 attribute, it is understood to inherit these values from the previous cell.

5 Because of the way in which the workbook was initially created and saved, cell D10 should be the first cell  
 6 calculated. The reason for this, which cannot be determined from examining the XML, is that cell D10 is the  
 7 only cell that needs calculating, due to the partial calculation optimization. Since the cells A2:B10 and C10  
 8 were previously calculated (as a result of entering formulas in those cells), when entering the formula in D10,  
 9 D10 is the only cell that needs to be calculated.

10 This calculation chain indicates that after D10 is calculated, C10 can be evaluated. In looking at the  
 11 dependencies, it is understood that during a full calculation, C10 would be evaluated before D10 can be  
 12 evaluated. However, because of the partial calculation optimization, at the time C10 was entered, it was  
 13 placed first on the calculation chain to be evaluated. Subsequent to that, D10 was entered, and so C10 was  
 14 moved to second position in the calculation chain, and that is why it is currently in the second place.

15 Moving through the rest of the cells with this same logic, just before C10 was entered, A3, then B3, then A4,  
 16 then B4, and so on up to A10 and B10 were added and then evaluated as part of the fill-down operation.

17 Finally, cells A2 and B2 were the first formulas to be added and calculated. All formulas in the workbook were  
 18 added after A2 and B2 were evaluated. Since A2 and B2 didn't need to be re-evaluated (due to the partial  
 19 calculation optimization) after that, they eventually settled to the end of the calculation chain.

### 20 3.5.2.2 First Full Calculation

21 Below is how the calculation chain will look after changing the values of A1 and B1, or after forcing the  
 22 application to perform a full calculation on the entire set of formulas:

```
23 <calcChain xmlns="...">
24   <c r="B2" i="1"/>
25   <c r="B3" s="1"/>
26   <c r="B4" s="1"/>
27   <c r="B5" s="1"/>
28   <c r="B6" s="1"/>
29   <c r="B7" s="1"/>
30   <c r="B8" s="1"/>
31   <c r="B9" s="1"/>
32   <c r="B10" s="1"/>
33   <c r="C10" s="1"/>
34   <c r="D10" s="1"/>
35   <c r="A2"/>
36   <c r="A3" s="1"/>
37   <c r="A4" s="1"/>
```

```

1   <c r="A5" s="1"/>
2   <c r="A6" s="1"/>
3   <c r="A7" s="1"/>
4   <c r="A8" s="1"/>
5   <c r="A9" s="1"/>
6   <c r="A10" s="1"/>
7   </calcChain>
```

8 Now the order of calculation seems more in line with the way in which the formulas depend on each other:  
9 cells B2:B10 are calculated in order, and cells A2:A10 are calculated in order.

10 Additionally, the application has discovered that the formulas in column B can be calculated in parallel with the  
11 formulas in column A (i.e., they don't depend on each other). This is evidenced by the presence of s="1" on the  
12 cell element for cells B2:B10, indicating that B2:10 are part of a "child-chain" starting with B2. Note also that  
13 C10 and D10 are included in that child chain, even though these formulas do, in fact, depend on calculated  
14 values from column A. This is due to the multi-threaded nature of the calculation engine. Currently the chain  
15 on which C10 and D10 reside can be calculated concurrently with the chain on which A2:A10 reside because  
16 by the time C10 and D10 need to be calculated (e.g., by CPU #1), A10 has already been calculated (e.g., by  
17 CPU #2). In some future calculation, the timing may be different, and at that time, the application will need to  
18 resort to moving C10 and D10 to a new calculation level (see §3.5.2.3).

### 19 3.5.2.3 Twentieth Full Calculation

20 After several full calculation iterations, this particular calculation chain will settle into a stable state. For  
21 example:

```

22   <calcChain xmlns="...">
23     <c r="B2" i="1"/>
24     <c r="B3" s="1"/>
25     <c r="B4" s="1"/>
26     <c r="B5" s="1"/>
27     <c r="B6" s="1"/>
28     <c r="B7" s="1"/>
29     <c r="B8" s="1"/>
30     <c r="B9" s="1"/>
31     <c r="B10" s="1"/>
32     <c r="A2"/>
33     <c r="A3" s="1"/>
34     <c r="A4" s="1"/>
35     <c r="A5" s="1"/>
36     <c r="A6" s="1"/>
```

```

1   <c r="A7" s="1"/>
2   <c r="A8" s="1"/>
3   <c r="A9" s="1"/>
4   <c r="A10" s="1"/>
5   <c r="C10" l="1"/>
6   <c r="D10" s="1"/>
7   </calcChain>
```

8 The difference introduced here is the concept of a dependency-level attribute *l*. This flag indicates that all  
 9 chain and child chain concurrent calculation must be completed (and all cells will have newly calculated values)  
 10 before proceeding with calculation.

11 In this example, cells C10 and D10 are marked to exist in a new and separate dependency level from the cells  
 12 A2:A10 and B2:B10. This makes sense given how the dependencies for these formulas are set up: A2:A10  
 13 can be calculated concurrently with B2:B10 because they do not depend on each other. A2:A10 exists as one  
 14 calculation chain, and B2:B10 exist as another parallel calculation chain. However, C10 and D10 are both  
 15 dependent on calculated results from the two parallel chains, and so can only be calculated after the first set  
 16 of parallel calculations are completed.

17 Dependency-Level *l* flags indicate where calculation must wait for all concurrent threads to complete before  
 18 continuing with calculation

## 19 3.6 Comments

### 20 3.6.1 Overview

21 A *comment* is a rich text note that is attached to, and associated with, a cell, separate from other cell content.  
 22 Comment content is stored separate from the cell, and is displayed in a drawing object (like a text box) that is  
 23 separate from, but associated with, a cell. Comments are used as reminders, such as noting how a complex  
 24 formula works, or to provide feedback to other users. Comments can also be used to explain assumptions  
 25 made in a formula or to call out something special about the cell.

### 26 3.6.2 Example

27 Consider the following graphic representation of a worksheet:

	A	B	C	D	E	F	G	H	I	J	K
1											
2			Q1	Q2	Q3	Q4					
3	Revenue			412.52							
4	Expenses			697.37							
5	Total			\$1,109.89							
6											
7											

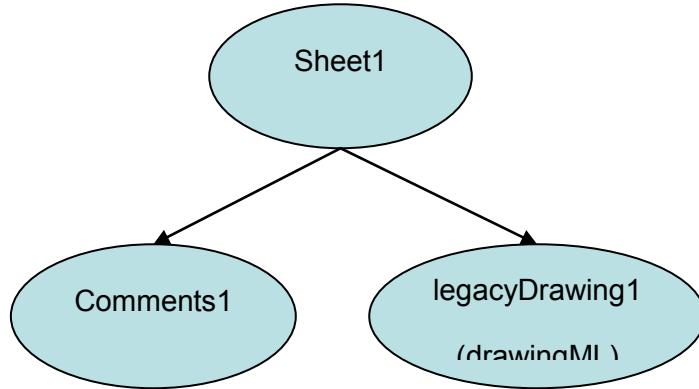
Chad:  
Why such high expense?

28

1 Note that cells D4 and J4 have comments.

### 2 3.6.3 File Architecture

3 Inside the file, the comment content ("Comments1") is expressed separately from the sheet information  
 4 ("Sheet1"), and separately from the drawing information ("legacyDrawing1") for the containing object:



5

6 The parts are related using relationships from the sheet to the comments and drawings parts. Comments are  
 7 stored together at the sheet-level. Therefore, if there are five worksheets in a workbook, and three of those  
 8 contain cells with associated comments, there will be three comment parts in the file, one for each sheet.

### 9 3.6.4 The XML

```

10 <comments>
11   <authors>
12     <author>Chad</author>
13     <author>CBR</author>
14   </authors>
15   <commentList>
16     <comment ref="D4" authorId="0">
17       <text>
18         <r>
19           <rPr>
20             <b/>
21             <sz val="8"/>
22             <color indexed="81"/>
23             <rFont val="Calibri"/>
24             <charset val="1"/>
25             <scheme val="minor"/>
26           </rPr>
27           <t>Chad:</t>
28         </r>
  
```

```

1      <r>
2          <rPr>
3              <sz val="8"/>
4              <color indexed="81"/>
5              <rFont val="Calibri"/>
6              <charset val="1"/>
7              <scheme val="minor"/>
8          </rPr>
9          <t xml:space="preserve">Why such high expense?</t>
10         </r>
11     </text>
12 </comment>
13 <comment ref="J4" authorId="1">
14     <text>
15         <r>
16             <rPr>
17                 <b/>
18                 <sz val="8"/>
19                 <color indexed="81"/>
20                 <rFont val="Calibri"/>
21                 <charset val="1"/>
22                 <scheme val="minor"/>
23             </rPr>
24             <t>CBR:</t>
25         </r>
26         <r>
27             <rPr>
28                 <sz val="8"/>
29                 <color indexed="81"/>
30                 <rFont val="Calibri"/>
31                 <charset val="1"/>
32                 <scheme val="minor"/>
33             </rPr>
34             <t xml:space="preserve">
35                 Pending a couple expenses in December.</t>
36         </r>
37     </text>
38 </comment>
39 </commentList>
40 </comments>

```

### 3.6.5 Authors

```

1   <comments>
2     <authors>
3       <author>Chad</author>
4       <author>CBR</author>
5     </authors>
6

```

7 The authors collection is a unique list of author names for all comments on a particular sheet. In this example,  
8 there are two authors listed. Each comment definition references the authors collection by zero-based index.

### 3.6.6 Comments

```

10    <commentList>
11      <comment ref="D4" authorId="0">
12        <text>
13          <r>
14            <rPr>
15              <b/>
16              <sz val="8"/>
17              <color indexed="81"/>
18              <rFont val="Calibri"/>
19              <charset val="1"/>
20              <scheme val="minor"/>
21            </rPr>
22            <t>Chad:</t>
23          </r>
24          <r>
25            <rPr>
26              <sz val="8"/>
27              <color indexed="81"/>
28              <rFont val="Calibri"/>
29              <charset val="1"/>
30              <scheme val="minor"/>
31            </rPr>
32            <t xml:space="preserve">
33              Why such high expense?</t>
34          </r>
35        </text>
36      </comment>

```

37 commentList is a listing of all comments on a sheet. The first comment in this example has ref="D4" and  
38 authorId="0". This indicates that the comment is associated with cell D4 and is associated with the author  
39 "Chad".

1 The content of comment is rich text, following the rich text schematics, including the author name and actual  
 2 comment.

### 3 **3.7 Styles**

#### 4 **3.7.1 Overview**

5 There are several ways to express formatting applied to objects in a worksheet. SpreadsheetML supports the  
 6 concepts of Styles, Themes, and Direct Formatting applied to cell ranges, Tables, PivotTables, Charts, and  
 7 Shapes.

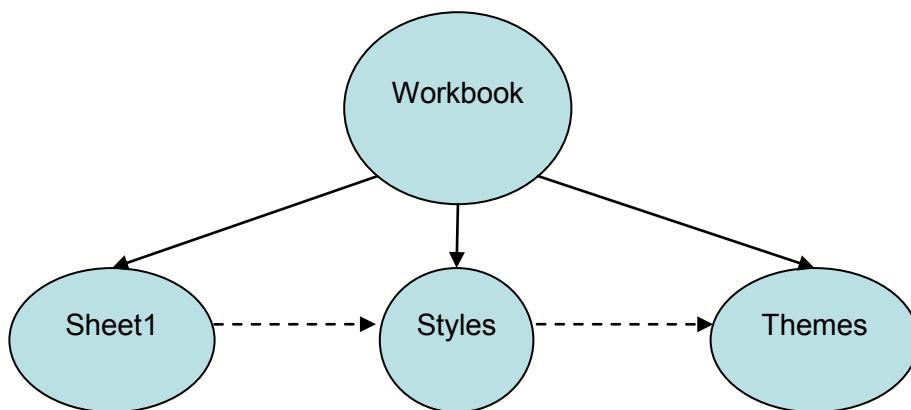
8 A *Style* is a named collection of formatting elements. A *cell style* can specify number format, cell alignment,  
 9 font information, cell border specifications, colors, and background / foreground fills. *Table styles* specify  
 10 formatting elements for the regions of a table (e.g. make the header row & totals bold face, and apply light  
 11 gray fill to alternating rows in the data portion of the table to achieve striped or banded rows). *PivotTable*  
 12 *styles* specify formatting elements for the regions of a PivotTable (e.g. 1st & 2nd level subtotals, row axis,  
 13 column axis, and page fields).

14 A *Style* can specify color, fonts, and shape effects directly, or these elements can be referenced indirectly by  
 15 referring to a *Theme* definition. Using *styles* allows for quicker application of formatting and more consistently  
 16 stylized documents.

17 *Themes* define a set of colors, font information, and effects on shapes (including Charts). If a style or  
 18 formatting element defines its color, font, or effect by referencing a theme, then picking a new theme switches  
 19 all the colors, fonts, and effects for that formatting element.

20 Applying *Direct Formatting* means that particular elements of formatting (e.g. a bold font face or a number  
 21 format) have been applied, but the elements of formatting have been chosen individually instead of  
 22 collectively by choosing a named *Style*. Note that when applying direct formatting, themes can still be  
 23 referenced, causing those elements to change as the theme is changed.

#### 24 **3.7.2 File Architecture**



25  
 26 For a workbook, a single Styles part holds all its formatting definitions. Similarly, a single Themes part defines  
 27 the theme information used in the workbook. These parts are referenced by relationship from the Workbook

1 part. Each of the formatted objects refers by index to a master formatting definition record expressed in the  
 2 Styles part. This master formatting record references additional supporting formatting element collections in  
 3 the Styles part. If the formatting element in the Styles part is defined in terms of a theme, then this formatting  
 4 element will reference an index to a theme element defined in the Theme part. The solid arrows denote that  
 5 there are relationships expressed from the Workbook part to each of the Sheet1, Styles, and Themes parts.  
 6 The dotted arrow from the Sheet1 part to the Styles part indicates that there are references in the Sheet1  
 7 part's markup that refer, by index, to elements defined in the markup in the Styles part. Similarly, the dotted  
 8 arrow from the Styles part to the Themes part indicates that there are references in the Styles part's markup  
 9 that refer, by index, to elements defined in the markup in the Themes part.

### 10 **3.7.3 Organization in the Styles Part**

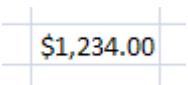
11 The Styles part is organized into element collections as described in the following subclauses. The element  
 12 collections must appear in the order presented below. The element collections are siblings in the Styles part  
 13 XML definition, whose parent, the root node of this part, is <styleSheet>.

14 Please refer to the reference material or schemas themselves for more precise descriptions on the required  
 15 order of elements.

#### 16 **3.7.3.1 Number Format Expressions**

17 This is where cell number formats used in this workbook are expressed. This collection never references a  
 18 theme. In this collection the numFmtId attribute is an actual ID, unlike the other sibling collections. That is,  
 19 instead of relying on the order in which a particular numFmt appears, it is referenced elsewhere by calling out  
 20 the numFmtId value. Then the corresponding numFmt record can be found by finding the numFmt record with  
 21 the matching numFmtId value. In the case of number formats, a set of numFmtId values are predefined and  
 22 fixed by this specification. These values map to actual number formatting expressions.

23 The following XML would format a cell containing the value 1234 to look like this:

24 

```
25   <numFmts count="1">
26     <numFmt numFmtId="165" formatCode=""$##0.00"/>
27   </numFmts>
```

28 A <numFmt> definition is referenced by ID (numFmtId) from either a <cellXf> or a <CellStyleXf>.

29 To read more about how to interpret number format codes like the value found in formatCode above, please  
 30 read the reference section on numFmt, in the Styles section.

#### 31 **3.7.3.2 Font Definitions**

32 This is where font definitions used in this workbook are expressed. Elements of the font definition may  
 33 reference theme definitions.

```

1   <fonts count="1">
2     <font>
3       <b/>
4       <sz val="11"/>
5       <color theme="1"/>
6       <name val="Calibri"/>
7       <family val="2"/>
8     </font>
9   </fonts>

```

10 This font definition specifies bold face, font size of "11", a font color specified in the Theme part, specifically  
 11 the color whose index is "1" in the `<clrScheme>` collection, a font name of "Calibri", and whose font family  
 12 value is "2" (for more explanation on font family, please refer to the Styles reference material). A `<font>`  
 13 definition is referenced by index (`fontId`) from either a `<cellXf>` or a `<cellStyleXf>`.

14 A `font` record is referenced by zero-based index, meaning the numerical order in which the `font` appears  
 15 under `fonts`.

### 3.7.3.3 Fill Definitions

17 This is where fills used in the workbook are expressed.

```

18 <fills count="1">
19   <fill>
20     <patternFill patternType="solid">
21       <fgColor theme="4"/>
22       <bgColor theme="4"/>
23     </patternFill>
24   </fill>
25 </fills>

```

26 This fill definition specifies a solid pattern fill, whose color uses a themed color, whose index is "4" in the  
 27 `<clrScheme>` collection of the Theme part. A `<fill>` definition is referenced by index (`fillId`) from either  
 28 a `<cellXf>` or a `<cellStyleXf>`.

29 A `fill` record is referenced by zero-based index, meaning the numerical order in which the `fill` appears  
 30 under `fills`.

31

### 3.7.3.4 Borders Definitions

33 This is where border formats are specified.

```

34 <borders count="1">
35   <border>
36     <left/>

```

```

1      <right/>
2      <top/>
3      <bottom/>
4      <diagonal/>
5    </border>
6  </borders>

```

7 This example specifies a cell with left, right, top, and bottom borders. A `<border>` definition is referenced by  
8 index (`borderId`) from either a `<cellXf>` or a `<cellStyleXf>`.

9 A border record is referenced by zero-based index, meaning the numerical order in which the border  
10 appears under borders.

11

### 12 3.7.3.5 Master Records - Cell Styles

13 The 'master' cell style record (`<xf>`) ties together all the formatting (e.g. number format, font information,  
14 and fill) for a named cell style. An `<xf>` inside `<cellStyleXfs>` is referenced by zero-based index (not ID)  
15 (`xfId`) from a `<cellStyle>` definition, which names a particular cell style.

```

16  <cellStyleXfs count="1">
17    <xf numFmtId="0" fontId="0" fillId="0" borderId="0"/>
18  </cellStyleXfs>

```

### 19 3.7.3.6 Master Records - Formatting

20 The 'master' cell style record (`<xf>`) ties together all the formatting (e.g. number format, font information,  
21 and fill) for a cell's direct formatting. An `<xf>` inside `<cellXfs>` is referenced by zero-based index (not ID) (s)  
22 from a cell definition (`<c>`) in one of the sheets.

```

23  <cellXfs count="1">
24    <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"/>
25  </cellXfs>

```

### 26 3.7.3.7 Cell Styles

27 This is a collection of cell styles used in the workbook.

```

28  <cellStyles count="1">
29    <cellStyle name="Accent1" xfId="1" builtinId="29"/>
30  </cellStyles>

```

### 31 3.7.3.8 Differential Formatting Records

32 "Differential formatting" enables subsets of formatting to be specified, without overriding other elements of  
33 formatting. For example, if it is desired to express "add bold face to whatever formatting is already there",  
34 then a `<dxf>` definition can be used. `<dxf>` definitions are used to express additional (or "differential")  
35 formatting that will be applied via Table styles or PivotTable styles. `<dxf>` definitions are referenced by index

1 (dxId) from a <tableStyleElement>. The formatting elements used in a <dxfs> definition are subsets of  
 2 formatting collections described above.

3 A dxfs record is referenced by zero-based index, meaning the numerical order in which the dxfs appears under  
 4 dxfs.

5

```
6   <dxfs count="1">
7     <dxfs>
8       <font>
9         <b/>
10        <color theme="0"/>
11      </font>
12      <fill>
13        <patternFill patternType="solid">
14          <fgColor theme="5"/>
15          <bgColor theme="5"/>
16        </patternFill>
17      </fill>
18    </dxfs>
19  </dxfs>
```

### 20 3.7.3.9 Custom Table Style Definitions

21 Built-in Table and PivotTable styles are not saved out, only custom-defined styles are saved out. In this  
 22 example, a custom table style defines formatting for an element of a table, the "whole table" region.

```
23 <tableStyles count="1" defaultTableStyle="TableStyleMedium9"
24   defaultPivotStyle="PivotStyleLight16">
25   <tableStyle name="TableStyleMedium10 - Custom" pivot="0" count="1">
26     <tableStyleElement type="wholeTable" dxId="6"/>
27   </tableStyle>
28 </tableStyles>
```

### 29 3.7.4 Example

#### 30 3.7.4.1 Illustration

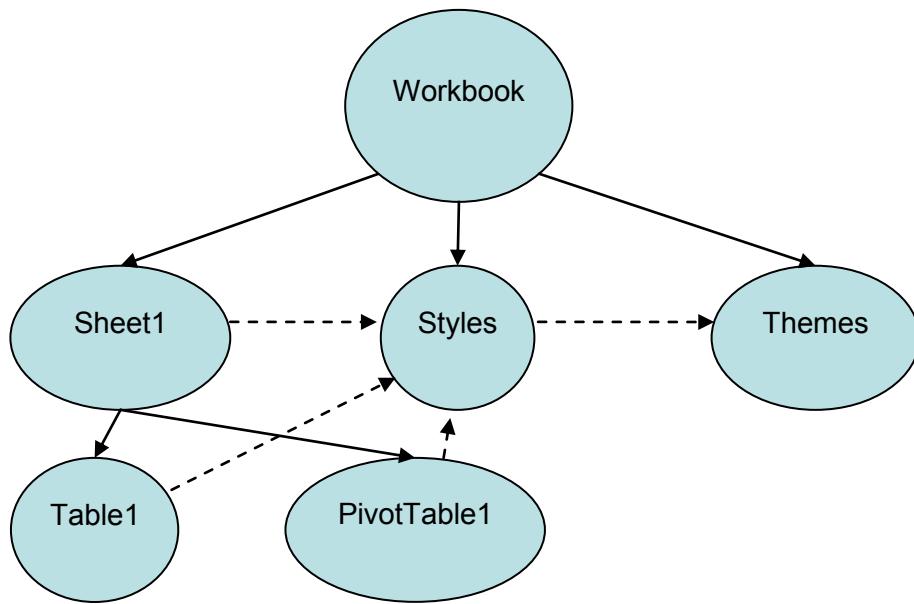
31 For this example, consider this graphic representation of a worksheet:

The screenshot shows a Microsoft Excel spreadsheet with several data regions and styling. At the top left, there's a table from L2:O6 with rows for Revenue, Expenses, and Total, and columns for Q1 through Q4. Cells D2, F2, H2, J2, and B3:B4 have a blue fill and white Calibri font. The header row L2 has a dark red background with white text for columns M, N, and O. The total row O6 has a black background with white text. Below this is a PivotTable from L9:N17. The header row L9:M9 has a dark red background with white text for columns M, N, and O. The body of the PivotTable from L11:N12 has a light gray background. The totals row L17:N17 has a black background with white text. The PivotTable interface includes dropdowns for 'Column4' and '(All)' under 'Data'.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2				Q1	Q2	Q3	Q4		Column1	Column2	Column3	Column4			
3	Revenue			412.52	515.21	866.74	524.92		A		381.41	513.27	357.29		
4	Expenses			697.37	539.72	149.51	546.44		B		470.33	411.76	723.52		
5	Total			\$1,109.89	\$1,054.93	\$1,016.25	\$1,071.36		C		624.47	287.49	365.38		
6									D		17.77	775.36	969.69		
7															
8															
9									Column4	(All)					
10															
11															
12															
13															
14															
15															
16															
17															

- 2 Looking at the top left region of the illustration, cells D2, F2, H2, J2, and B3:B4 have the cell style "Accent1" applied to them. "Accent1" is a theme-driven style, and results in a blue cell fill and white / Calibri font formatting. Additionally these cells have direct border formatting applied which isn't specified as part of the "Accent1" cell style.
- 6 Cells D3:D4, F3:F4, H3:H4, and J3:J4 have a light blue cell fill applied. The light blue color is part of a themed color scheme, and will update when a new theme is selected.
- 8 Cells D5, F5, H5, and J5 have a currency number format applied as well as a green cell fill. While the cell fill is a themed color, the number format is fixed and will not vary or change if a new theme is selected.
- 10 The table in L2:O6 has a table style applied, called "TableStyleMedium10", which specifies formatting for the header row, row striping, and total row (even though the total row isn't shown in this example).
- 12 The PivotTable in L9:N17 has a PivotTable style applied, called "PivotStyleMedium10", which specifies formatting for the regions of a PivotTable, including the page field area in L9:M9, the header row area in L11:N12, the totals row in L17:N17, and the body of data in L13:N16.

## 1    3.7.4.2      File Architecture



2

3    All of the cells illustrated are defined in the "Sheet1" part in this example. The table is defined in the "Table1"  
 4    part and the PivotTable is defined in the part named "PivotTable1". Each of the formatted objects refers to a  
 5    set of formatting definitions which are expressed in the "Styles" part. If the formatting element is part of a  
 6    themed set, the element will reference a theme element defined in the "Themes" part. The solid arrows  
 7    represent relationships among the parts, the dotted arrows represent references by Id or index to various  
 8    elements in the target part.

## 9    3.7.4.3      The XML For This Example

```

10   <styleSheet>
11     <numFmts count="1">
12       <numFmt numFmtId="164" formatCode=""$&quot;#,##0.00"/>
13     </numFmts>
14     <fontS count="5">
15       <font>
16         <sz val="11"/>
17         <color theme="1"/>
18         <name val="Calibri"/>
19         <scheme val="minor"/>
20       </font>
21       <font>
22         <b/>
23         <sz val="11"/>
24         <color theme="1"/>
25         <name val="Calibri"/>
26         <family val="2"/>
27       </font>
  
```

```
1   <font>
2       <b/>
3       <sz val="8"/>
4       <color indexed="81"/>
5       <name val="Calibri"/>
6       <charset val="1"/>
7       <scheme val="minor"/>
8   </font>
9   <font>
10      <sz val="8"/>
11      <color indexed="81"/>
12      <name val="Calibri"/>
13      <charset val="1"/>
14      <scheme val="minor"/>
15   </font>
16   <font>
17      <sz val="11"/>
18      <color theme="0"/>
19      <name val="Calibri"/>
20      <scheme val="minor"/>
21   </font>
22 </fonts>
23 <fills count="5">
24   <fill>
25     <patternFill patternType="none"/>
26   </fill>
27   <fill>
28     <patternFill patternType="gray125"/>
29   </fill>
30   <fill>
31     <patternFill patternType="solid">
32       <fgColor theme="4"/>
33       <bgColor theme="4"/>
34     </patternFill>
35   </fill>
36   <fill>
37     <patternFill patternType="solid">
38       <fgColor theme="6" tint="0.59999389629810485"/>
39       <bgColor indexed="65"/>
40     </patternFill>
41   </fill>
42   <fill>
43     <patternFill patternType="solid">
```

```
1             <fgColor theme="4" tint="0.79998168889431442"/>
2             <bgColor indexed="65"/>
3         </patternFill>
4     </fill>
5 </fills>
6 <borders count="5">
7     <border>
8         <left/>
9         <right/>
10        <top/>
11        <bottom/>
12        <diagonal/>
13    </border>
14    <border>
15        <left/>
16        <right/>
17        <top/>
18        <bottom style="double">
19            <color indexed="64"/>
20        </bottom>
21        <diagonal/>
22    </border>
23    <border>
24        <left style="thick">
25            <color auto="1"/>
26        </left>
27        <right style="thick">
28            <color auto="1"/>
29        </right>
30        <top style="thick">
31            <color auto="1"/>
32        </top>
33        <bottom style="thick">
34            <color auto="1"/>
35        </bottom>
36        <diagonal/>
37    </border>
38    <border>
39        <left style="thick">
40            <color auto="1"/>
41        </left>
42        <right style="thick">
43            <color auto="1"/>
```

```

1          </right>
2          <top style="thick">
3              <color auto="1"/>
4          </top>
5          <bottom/>
6          <diagonal/>
7      </border>
8      <border>
9          <left style="thick">
10             <color auto="1"/>
11         </left>
12         <right style="thick">
13             <color auto="1"/>
14         </right>
15         <top/>
16         <bottom style="thick">
17             <color auto="1"/>
18         </bottom>
19             <diagonal/>
20         </border>
21     </borders>
22     <cellStyleXfs count="2">
23         <xf numFmtId="0" fontId="0" fillId="0" borderId="0"/>
24         <xf numFmtId="0" fontId="4" fillId="2" borderId="0"
25 applyNumberFormat="0" applyBorder="0" applyAlignment="0" applyProtection="0">
26             <protection/>
27         </xf>
28     </cellStyleXfs>
29     <cellXfs count="14">
30         <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"/>
31         <xf numFmtId="0" fontId="1" fillId="0" borderId="0" xfId="0"
32 applyFont="1"/>
33         <xf numFmtId="4" fontId="0" fillId="0" borderId="0" xfId="0"
34 applyNumberFormat="1" applyBorder="1"/>
35         <xf numFmtId="164" fontId="0" fillId="0" borderId="0" xfId="0"
36 applyNumberFormat="1" applyBorder="1"/>
37         <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"
38 pivotButton="1"/>
39         <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"
40 applyAlignment="1">
41             <alignment horizontal="left"/>
42         </xf>

```

```

1      <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"
2  applyNumberFormat="1"/>
3      <xf numFmtId="0" fontId="4" fillId="2" borderId="2" xfId="1"
4  applyBorder="1"/>
5      <xf numFmtId="0" fontId="4" fillId="2" borderId="3" xfId="1"
6  applyBorder="1"/>
7      <xf numFmtId="0" fontId="4" fillId="2" borderId="4" xfId="1"
8  applyBorder="1"/>
9      <xf numFmtId="0" fontId="1" fillId="3" borderId="0" xfId="0"
10 applyFont="1" applyFill="1"/>
11     <xf numFmtId="164" fontId="0" fillId="3" borderId="0" xfId="0"
12  applyNumberFormat="1" applyFill="1" applyBorder="1"/>
13     <xf numFmtId="4" fontId="0" fillId="4" borderId="0" xfId="0"
14  applyNumberFormat="1" applyFill="1" applyBorder="1"/>
15     <xf numFmtId="4" fontId="0" fillId="4" borderId="1" xfId="0"
16  applyNumberFormat="1" applyFill="1" applyBorder="1"/>
17      </cellXfs>
18      <cellStyles count="2">
19          <cellStyle name="Accent1" xfId="1" builtinId="29"/>
20          <cellStyle name="Normal" xfId="0" builtinId="0"/>
21      </cellStyles>
22      <dxf count="7">
23          <dxf>
24              <fill>
25                  <patternFill patternType="solid">
26                      <fgColor theme="0" tint="-0.14999847407452621"/>
27                      <bgColor theme="0" tint="-0.14999847407452621"/>
28                  </patternFill>
29              </fill>
30          </dxf>
31          <dxf>
32              <fill>
33                  <patternFill patternType="solid">
34                      <fgColor theme="0" tint="-0.14999847407452621"/>
35                      <bgColor theme="0" tint="-0.14999847407452621"/>
36                  </patternFill>
37              </fill>
38          </dxf>
39          <dxf>
40              <font>
41                  <b/>
42                  <color theme="0"/>
43              </font>

```

```
1          <fill>
2              <patternFill patternType="solid">
3                  <fgColor theme="5"/>
4                  <bgColor theme="5"/>
5              </patternFill>
6          </fill>
7      </dxf>
8      <dxf>
9          <font>
10             <b/>
11             <color theme="0"/>
12         </font>
13         <fill>
14             <patternFill patternType="solid">
15                 <fgColor theme="5"/>
16                 <bgColor theme="5"/>
17             </patternFill>
18         </fill>
19     </dxf>
20     <dxf>
21         <border>
22             <top style="double">
23                 <color theme="1"/>
24             </top>
25         </border>
26     </dxf>
27     <dxf>
28         <font>
29             <b/>
30             <color theme="0"/>
31         </font>
32         <fill>
33             <patternFill patternType="solid">
34                 <fgColor theme="5"/>
35                 <bgColor theme="5"/>
36             </patternFill>
37         </fill>
38         <border>
39             <bottom style="medium">
40                 <color theme="1"/>
41             </bottom>
42         </border>
43     </dxf>
```

```

1      <dxf>
2          <font>
3              <color theme="1"/>
4          </font>
5          <border>
6              <top style="medium">
7                  <color theme="1"/>
8              </top>
9              <bottom style="medium">
10                 <color theme="1"/>
11             </bottom>
12         </border>
13     </dxf>
14 </dxfs>
15 <tableStyles count="1" defaultTableStyle="TableStyleMedium9"
16 defaultPivotStyle="PivotStyleLight16">
17     <tableStyle name="TableStyleMedium10 - Custom" pivot="0" count="7">
18         <tableStyleElement type="wholeTable" dxfId="6"/>
19         <tableStyleElement type="headerRow" dxfId="5"/>
20         <tableStyleElement type="totalRow" dxfId="4"/>
21         <tableStyleElement type="firstColumn" dxfId="3"/>
22         <tableStyleElement type="lastColumn" dxfId="2"/>
23         <tableStyleElement type="firstRowStripe" dxfId="1"/>
24         <tableStyleElement type="firstColumnStripe" dxfId="0"/>
25     </tableStyle>
26 </tableStyles>
27 <colors/>
28 </styleSheet>

```

#### 3.7.4.4 Cell D2 Formatting

30

	A	B	C	D
1				
2			Q1	
3	Revenue		412.52	
4	Expenses		697.37	
5	Total		\$1,109.89	

31 Cell D2 contains the text "Q1" and is defined in the cell table of sheet1 as:

```

32 <c r="D2" s="7" t="s">
33     <v>0</v>
34 </c>

```

1 On this cell, the attribute value `s="7"` indicates that the 7th (zero-based) `<xf>` definition of `<cellXfs>` holds  
 2 the formatting information for the cell. The 7th `<xf>` of `<cellXfs>` is defined as:

3     `<xf numFmtId="0" fontId="4" fillId="2" borderId="2" xfId="1" applyBorder="1"/>`

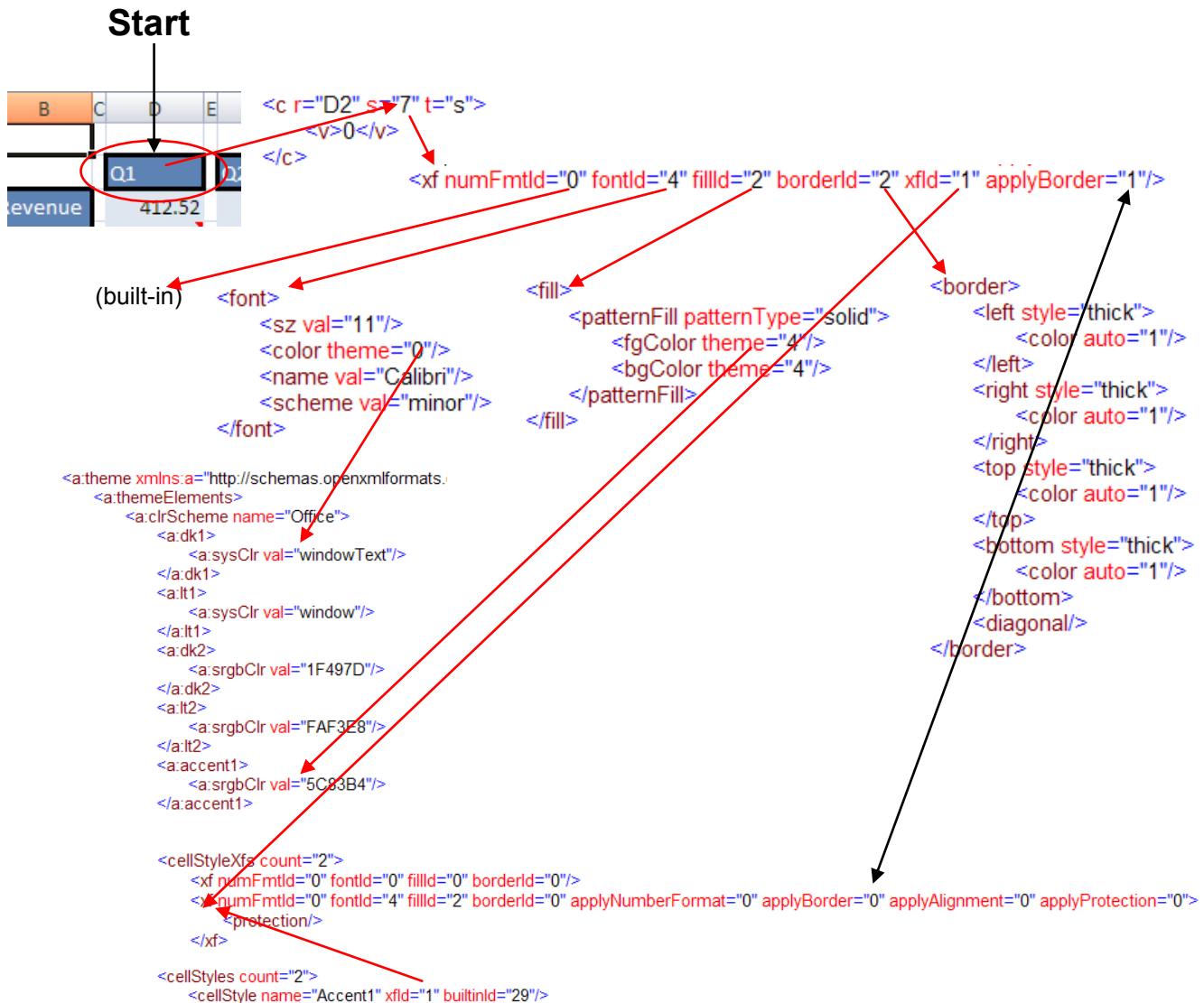
4 The number formatting information cannot be found in a `<numFmt>` definition because it is a built-in format;  
 5 instead, it is implicitly understood to be the 0th built-in number format. Remembering that the indexes to  
 6 other element collections are also zero-based, the font information can be found in the 4th `<font>` definition;  
 7 the fill information in the 2nd `<fill>` definition; and the border information in the 2nd `<border>` definition.  
 8 The cell uses a cell style which is defined in the 1st `<cellStyleXf>` definition and, finally, borders specified in  
 9 this master formatting record should be applied.

10 Remember that these collections are zero-based.

11 Additionally the `<fill>` definition for D2 references a themed color, whose index is 4th in the `<clrScheme>`  
 12 definition of the theme part:

```
13     <fill>
14         <patternFill patternType="solid">
15             <fgColor theme="4"/>
16             <bgColor theme="4"/>
17         </patternFill>
18     </fill>
```

19 Graphically, the index references can be shown like this:



## 3.7.4.5 Custom Table Style

L	M	N	O
Column1	Column2	Column3	Column4
A	381.41	513.27	357.29
B	470.33	411.76	723.52
C	624.47	287.49	365.38
D	17.77	775.36	969.69

- 4 This range of cells is a Table object with a custom Table style applied. The table definition in table1 specifies  
 5 which table style is applied, and which aspects of the table style definition are 'turned on' and should be  
 6 applied:

```

1   <table id="2" name="Table11" displayName="Table11" ref="L20:024"
2     totalsRowShown="0">
3       <tableStyleInfo name="TableStyleMedium10 - Custom" showFirstColumn="0"
4     showLastColumn="0" showRowStripes="1" showColumnStripes="0"/>
5   </table>
6
7   The <tableStyleInfo> element indicates that this Table uses the "TableStyleMedium10 - Custom" style
8   and that "first column", "last column", and "column stripes" formatting are OFF. It also indicates that "row
9   stripes" formatting is ON.

```

9 Here is the "TableStyleMedium10 - Custom" definition in the Styles part:

```

10      <tableStyles count="1" defaultTableStyle="TableStyleMedium9"
11        defaultPivotStyle="PivotStyleLight16">
12          <tableStyle name="TableStyleMedium10 - Custom" pivot="0" count="7">
13            <tableStyleElement type="wholeTable" dxfId="6"/>
14            <tableStyleElement type="headerRow" dxfId="5"/>
15            <tableStyleElement type="totalRow" dxfId="4"/>
16            <tableStyleElement type="firstColumn" dxfId="3"/>
17            <tableStyleElement type="lastColumn" dxfId="2"/>
18            <tableStyleElement type="firstRowStripe" dxfId="1"/>
19            <tableStyleElement type="firstColumnStripe" dxfId="0"/>
20          </tableStyle>
21        </tableStyles>
22        <colors/>
23      </styleSheet>

```

24 Note that even though column stripes are defined for this table style, they are not used for this instance of the  
25 table.

26 The header row formatting for this table is defined by the 5th <dxf> definition:

```

27      <dxf>
28        <font>
29          <b/>
30          <color theme="0"/>
31        </font>
32        <fill>
33          <patternFill patternType="solid">
34            <fgColor theme="5"/>
35            <bgColor theme="5"/>
36          </patternFill>
37        </fill>
38        <border>
39          <bottom style="medium">

```

```
1           <color theme="1"/>
2           </bottom>
3       </border>
4   </dxfs>
```

5 This formatting indicates that for the header row of this table, the font is bold face and uses a themed color;  
6 the fill is solid and uses a themed color; and there is a bottom border on the cells.

## 7 **3.8 Worksheet Metadata**

### 8 **3.8.1 Overview**

9 Value and cell metadata are additional properties that can be associated with a particular cell or value. Cell  
10 metadata properties can be carried along with the cell as it moves (e.g., via insert, shift, copy/paste, merge, or  
11 unmerge) and value metadata properties can be propagated along with the value as it is referenced in  
12 formulas.

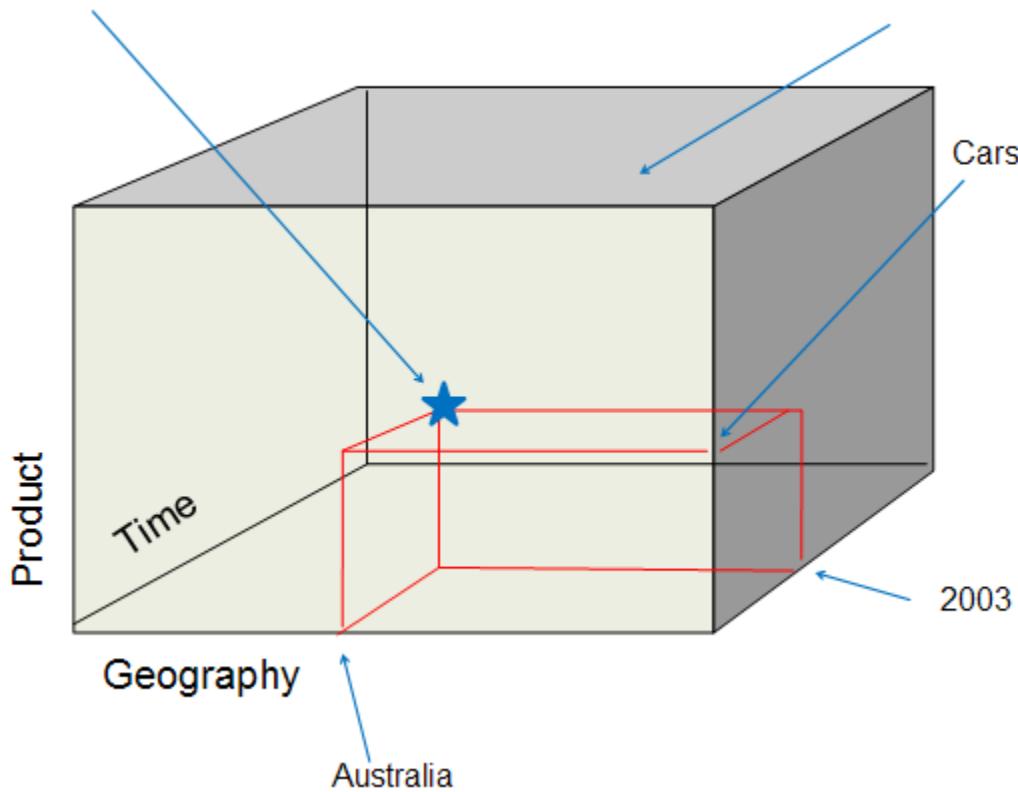
13 All of this metadata is stored separately in the metadata.xml part in the workbook.

14 While the architecture of this feature allows for future extensions, only MDX metadata—metadata that is  
15 associated with a particular cube function and its results—is currently defined. For example, if a CUBEMEMBER  
16 function call is used to identify a particular member in an OLAP cube, then the metadata would express the  
17 OLAP connection name, the mdx expression identifying that member, and various operational attributes of  
18 that metadata (e.g., whether it propagates through formula assignment, shifts with the cell when the cell  
19 moves locations in the grid, and so on).

## 1 3.8.1.1 OLAP Cube Review

Car sales in Australia for 2003

Sales in USD



2

3 Consider the 3-dimensional OLAP cube above. The three *dimensions* of the cube are "Product", "Geography",  
 4 and "Time". "Sales Amount" is the *measure* being summarized, and is often considered an additional  
 5 dimension of the cube. OLAP cubes can be N-dimensional, while this one has three dimensions.

6 Within each dimension are *hierarchies*, or ways of organizing the dimension into various levels of granularity.  
 7 For example, within the Time dimension there can exist the Calendar Year / Quarter / Month / Day hierarchy.  
 8 Likewise, the Time dimension can also have the Fiscal Year / Quarter / Month / Day hierarchy. Each of Year /  
 9 Quarter / Month / Day represents *levels* in the hierarchy. '2003' is considered a *member* of the 'Year' level  
 10 within the hierarchy. Likewise, the Product dimension can have multiple hierarchies. A hierarchy could be  
 11 constructed based on the type of product while another hierarchy could be constructed based on the color of  
 12 the product.

13 In the example above, picking a member from one dimension would be visualized as a slice through the cube.  
 14 For example, picking 'Australia' from the Geography dimension could be a relatively thick slice of the cube, if  
 15 there were many levels underneath 'Country', like 'State', 'City', and 'PostalCode'. Picking a member from  
 16 Geography that is more granular than 'Australia" results in a thinner slice of the cube in the Geography  
 17 dimension, because now some of Australia will have been omitted from the data.

18 A *tuple* is the intersection of two or more members from distinct dimensions. In the example above, three  
 19 members from three dimensions are expressed:

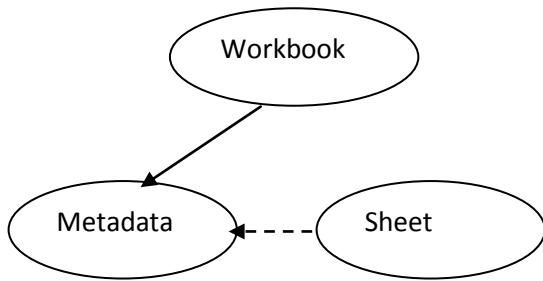
- 1        1. From Geography we have [Geography].[Country].&[Australia]  
 2        2. From Product we have [Product].[Cars].children  
 3        3. From Time we have [Time].[Calendar].[Calendar Year].&[2003]

4        **3.8.1.2      OLAP Function Summary**

5        There are seven recognized “CUBE” functions that can be used in cell formulas. These functions enable  
 6        formulas in any cell to fetch data from Analysis Services. Specifically, the functions can fetch any member, set,  
 7        aggregated value, property, or KPI from the OLAP cube. Because it is formula-driven, the layout of this data is  
 8        as flexible as cells and formulas. The OLAP data can be placed anywhere on the spreadsheet, intermingled with  
 9        other local calculations or within other formulas.

10      The function names are: CUBEKPIMEMBER, CUBEMEMBER, CUBEMEMBERPROPERTY, CUBERANKEDMEMBER,  
 11      CUBESET, CUBESETCOUNT, and CUBEVALUE.

12      **3.8.2      File Architecture – Relationships**



18      The workbook holds the relationship to the metadata part, and cells within a sheet reference the items in the  
 19      metadata part.

20      **3.8.3      Example**

21      In the following example, Cube functions are used to build up a report of internet sales by country, for all  
 22      products, for the calendar years 2003 and 2004.

### 1 3.8.3.1 Illustration

Diagram illustrating the execution flow of a CUBESET formula:

```

=CUBESET("xlextdat9 Adventure Works", "[Customer].[Customer Geography].[All Customers].children", "Countries", 2, D5)
  
```

The formula generates the following table:

	A	B	C	D
1		Internet Sales Amount		
2		CY 2003	CY 2004	
3		All Products		
4				
5	Countries	6 2003 Sales	2004 Sales	
6	United States	\$2,838,512.36	\$3,324,031.16	=CUBEMEMBER("xlextdat9 Adventure Works", C1:C3, "2003 Sales")
7	Australia	\$3,033,784.21	\$2,563,884.29	=CUBEMEMBER("xlextdat9 Adventure Works", C1:D2,C3, "2004 Sales")
8	United Kingdom	\$1,298,248.57	\$1,210,286.27	
9	Germany	\$1,058,405.73	\$1,076,890.77	
10	France	\$1,026,324.97	\$922,179.04	
11	Canada	\$535,784.46	\$673,628.21	

Annotations highlight specific cells and formulas:

- C1:C3, "2003 Sales"**: Points to the range C1:C3.
- C1:D2,C3, "2004 Sales"**: Points to the range C1:D2,C3.
- D5, ROW(A1)**: Points to cell D5 and the formula =CUBERANKEDMEMBER(...).
- D\$5,\$A6**: Points to cells D5 and A6, which are part of the formula =CUBEVALUE(...).
- C5\$5,SA6**: Points to cells C5 and A6, which are part of the formula =CUBEVALUE(...).

2

### 3.8.3.1.1 Function Summary

- 4 CUBEMEMBER(*connection*, *member-expression*, *caption*) returns a member in the cube (e.g., Bicycles, Cars, All  
5 Products, CY 2004) (member can be a tuple)

6 CUBESET(*connection*, *set-expression*, *caption*, *sort-order*, *sort-by*) returns a set of members (e.g., all  
7 Countries)

8 CUBERANKEDMEMBER(*connection*, *set-expression*, *rank*, *caption*) returns one member of the referenced set  
9 (e.g., Australia)

10 CUBEVALUE(*connection*, *member-expression-1*, *member-expression-2*, ...) returns the aggregate summarized  
11 value for the intersection of members specified.

12 **3.8.3.1.2 Walk Through**

13 C1 contains a CUBEMEMBER function call specifying the "Internet Sales Amount" member from the Measures  
14 dimension. C2 contains a CUBEMEMBER function call specifying "CY 2003" from the Date dimension. D2  
15 contains a similar function specifying "CY2004" from the Date dimension. C3 contains a CUBEMEMBER function  
16 call specifying "All Products" from the Product dimension. Each of these cells contain simple string values (e.g.,  
17 "All Products" in C3), and each of these cells is associated with mdx metadata which specifies the mdx  
18 expression identifying a particular member of a particular dimension (e.g., [Measures].[Internet Sales  
19 Amount]).

1 A5 contains a CUBESET function call specifying a set of members. Additionally, the CUBESET function call  
 2 allows for specifying a caption for the cell ("Countries"), a sort order for sorting the set (in this case,  
 3 "2" corresponds to descending), and a sort by field (in this case the set will be sorted by the member as  
 4 expressed in the mdx associated with cell D5, labeled "2004 Sales"). Finally, it should be noted that CUBESET  
 5 returns a set of members, not just a single member.

6 Cells A6:A11 use the CUBERANKEDMEMBER function to return the individual members, by rank, returned from  
 7 the CUBESET function call in A5. For example, A6 uses the "xlextdat9 Adventure Works" connection to connect  
 8 to the OLAP cube, and addresses the first member (because "ROW(A1)" resolves to "1") in the set returned  
 9 in A5.

10 Cell C6 uses the CUBEVALUE function to return measure data identified by intersecting the mdx expression  
 11 found in A6 with the mdx expression found in C5 ("CY 2003 Internet Sales for All Products in the United  
 12 States"). C7:C11 use similar CUBEVALUE function calls. D6:D11 involves similar functions as well, but using  
 13 "CY 2004" instead.

14 The power of metadata in this example is that anytime a CUBE function argument referenced another cell, and  
 15 that argument requires a set or member mdx expression, the mdx metadata for that referenced cell is  
 16 returned to the calling function instead of the simple string value. For example, A6 contains a  
 17 CUBERANKEDMEMBER function call, whose second argument is expecting a set of members. The reference for  
 18 that argument is \$A\$5. Instead of using the A5's string value of "Countries" (which would result in a error),  
 19 A5's mdx expression metadata is used instead, which returns a set. Similarly any of the CUBEVALUE function  
 20 calls rely on cell references, where those cells contain mdx metadata used to pinpoint the measure data  
 21 requested. Furthermore, each of the cells referenced by C6:D11, in turn reference other cells' mdx metadata.  
 22 In this way, the mdx metadata is able to propagate through the formula calculation chain.

### 23        3.8.3.2           Worksheet Metadata XML

#### 24        3.8.3.2.1       General Organization

```
25 <metadata>
26   <metadataTypes/>
27   <metadataStrings/>
28   <mdxMetadata/>
29   <valueMetadata/>
30 </metadata>
```

31 There are four general collections in the metadata part:

- 32     • metadataTypes - expresses various application runtime behaviors that apply to a set of metadata.
- 33     • metadataStrings - expresses supporting string resources for the metadata part. This includes the  
       connection name to the OLAP cube as well as mdx expressions identifying members and sets.
- 34     • mdxMetadata - expresses the tuples in use in this workbook.
- 35     • valueMetadata - The block (bk) elements stored in valueMetadata are referenced from cells in the  
       sheet definition (vm on the cell is an index (1-based) to a bk element). Each record in a block

1 references additional element collections in the metadata part to define fully the metadata associated  
 2 with this particular record, and therefore the full metadata definition for a particular cell's value. The  
 3 records in valueMetadata serve as the bridge between the metadata definitions and the cells or values  
 4 in the sheet.

5    3.8.3.2.2                  Metadata Behaviors

6    The metadata type expresses operations on cells that allow the metadata to remain associated with the cell.  
 7    Operations not listed or set to '0' would cause the metadata to no longer be associated with the cell.

```
8 <metadataTypes count="1">
 9   <metadataType name="XLMDX" minSupportedVersion="120000" copy="1"
10     pasteAll="1" pasteValues="1" merge="1" splitFirst="1"
11     rowColShift="1" clearFormats="1" clearComments="1" assign="1"
12     coerce="1"/>
13 </metadataTypes>
```

14 Regarding metadataTypes:

- 15    • count is the number of metadataType elements.
- 16    • type is a particular set of cell operations.

17 Regarding metadataType:

- 18    • name is the name of this particular metadata type.
- 19    • minSupportedVersion indicates the earliest version of the application which supports this metadata  
20      type.
- 21    • copy value of 1 indicates that this metadata will be copied to other cells when the cell is copied.
- 22    • pasteAll value of 1 indicates that this metadata will be pasted to another cell when 'paste all' is chosen  
23      during a copy/paste operation.
- 24    • pasteValues value of 1 indicates that this metadata will be pasted to another cell when only the  
25      values of the cell is pasted during a copy/paste operation.
- 26    • merge value of 1 indicates that when the cell is merged, the metadata associated with the cell  
27      remains.
- 28    • splitFirst value of 1 indicates that when a merged cell is split, the metadata associated with the  
29      merged cell is only applied to the first (from top left) cell resulting from the split.
- 30    • rowColShift value of 1 indicates that metadata associated with a cell remains after rows and columns  
31      are inserted, even when the cell is moved.
- 32    • clearFormats value of 1 indicates that the metadata remains after the cell has been cleared of all  
33      formatting.
- 34    • clearComments value of 1 indicates that the metadata remains after comments have been cleared  
35      from the cell.
- 36    • assign value of 1 indicates that the metadata propagates through formula assignment operations

- 1       • coerce value of 1 indicates that the metadata can be removed when the data type is coerced to  
 2       another type.

3     3.8.3.2.3            Metadata Strings

4     This collection is a set of string resources for the metadata part. Most follow the format of an mdx expression.  
 5     Connection names (to OLAP cubes) are also expressed here.

```
6       <metadataStrings count="12">
7        <s v="x1extdat9 Adventure Works"/>
8        <s v="[Measures].[Internet Sales Amount]"/>
9        <s v="[Date].[Calendar].[Calendar Year]&[2003]"/>
10       <s v="[Date].[Calendar].[Calendar Year]&[2004]"/>
11       <s v="[Product].[Product Categories].[All Products]"/>
12       <s v="[Customer].[Customer Geography].[All Customers].children"/>
13       <s v="[Customer].[Customer Geography].[Country]&[Australia]"/>
14       <s v="[Customer].[Customer Geography].[Country]&;
15           [United States]"/>
16       <s v="[Customer].[Customer Geography].[Country]&;
17           [United Kingdom]"/>
18       <s v="[Customer].[Customer Geography].[Country]&[Germany]"/>
19       <s v="[Customer].[Customer Geography].[Country]&[France]"/>
20       <s v="[Customer].[Customer Geography].[Country]&[Canada]"/>
21     </metadataStrings>
```

22   Regarding metadataStrings:

- 23       • count indicates the number of strings in the collection.  
 24       • s is the string container element  
 25       • v is the string value itself.

26     3.8.3.2.4            mdxMetadata

27     This collection expresses mdx metadata, and builds up the mdx members, sets, KPIs, and member properties.

28   valueMetadata records reference these records.

```
29       <mdxMetadata count="26">
30        <mdx n="0" f="m">
31          <t c="1">
32           <n x="1"/>
33           </t>
34        </mdx>
```

```
1  <mdx n="0" f="m">
2      <t c="1">
3          <n x="2"/>
4      </t>
5  </mdx>
6  <mdx n="0" f="m">
7      <t c="1">
8          <n x="3"/>
9      </t>
10 </mdx>
11 <mdx n="0" f="m">
12     <t c="1">
13         <n x="4"/>
14     </t>
15 </mdx>
16 <mdx n="0" f="m">
17     <t c="3">
18         <n x="1"/>
19         <n x="2"/>
20         <n x="4"/>
21     </t>
22 </mdx>
23 <mdx n="0" f="m">
24     <t c="3">
25         <n x="1"/>
26         <n x="3"/>
27         <n x="4"/>
28     </t>
29 </mdx>
30 <mdx n="0" f="r">
31     <t c="1">
32         <n x="6"/>
33     </t>
34 </mdx>
35 <mdx n="0" f="r">
36     <t c="1">
37         <n x="7"/>
38     </t>
39 </mdx>
```

```
1  <mdx n="0" f="r">
2      <t c="1">
3          <n x="8"/>
4      </t>
5  </mdx>
6  <mdx n="0" f="r">
7      <t c="1">
8          <n x="9"/>
9      </t>
10 </mdx>
11 <mdx n="0" f="r">
12     <t c="1">
13         <n x="10"/>
14     </t>
15 </mdx>
16 <mdx n="0" f="r">
17     <t c="1">
18         <n x="11"/>
19     </t>
20 </mdx>
21 <mdx n="0" f="v">
22     <t c="4" ct="en-US">
23         <n x="1"/>
24         <n x="2"/>
25         <n x="4"/>
26         <n x="6"/>
27     </t>
28 </mdx>
29 <mdx n="0" f="v">
30     <t c="4" ct="en-US">
31         <n x="1"/>
32         <n x="3"/>
33         <n x="4"/>
34         <n x="7"/>
35     </t>
36 </mdx>
```

```
1 <mdx n="0" f="v">
2   <t c="4" ct="en-US">
3     <n x="1"/>
4     <n x="2"/>
5     <n x="4"/>
6     <n x="7"/>
7   </t>
8 </mdx>
9 <mdx n="0" f="v">
10  <t c="4" ct="en-US">
11    <n x="1"/>
12    <n x="3"/>
13    <n x="4"/>
14    <n x="8"/>
15  </t>
16 </mdx>
17 <mdx n="0" f="v">
18  <t c="4" ct="en-US">
19    <n x="1"/>
20    <n x="2"/>
21    <n x="4"/>
22    <n x="8"/>
23  </t>
24 </mdx>
25 <mdx n="0" f="v">
26  <t c="4" ct="en-US">
27    <n x="1"/>
28    <n x="3"/>
29    <n x="4"/>
30    <n x="9"/>
31  </t>
32 </mdx>
33 <mdx n="0" f="v">
34  <t c="4" ct="en-US">
35    <n x="1"/>
36    <n x="2"/>
37    <n x="4"/>
38    <n x="9"/>
39  </t>
40 </mdx>
```

```
1 <mdx n="0" f="v">
2   <t c="4" ct="en-US">
3     <n x="1"/>
4     <n x="3"/>
5     <n x="4"/>
6     <n x="10"/>
7   </t>
8 </mdx>
9 <mdx n="0" f="v">
10  <t c="4" ct="en-US">
11    <n x="1"/>
12    <n x="2"/>
13    <n x="4"/>
14    <n x="10"/>
15  </t>
16 </mdx>
17 <mdx n="0" f="v">
18  <t c="4" ct="en-US">
19    <n x="1"/>
20    <n x="3"/>
21    <n x="4"/>
22    <n x="11"/>
23  </t>
24 </mdx>
25 <mdx n="0" f="v">
26  <t c="4" ct="en-US">
27    <n x="1"/>
28    <n x="2"/>
29    <n x="4"/>
30    <n x="11"/>
31  </t>
32 </mdx>
33 <mdx n="0" f="v">
34  <t c="4" ct="en-US">
35    <n x="1"/>
36    <n x="3"/>
37    <n x="4"/>
38    <n x="6"/>
39  </t>
40 </mdx>
```

```

1   <mdx n="0" f="s">
2     <ms ns="5" c="3" o="d">
3       <n x="1"/>
4       <n x="3"/>
5       <n x="4"/>
6     </ms>
7   </mdx>
8   <mdx n="0" f="c">
9     <ms ns="5" c="3" o="d">
10    <n x="1"/>
11    <n x="3"/>
12    <n x="4"/>
13    </ms>
14  </mdx>
15 </mdxMetadata>

```

16 Regarding mdxMetadata:

- 17 • count indicates the number of mdx statements in the collection.

18 Regarding mdx, which is a particular mdx statement:

- 19 • n indicates the index of the record in metadataStrings containing the connection name.  
20 • f indicates the name of the calling cube function in the workbook.

21 Regarding t, which is an mdx tuple:

- 22 • c is the count of member expressions in the mdx tuple.

23 Regarding n:

- 24 • x is the index value into metadataStrings indicating the particular member expression for this  
25 dimension of the tuple expression.

26 For example, cell C5 has a CUBEMEMBER function call expressing the result of "Internet Sales Amount of All  
27 Products for CY 2003". In sheet1.xml, cell C5 has vm="5", which means it has an associated valueMetadata  
28 record whose index is "5". Looking ahead into the valueMetadata records, the 5th (1-based) record points to  
29 the 4th (zero-based) mdx collection in mdxMetadata.

30 The 5th mdx collection:

```

1   <mdx n="0" f="m">
2     <t c="3">
3       <n x="1"/>
4       <n x="2"/>
5       <n x="4"/>
6     </t>
7   </mdx>
```

8 Where `<n x="1"/>` corresponds to the 1st position in the string store, namely

9 `<s v="[Measures].[Internet Sales Amount]"/>`

10 and where `<n x="2"/>` corresponds to the 2nd position in the string store, namely

11 `<s v="[Date].[Calendar].[Calendar Year].&[2003]"/>`

12 and where `<n x="4"/>` corresponds to the 4th position in the string store, namely

13 `<s v="[Product].[Product Categories].[All Products]"/>.`

14 Therefore this data point in the cube is addressed by intersecting these three hierarchies, one in each  
15 dimension of the OLAP cube:

- [Measures].[Internet Sales Amount]
- [Date].[Calendar].[Calendar Year].[2003]
- [Product].[Product Categories].[All Products]

19 Regarding ms:

- ns is the index of the mdx set definition in the string store.
- c is the number of sort-by member indicies, in this case 3 because the set is sorted by the contents of D5, which happens to be a member defined by 3 coordinates in the cube.
- o indicates the order of the sort; in this case, 'descending'.
- n is the index indicating the mdx expressions in the string store used to identify the members used to define the sort-by set.

### 26 3.8.3.2.5 valueMetadata

27 This collection defines cell or value metadata information (depending on the value of metadataType's  
28 cellMeta)

```

29 <valueMetadata count="26">
30   <bk>
31     <rc t="1" v="0"/>
32   </bk>
```

```
1  <bk>
2    <rcc t="1" v="1"/>
3  </bk>
4  <bk>
5    <rcc t="1" v="2"/>
6  </bk>
7  <bk>
8    <rcc t="1" v="3"/>
9  </bk>
10 <bk>
11   <rcc t="1" v="4"/>
12 </bk>
13 <bk>
14   <rcc t="1" v="5"/>
15 </bk>
16 <bk>
17   <rcc t="1" v="6"/>
18 </bk>
19 <bk>
20   <rcc t="1" v="7"/>
21 </bk>
22 <bk>
23   <rcc t="1" v="8"/>
24 </bk>
25 <bk>
26   <rcc t="1" v="9"/>
27 </bk>
28 <bk>
29   <rcc t="1" v="10"/>
30 </bk>
31 <bk>
32   <rcc t="1" v="11"/>
33 </bk>
34 <bk>
35   <rcc t="1" v="12"/>
36 </bk>
37 <bk>
38   <rcc t="1" v="13"/>
39 </bk>
40 <bk>
41   <rcc t="1" v="14"/>
42 </bk>
```

```

1   <bk>
2     <rc t="1" v="15"/>
3   </bk>
4   <bk>
5     <rc t="1" v="16"/>
6   </bk>
7   <bk>
8     <rc t="1" v="17"/>
9   </bk>
10  <bk>
11    <rc t="1" v="18"/>
12  </bk>
13  <bk>
14    <rc t="1" v="19"/>
15  </bk>
16  <bk>
17    <rc t="1" v="20"/>
18  </bk>
19  <bk>
20    <rc t="1" v="21"/>
21  </bk>
22  <bk>
23    <rc t="1" v="22"/>
24  </bk>
25  <bk>
26    <rc t="1" v="23"/>
27  </bk>
28  <bk>
29    <rc t="1" v="24"/>
30  </bk>
31  <bk>
32    <rc t="1" v="25"/>
33  </bk>
34  </valueMetadata>
```

35 Regarding valueMetadata:

- 36 • count indicates the number of metadata block records.

37 Regarding bk, which is a metadata block, and rc, which is a metadata record:

- 38 • t indicates the index of the metadataType record in metadataTypes collection.
- 39 • v is the index of metadata record value in the storage corresponding to record type.

1 Looking at the first block using the `bk` element, the type of metadata with which this record is associated is the  
 2 first (and only) metadataType record, which is of type "XLMDX". This indicates that the `v` index is pointing to  
 3 the 0th mdxMetadata record.

## 4 3.9 Pivot Table, Pivot Cache, and Common Types

### 5 3.9.1 Feature Overview

6 PivotTables display aggregated views of data easily and in an understandable layout. Hundreds or thousands of  
 7 pieces of underlying information can be aggregated on row & column axes, revealing the meanings behind the  
 8 data. PivotTable reports are used to organize and summarize your data in different ways. Creating a PivotTable  
 9 report is about moving pieces of information around to see how they fit together. In a few gestures the pivot  
 10 rows and columns can be moved into different arrangements and layouts.

11 A PivotTable object has a row axis area, a column axis area, a values area, and a report filter area. Additionally,  
 12 PivotTables have a corresponding field list pane displaying all the fields of data which can be placed on one of  
 13 the PivotTable areas.

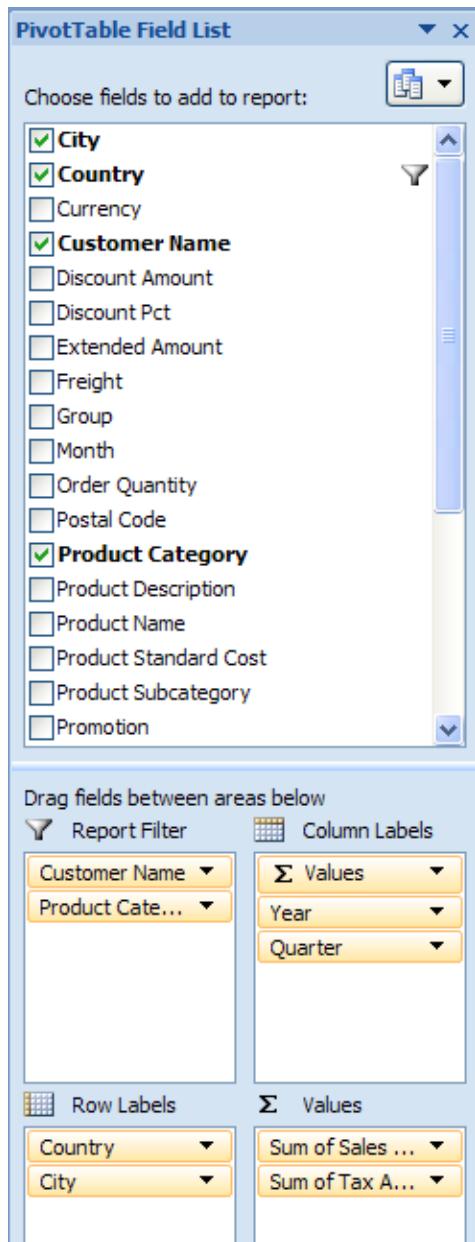
14 Consider this source data:

	A	C	F	H	I	O	P	Q	Z	AA	AB
1	Customer Name	Country	City	Product Category	Product Subcategory	Year	Quarter	Month	Sales Amount	Tax Amount	Freight
2	Michele Raman	Australia	Bendigo	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
3	Misty Raji	Australia	Bendigo	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568
4	Tabitha E Arthur	Australia	Bendigo	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568
5	Clarence D Rai	Australia	Bendigo	Bikes	Mountain Bikes	2001		3 July	3399.99	271.9992	84.9998
6	Jimmy L Moreno	Australia	Bendigo	Bikes	Mountain Bikes	2001		3 July	3399.99	271.9992	84.9998
7	Rob Verhoff	Australia	Bendigo	Bikes	Mountain Bikes	2001		3 July	3374.99	269.9992	84.3748
8	Levi Sai	Australia	Bendigo	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568
9	Logan Gonzales	Australia	Brisbane	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568
10	Dalton J Lee	Australia	Brisbane	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
11	Jessie J Ortega	Australia	Brisbane	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
12	Paul J. Shakespear	Australia	Caloundra	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
13	Joan R Martin	Australia	Caloundra	Bikes	Road Bikes	2001		3 September	699.0982	55.9279	17.4775
14	Casey Pal	Australia	Caloundra	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568
15	Ethan G Coleman	Australia	Caloundra	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
16	Kendra Rubio	Australia	Caloundra	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
17	Bethany G Yuan	Australia	Cloverdale	Bikes	Mountain Bikes	2001		3 August	3399.99	271.9992	84.9998
18	Jasmine Wilson	Australia	Coffs Hart	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
19	Micah Wu	Australia	Coffs Hart	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
20	Warren L Zhang	Australia	Coffs Hart	Bikes	Road Bikes	2001		3 July	699.0982	55.9279	17.4775
21	Ariana Stewart	Australia	Coffs Hart	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
22	Suzanne K Lu	Australia	Coffs Hart	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
23	Randall M Rubio	Australia	Cranbourn	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
24	Deborah K Kumar	Australia	Cranbourn	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
25	Krystal Holt	Australia	Cranbourn	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568
26	Patricia T Raman	Australia	Cranbourn	Bikes	Road Bikes	2001		3 August	3578.27	286.2616	89.4568
27	Wendy Dominguez	Australia	Cranbourn	Bikes	Mountain Bikes	2001		3 August	3374.99	269.9992	84.3748
28	Willie She	Australia	Darlinghu	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
29	Alan Zhu	Australia	Darlinghu	Bikes	Road Bikes	2001		3 September	3578.27	286.2616	89.4568
15	Dawn R Tang	Australia	Darlinehu	Bikes	Road Bikes	2001		3 July	3578.27	286.2616	89.4568

16 This data can be consolidated and summarized in a PivotTable. One way to organize the information would  
 17 look like this:

	A	B	C	D	E	F	G	H
1								
2		Customer Name (All)						
3		Product Category (All)						
4								
5		Column Labels						
6		Sum of Sales Amount			Sum of Tax Amount			
7		2001		2001 Total	2001		2001	
8	Row Labels		3	4		3	4	
9	Australia	606184.7066	702862.4912	1309047.198	48494.7771	56229	104723.7771	
10	Goulburn	40580.8864	25047.89	65628.7764	3246.471	2003.8312	5250.3022	
11	Warrnambool	28091.32	27863.04	55954.36	2247.3056	2229.0432	4476.3488	
12	Port Macquarie	25746.9882	24463.05	50210.0382	2059.7591	1957.044	4016.8031	
13	Wollongong	24691.33	25390.4282	50081.7582	1975.3064	2031.2343	4006.5407	
14	Bendigo	24488.05	17484.79	41972.84	1959.044	1398.7832	3357.8272	
15	Geelong	21113.06	21088.06	42201.12	1689.0448	1687.0448	3376.0896	
16	Hobart	21088.06	6953.26	28041.32	1687.0448	556.2608	2243.3056	
17	Lavender Bay	21063.06	21965.4382	43028.4982	1685.0448	1757.2351	3442.2799	
18	Matraville	20909.78	6799.98	27709.76	1672.7824	543.9984	2216.7808	

- 2 Here is the corresponding PivotTable field list:



1

## 2 3.9.2 File Architecture

- 3 The workbook points to (and owns the longevity of) the pivotCacheDefinition part, which in turn points to and  
 4 owns the pivotCacheRecords part. The workbook also points to and owns the sheet part, which in turn points  
 5 to and owns a pivotTable part definition, when a PivotTable is on the sheet (there can be multiple PivotTables  
 6 on a sheet). The pivotTable part points to the appropriate pivotCacheDefinition which it is using. Since multiple  
 7 PivotTables can use the same cache, the pivotTable part does not own the longevity of the  
 8 pivotCacheDefinition.
- 9 The pivotTable part describes the particulars of the layout of the PivotTable on the sheet. It indicates what  
 10 fields are on the row axis, the column axis, report filter, and values areas of the PivotTable. It also indicates

formatting information about the PivotTable. If conditional formatting has been applied to the PivotTable, that is also expressed in the pivotTable part.

- 3 The pivotCacheRecords part contains the underlying data to be aggregated. It is a cache of the source data.
- 4 The pivotCacheDefinition part defines each field in the pivotCacheRecords part, including field name and
- 5 information about the data contained in the field. The pivotCacheDefinition part also defines pivot items that
- 6 are shared among the pivotTable and pivotRecords parts.

### 3.9.3 Example - Native with Range Source

### 8      3.9.3.1              Illustration

9 Consider the source data pictured in the overview section. There are 28 fields of data in total (some aren't  
10 shown). A corresponding PivotTable summary of the data can look like this:

A	B	C	D	E	F	G
1						
2	Country	(All)				
3	State	(All)				
4	City	(All)				
5						
6	Sum of Sales Amount	Column Labels				
7		2001				2001 Total
8			3			3 Total
9	Row Labels	July	August	September		
10	Bikes	209652.9046	222538.2892	173993.5128	606184.7066	606184.7066
11	Mountain Bikes	64424.81	60899.82	10174.97	135499.6	135499.6
12	Road Bikes	145228.0946	161638.4692	163818.5428	470685.1066	470685.1066
13	Grand Total	209652.9046	222538.2892	173993.5128	606184.7066	606184.7066

Regarding the layout of the PivotTable, notice that "Country", "State", and "City" are in the report filter area of the PivotTable. "Product Category" and "Product Subcategory" are on the row axis ("Bikes" belongs to the "Product Category" field and both "Mountain Bikes" and "Road Bikes" belong to the "Product Subcategory" field). On the column axis are "Year" ("2001"), "Quarter" ("3"), and "Month" ("July", "August", and "September") fields.

17 Row Grand Totals are turned on, and column Subtotals are turned on for Quarter and Year (if there was more  
18 than 1 quarter in the source data the Year Subtotal would be more interesting).

### 3.9.3.2 XML - pivotCacheDefinition part

20 The pivotCacheDefinition part defines each field in the source data, including the name, the string resources of  
21 the instance data (for shared items), and information about the type of data appearing in the field. Note: some  
22 of the "Customer Name" and "City" values have been removed to improve readability and reduce length.

```
1 <pivotCacheDefinition xmlns:r="..." r:id="rId1" refreshedBy="AnonUser"
2   refreshedDate="2006-05-22T10:07:16" createdVersion="3"
3   refreshedVersion="3" minRefreshableVersion="3" recordCount="182">
4   <cacheSource type="worksheet">
5     <worksheetSource name="Table1"/>
6   </cacheSource>
7   <cacheFields count="28">
8     <cacheField name="Customer Name" numFmtId="0">
9       <sharedItems count="7">
10      <s v="Michele Raman"/>
11      <s v="Misty Raji"/>
12      <s v="Tabitha E Arthur"/>
13      <s v="Clarence D Rai"/>
14      <s v="Jimmy L Moreno"/>
15      <s v="Rob Verhoff"/>
16      <s v="Levi Sai"/>
17    </sharedItems>
18  </cacheField>
19  <cacheField name="Group" numFmtId="0">
20    <sharedItems/>
21  </cacheField>
22  <cacheField name="Country" numFmtId="0">
23    <sharedItems count="1">
24      <s v="Australia"/>
25    </sharedItems>
26  </cacheField>
27  <cacheField name="Region" numFmtId="0">
28    <sharedItems/>
29  </cacheField>
30  <cacheField name="State" numFmtId="0">
31    <sharedItems count="5">
32      <s v="Victoria"/>
33      <s v="Queensland"/>
34      <s v="South Australia"/>
35      <s v="New South Wales"/>
36      <s v="Tasmania"/>
37    </sharedItems>
38  </cacheField>
```

```
1 <cacheField name="City" numFmtId="0">
2   <sharedItems count="7">
3     <s v="Bendigo"/>
4     <s v="Brisbane"/>
5     <s v="Caloundra"/>
6     <s v="Cloverdale"/>
7     <s v="Coffs Harbour"/>
8     <s v="Cranbourne"/>
9     <s v="Darlinghurst"/>
10    </sharedItems>
11  </cacheField>
12  <cacheField name="Postal Code" numFmtId="0">
13    <sharedItems/>
14  </cacheField>
15  <cacheField name="Product Category" numFmtId="0">
16    <sharedItems count="1">
17      <s v="Bikes"/>
18    </sharedItems>
19  </cacheField>
20  <cacheField name="Product Subcategory" numFmtId="0">
21    <sharedItems count="2">
22      <s v="Road Bikes"/>
23      <s v="Mountain Bikes"/>
24    </sharedItems>
25  </cacheField>
26  <cacheField name="Product Name" numFmtId="0">
27    <sharedItems/>
28  </cacheField>
29  <cacheField name="Product Description" numFmtId="0">
30    <sharedItems/>
31  </cacheField>
32  <cacheField name="Promotion Category" numFmtId="0">
33    <sharedItems/>
34  </cacheField>
35  <cacheField name="Promotion" numFmtId="0">
36    <sharedItems/>
37  </cacheField>
38  <cacheField name="Promotion Type" numFmtId="0">
39    <sharedItems/>
40  </cacheField>
```

```
1 <cacheField name="Year" numFmtId="0">
2   <sharedItems count="1">
3     <s v="2001"/>
4   </sharedItems>
5 </cacheField>
6 <cacheField name="Quarter" numFmtId="0">
7   <sharedItems containsSemiMixedTypes="0" containsString="0"
8     containsNumber="1" containsInteger="1" minValue="3" maxValue="3"
9     count="1">
10    <n v="3"/>
11  </sharedItems>
12 </cacheField>
13 <cacheField name="Month" numFmtId="0">
14   <sharedItems count="3">
15     <s v="September"/>
16     <s v="July"/>
17     <s v="August"/>
18   </sharedItems>
19 </cacheField>
20 <cacheField name="Currency" numFmtId="0">
21   <sharedItems/>
22 </cacheField>
23 <cacheField name="Order Quantity" numFmtId="0">
24   <sharedItems containsSemiMixedTypes="0" containsString="0"
25     containsNumber="1" containsInteger="1" minValue="1"
26     maxValue="1"/>
27 </cacheField>
28 <cacheField name="Unit Price" numFmtId="0">
29   <sharedItems containsSemiMixedTypes="0" containsString="0"
30     containsNumber="1" minValue="699.0982000000002"
31     maxValue="3578.27"/>
32 </cacheField>
33 <cacheField name="Extended Amount" numFmtId="0">
34   <sharedItems containsSemiMixedTypes="0" containsString="0"
35     containsNumber="1" minValue="699.0982000000002"
36     maxValue="3578.27"/>
37 </cacheField>
38 <cacheField name="Discount Pct" numFmtId="0">
39   <sharedItems containsSemiMixedTypes="0" containsString="0"
40     containsNumber="1" containsInteger="1" minValue="0"
41     maxValue="0"/>
42 </cacheField>
```

```

1   <cacheField name="Discount Amount" numFmtId="0">
2     <sharedItems containsSemiMixedTypes="0" containsString="0"
3       containsNumber="1" containsInteger="1" minValue="0"
4       maxValue="0"/>
5   </cacheField>
6   <cacheField name="Product Standard Cost" numFmtId="0">
7     <sharedItems containsSemiMixedTypes="0" containsString="0"
8       containsNumber="1" minValue="413.1463"
9       maxValue="2171.294199999998"/>
10  </cacheField>
11  <cacheField name="Total Product Cost" numFmtId="0">
12    <sharedItems containsSemiMixedTypes="0" containsString="0"
13      containsNumber="1" minValue="413.1463"
14      maxValue="2171.294199999998"/>
15  </cacheField>
16  <cacheField name="Sales Amount" numFmtId="0">
17    <sharedItems containsSemiMixedTypes="0" containsString="0"
18      containsNumber="1" minValue="699.0982000000002"
19      maxValue="3578.27"/>
20  </cacheField>
21  <cacheField name="Tax Amount" numFmtId="0">
22    <sharedItems containsSemiMixedTypes="0" containsString="0"
23      containsNumber="1"
24      minValue="55.92790000000001" maxValue="286.2615999999999"/>
25  </cacheField>
26  <cacheField name="Freight" numFmtId="0">
27    <sharedItems containsSemiMixedTypes="0" containsString="0"
28      containsNumber="1" minValue="17.47749999999999"
29      maxValue="89.45680000000001"/>
30  </cacheField>
31 </cacheFields>
32 </pivotCacheDefinition>

```

33 In the context of pivotCacheDefinition:

- 34 • r:id indicates the relationship id pointing to the corresponding pivotCacheRecords part.
- 35 • refreshedBy indicates the username of whomever last refreshed the PivotCache.
- 36 • refreshedDate indicates when the PivotCache was last refreshed.
- 37 • createdVersion indicates the version of the producer which created the PivotCache.
- 38 • refreshedVersion indicates the version of the producer which last refreshed the PivotCache.
- 39 • minRefreshableVersion indicates the minimum version of the producer required to be able to refresh this PivotCache.

41 In the context of cacheSource:

- 1       • type indicates that data in a worksheet is the source for this PivotCache.  
 2       • worksheetSource identifies the particular location of the source data. In this case, it is a named range  
 3       whose name is "Table1".

4     In the context of cacheFields, which is a collection of all the field definitions in the source data:

- 5       • cacheField indicates the name of the field and provides number format information.

6     In the context of cacheField:

- 7       • sharedItems indicates various flags about the data in this field. Child elements express the values of  
 8       the shared items.

9     In the context of sharedItems:

- 10       • containsSemiMixedTypes "1" indicates that this field contains text values possibly mixed with other  
   11       types of values, this can contain blanks. In this example the value is "0".  
   12       • containsString value of "1" indicates that this field contains a text value. In this example, the value  
   13       is "0".  
   14       • containsNumber value of "1" indicates that this field contains numeric values.  
   15       • containsInteger indicates that this field contains integer values.  
   16       • minValue indicates that this field's minimum value is "3".  
   17       • maxValue indicates that this field's maximum value is "3".  
   18       • s indicates string content for this item value (expressed in v).  
   19       • n indicates the numeric content for this item value (expressed in v).

20     If there are no shared items expressed for a particular field, then the values are expressed directly in the  
 21     pivotCacheRecords part.

22     Items in the PivotCacheDefinition can be shared, in order to reduce the redundancy of those values, since  
 23     they're referenced in multiple places across all the PivotTable parts. For example, a value might be part of a  
 24     filter, it might appear on a row or column axis, and will appear in the pivotCacheRecords definition as well.  
 25     However, because of the performance cost of creating the optimized shared items, items are only shared if  
 26     they are actually in use in the PivotTable. Therefore, depending on user actions on the PivotTable layout, the  
 27     pivotCacheDefinition and underlying PivotCacheRecords part may be updated.

### 28     3.9.3.3           XML - pivotCacheRecords part

29     This part expresses the underlying source data that the PivotTable is aggregating. (Note that the data has been  
 30     trimmed down to two records to increase readability.)

```
31       <pivotCacheRecords ... xmlns:r="..." count="2">
  32        <r>
  33          <x v="0"/>
  34          <s v="Pacific"/>
```

```

1      <x v="0"/>
2      <s v="Australia"/>
3      <x v="0"/>
4      <x v="0"/>
5      <s v="3550"/>
6      <x v="0"/>
7      <x v="0"/>
8      <s v="Road-150 Red, 62"/>
9      <s v="This bike is ridden by race winners. Developed with the Adventure
10 Works Cycles professional race team, it has a extremely light heat-treated
11 aluminum frame, and steering that allows precision control."/>
12     <s v="No Discount"/>
13     <s v="No Discount"/>
14     <s v="No Discount"/>
15     <x v="0"/>
16     <x v="0"/>
17     <x v="0"/>
18     <s v="Australian Dollar"/>
19     <n v="1"/>
20     <n v="3578.27"/>
21     <n v="3578.27"/>
22     <n v="0"/>
23     <n v="0"/>
24     <n v="2171.2941999999998"/>
25     <n v="2171.2941999999998"/>
26     <n v="3578.27"/>
27     <n v="286.2615999999999"/>
28     <n v="89.45680000000001"/>
29     </r>
30     <r>
31     <x v="1"/>
32     <s v="Pacific"/>
33     <x v="0"/>
34     <s v="Australia"/>
35     <x v="0"/>
36     <x v="0"/>
37     <s v="3550"/>
38     <x v="0"/>
39     <x v="0"/>
40     <s v="Road-150 Red, 44"/>
41     <s v="This bike is ridden by race winners. Developed with the Adventure
42 Works Cycles professional race team, it has a extremely light heat-treated
43 aluminum frame, and steering that allows precision control."/>

```

```

1      <s v="No Discount"/>
2      <s v="No Discount"/>
3      <s v="No Discount"/>
4      <x v="0"/>
5      <x v="0"/>
6      <x v="1"/>
7      <s v="Australian Dollar"/>
8      <n v="1"/>
9      <n v="3578.27"/>
10     <n v="3578.27"/>
11     <n v="0"/>
12     <n v="0"/>
13     <n v="2171.2941999999998"/>
14     <n v="2171.2941999999998"/>
15     <n v="3578.27"/>
16     <n v="286.2615999999999"/>
17     <n v="89.45680000000001"/>
18     </r>
19   </pivotCacheRecords>

```

20 In the context of pivotCacheRecords:

- 21 • r contains one record.

22 In the context of r:

- 23 • x is an index value referencing an item for this field, as defined in the pivotCacheDefinition part.
- 24 • s indicates that a value is being expressed inline in this record, and it is a string value.
- 25 • n indicates that a value is being expressed inline in this record, and it is a numeric value.

### 26 3.9.3.4 XML - pivotTable part

27 The pivotTable part is organized into 11 sections.

- 28 • Top-level attributes
- 29 • Location information
- 30 • Collection of Fields
- 31 • Fields on the row axis
- 32 • Items on the row axis (specific values)
- 33 • Fields on the column axis
- 34 • Items on the column axis (specific values)
- 35 • Fields in the report filter area
- 36 • Fields in the values area
- 37 • Style information
- 38 • This is what the shell of that structure looks like:

```

1   <pivotTableDefinition>
2     <location/>
3     <pivotFields/>
4     <rowFields/>
5     <rowItems/>
6     <colFields/>
7     <colItems/>
8     <pageFields/>
9     <dataFields/>
10    </dataFields>
11    <conditionalFormats/>
12    <pivotTableStyleInfo/>
13  </pivotTableDefinition>
```

14 Each collection will now be addressed section by section.

#### 15 3.9.3.4.1 Attributes on pivotTableDefinition

```

16  <pivotTableDefinition xmlns:sh="..." name="PivotTable2" cacheId="5"
17    applyNumberFormats="0" applyBorderFormats="0" applyFontFormats="0"
18    applyPatternFormats="0" applyAlignmentFormats="0"
19    applyWidthHeightFormats="1"
20    dataCaption="Values" updatedVersion="3" minRefreshableVersion="3"
21    showCalcMbrs="0" useAutoFormatting="1" colGrandTotals="0"
22    itemPrintTitles="1"
23    createdVersion="3" indent="0" outline="1" outlineData="1"
24    multipleFieldFilters="0">
```

25 In the context of pivotTableDefinition:

- 26 • name indicates the name of the PivotTable.
- 27 • cacheId references by Id a particular pivotCache in the pivotCaches collection listed in workbook.xml.
- 28 • applyNumberFormats value of "1" means to apply legacy autoformat number format properties.
- 29 • applyBorderFormats value of "1" means to apply legacy autoformat border format properties.
- 30 • applyFontFormats value of "1" means to apply legacy autoformat Font format properties.
- 31 • applyPatternFormats value of "1" means to apply legacy autoformat pattern format properties.
- 32 • applyAlignmentFormats value of "1" means to apply legacy autoformat alignment format properties.
- 33 • applyWidthHeightFormats value of "1" means to apply legacy autoformat width and height format properties.
- 34 • dataCaption is the name of the values area header cell which can appear in the PivotTable when two or more fields are in the values area.
- 35 • updatedVersion is the Pivot version that last updated the PivotTable.
- 36 • minRefreshableVersion is the minimum Pivot version required to update this PivotTable's Pivot Cache.

- 1     • showCalcMbrs indicates whether calculated members should be shown in the PivotTable. Only applies  
2        to PivotTables based on OLAP sources.
- 3     • useAutoFormatting indicates whether autoformatting has been applied to the PivotTable.
- 4     • colGrandTotals indicates whether column grand totals are on for this PivotTable.
- 5     • rowGrandTotals defaults to "1" and therefore is not written.
- 6     • itemPrintTitles flag indicating whether PivotItem names should be repeated at the top of each  
7        printed page.
- 8     • createdVersion The Pivot version that created the cache.
- 9     • indent indentation increment for compact row axis, which means the Report Layout is set to Compact  
10      Form.
- 11    • outline flag indicating whether new fields should have their outline form flag set to "1".
- 12    • outlineData flag indicating whether the values field in the PivotTable should be displayed in outline  
13      form.
- 14    • multipleFieldFilters flag indicating whether each field of a pivot table can have multiple filters set on  
15      it.

#### 16   3.9.3.4.2                  Location Information

17   Location provides details on where the PivotTable is located in the sheet.

```
18   <location ref="B6:G13" firstHeaderRow="1" firstDataRow="4"  
19      firstDataCol="1" rowPageCount="3" colPageCount="1"/>
```

20   In the context of location:

- 21    • ref the location of the PivotTable area, not including the report filter area.
- 22    • firstHeaderRow the first row of the PivotTable header, relative to the top left cell in ref value.
- 23    • firstDataRow the first row of the PivotTable values area, relative to the top left cell in ref value.
- 24    • firstDataCol the first column of the PivotTable values area, relative to the top left cell in ref value.
- 25    • rowPageCount indicates how many rows the report filter area will occupy, as fields are added to it,  
26        before taking up another column (there can be multiple rows and columns of fields in the report filter  
27        area). By default there is a single column of report filter fields and the fields occupy as many rows as  
28        there are fields..
- 29    • colPageCount indicates how many columns the report filter region will occupy, as fields are added to  
30        it, before taking up another row (there can be multiple rows and columns of fields in the report filter  
31        region). By default, there is a single column of report filter fields and the fields occupy as many rows as  
32        there are fields.

#### 33   3.9.3.4.3                  PivotTable Fields

34   This collection expresses item order and field information for each field associated with the PivotTable,  
35   whether shown in the PivotTable report or not. (Note that items have been removed from the "Customer  
36   Name" and "City" fields (1st and 6th) to shorten the example.)

```
1 <pivotFields count="28">
2   <pivotField showAll="0" includeNewItemInFilter="1">
3     <items count="8">
4       <item x="66"/>
5       <item x="133"/>
6       <item x="74"/>
7       <item x="27"/>
8       <item x="118"/>
9       <item x="63"/>
10      <item x="141"/>
11      <item t="default"/>
12    </items>
13  </pivotField>
14  <pivotField showAll="0" includeNewItemInFilter="1"/>
15  <pivotField axis="axisPage" showAll="0" includeNewItemInFilter="1">
16    <items count="2">
17      <item x="0"/>
18      <item t="default"/>
19    </items>
20  </pivotField>
21  <pivotField showAll="0" includeNewItemInFilter="1"/>
22  <pivotField axis="axisPage" showAll="0" includeNewItemInFilter="1">
23    <items count="6">
24      <item x="3"/>
25      <item x="1"/>
26      <item x="2"/>
27      <item x="4"/>
28      <item x="0"/>
29      <item t="default"/>
30    </items>
31  </pivotField>
32  <pivotField axis="axisPage" showAll="0" includeNewItemInFilter="1">
33    <items count="8">
34      <item x="0"/>
35      <item x="1"/>
36      <item x="2"/>
37      <item x="3"/>
38      <item x="4"/>
39      <item x="5"/>
40      <item x="6"/>
41      <item t="default"/>
42    </items>
43  </pivotField>
```

```
1 <pivotField showAll="0" includeNewItemsInFilter="1"/>
2 <pivotField axis="axisRow" showAll="0" includeNewItemsInFilter="1">
3   <items count="2">
4     <item x="0"/>
5     <item t="default"/>
6   </items>
7 </pivotField>
8 <pivotField axis="axisRow" showAll="0" includeNewItemsInFilter="1">
9   <items count="3">
10    <item x="1"/>
11    <item x="0"/>
12    <item t="default"/>
13   </items>
14 </pivotField>
15 <pivotField showAll="0" includeNewItemsInFilter="1"/>
16 <pivotField showAll="0" includeNewItemsInFilter="1"/>
17 <pivotField showAll="0" includeNewItemsInFilter="1"/>
18 <pivotField showAll="0" includeNewItemsInFilter="1"/>
19 <pivotField showAll="0" includeNewItemsInFilter="1"/>
20 <pivotField axis="axisCol" showAll="0" includeNewItemsInFilter="1">
21   <items count="2">
22     <item x="0"/>
23     <item t="default"/>
24   </items>
25 </pivotField>
26 <pivotField axis="axisCol" showAll="0" includeNewItemsInFilter="1">
27   <items count="2">
28     <item x="0"/>
29     <item t="default"/>
30   </items>
31 </pivotField>
32 <pivotField axis="axisCol" showAll="0" includeNewItemsInFilter="1">
33   <items count="4">
34     <item x="1"/>
35     <item x="2"/>
36     <item x="0"/>
37     <item t="default"/>
38   </items>
39 </pivotField>
40 <pivotField showAll="0" includeNewItemsInFilter="1"/>
41 <pivotField showAll="0" includeNewItemsInFilter="1"/>
42 <pivotField showAll="0" includeNewItemsInFilter="1"/>
43 <pivotField showAll="0" includeNewItemsInFilter="1"/>
```

```

1   <pivotField showAll="0" includeNewItemInFilter="1"/>
2   <pivotField showAll="0" includeNewItemInFilter="1"/>
3   <pivotField showAll="0" includeNewItemInFilter="1"/>
4   <pivotField showAll="0" includeNewItemInFilter="1"/>
5   <pivotField dataField="1" showAll="0" includeNewItemInFilter="1"/>
6   <pivotField showAll="0" includeNewItemInFilter="1"/>
7   <pivotField showAll="0" includeNewItemInFilter="1"/>
8   </pivotFields>
```

9 In the context of pivotField:

- 10 • showAll flag indicating whether to show all items for this field.
- 11 • includeNewItemInFilter Flag indicating if new items in the data source are included in the filter automatically after refresh when there was at least one hidden item for the field.
- 12 • axis indicates on which axis this field is shown on the PivotTable.
- 13 • dataField indicates that this field is in the values area of the PivotTable.

15 In the context of items, which is a listing of items (by index) in this field. The order in which the items are listed  
16 is the order they would appear on a particular axis (row or column, for example). In this example, the first field  
17 is "Customer Name" and the first item referenced here is <item x="66"/>, which references the value "Adam L  
18 Flores" in the pivotCacheDefinition. Therefore if one added "Customer Name" to the row axis, "Adam L Flores"  
19 would be the first row item listed.

20 In the context of item:

- 21 • t value of 'default' indicates the subtotal or total item.

#### 22 3.9.3.4.4 Row Axis Fields

23 This collection indicates which fields are on the row axis of the PivotTable.

```

24 <rowFields count="2">
25   <field x="7"/>
26   <field x="8"/>
27 </rowFields>
```

28 In the context of field within rowFields:

- 29 • x is a zero based index into the pivotFields collection.

30 For this example, this collection indicates that "Product Category" and "Product Subcategory" are on the row  
31 axis of the PivotTable, in that order.

1    3.9.3.4.5              Row Items

2    This collection is a listing of all the values on the row axis of the PivotTable. In the spreadsheet example, the  
 3    item values are found in cells B10:B13. For example, "Bikes" is in B10, and corresponds to the first I element  
 4    below.

```

5      <rowItems count="4">
6          <i>
7              <x/>
8          </i>
9          <i r="1">
10         <x/>
11         </i>
12         <i r="1">
13             <x v="1"/>
14         </i>
15         <i t="grand">
16             <x/>
17         </i>
18     </rowItems>
```

19    In the context of rowItems:

- 20       • i expresses all the values (for all fields) in one row of the row axis. There will be an I element for every  
 21       row in the PivotTable.

22    In the context of i:

- 23       • r indicates how many fields/item values to "fill down" from the previous row item.

24    Note that the first item has no r explicitly written. Since a default of "0" is specified in the schema, for any  
 25    item whose r is missing, a default value of "0" is implied.

26    In the context of x:

- 27       • v is a zero-based index referencing a pivotField item value. There will be as many x elements as  
 28       there are item values in any particular row. Note that these x elements may not be explicitly written,  
 29       but instead "inherited" from the previous row/i element, via the value of r. Note also that the  
 30       pivotField items don't list values explicitly, but instead reference a shared item value in the  
 31       pivotCacheDefinition part.

32    Note that the first instance of x has no attribute value v associated with it, so v's default value of "0" is implied.

33    Looking at the layout of the PivotTable in the spreadsheet for this example, "Bikes" is the first (and only) item  
 34    value in the first row, in cell B10. In the XML defining the PivotTable row item values, the first I element  
 35    corresponds to the first row. There is a single index element x. The first (and only) x element corresponds to

1 the first field on the row axis, namely "Product Category", and an index value of "0" indicates that the 0th item  
 2 in the items collection for that pivotField definition is how to obtain the item value. Note that "Bikes" isn't  
 3 explicitly listed as a value here, but instead the 0th item is an index to this field's shared items collection in the  
 4 pivotCacheDefinition part.

5 For the second row, there are two item values, one item value (Bikes) from the first field in that row (Product  
 6 Category) and one item value (Mountain Bikes) from the second field in that row (Product Subcategory). In the  
 7 PivotTable, the first item value "Bikes" is hidden from view. In the XML for this example, the second I element  
 8 expresses both item values for this row. The first item value "Bikes" is expressed implicitly, because the value  
 9 of r on the second i element is '1', indicating that the first item value from the previous row will be reused  
 10 again as the first item value for the current row. The second item value is expressed explicitly via the x element  
 11 under the second i element. The index of '0' indicates that the 0th item in the pivotField element for that field  
 12 is how to obtain the item value. Note again that the 0th item is itself an index into this field's shared items  
 13 collection in the pivotCacheDefinition part.

14 The item values for the third row can be discovered in a similar way, so will not be discussed in detail here.

15 In the context of item:

- 16 • t value of 'default' indicates a grand total as the last row item value.

#### 17 3.9.3.4.6 Column Axis Fields

18 This collection indicates which fields are on the column axis of the PivotTable.

```
19 <colFields count="3">
20   <field x="14"/>
21   <field x="15"/>
22   <field x="16"/>
23 </colFields>
```

24 In the context of field:

- 25 • x is a zero based index into the pivotFields collection defined in this part.

26 For this example, the collection indicates that "Year", "Quarter" and "Month" are on the column axis of the  
 27 PivotTable, in that order.

#### 28 3.9.3.4.7 Column Items

29 This collection is a listing of all the values on the column axis of the PivotTable. In this example, the item values  
 30 are found in cells C6:H8. For example, "2001" / "3" / "July" values are in C7:C9. Those are the first column  
 31 item values and are referenced by the first *<i>* element below.

```

1   <colItems count="5">
2     <i>
3       <x/>
4       <x/>
5       <x/>
6     </i>
7     <i r="2">
8       <x v="1"/>
9     </i>
10    <i r="2">
11      <x v="2"/>
12    </i>
13    <i t="default" r="1">
14      <x/>
15    </i>
16    <i t="default">
17      <x/>
18    </i>
19  </colItems>
```

20 In the context of colItems:

- 21 • i expresses all the values (for all fields) in one column of the column axis. There will be an i element for  
22 every column in the PivotTable column area.

23 In the context of i:

- 24 • r indicates how many fields/item values to "fill right" from the previous column.

25 Note that the first item has no r explicitly written so the default value of "0" is implied.

26 In the context of x:

- 27 • v is a zero-based index referencing a pivotField item value. There will be as many x elements as  
28 there are item values in any particular column. Note that these x elements sometimes are not  
29 explicitly written, but instead "inherited" from the previous column/i element, via the value of r. Note  
30 also that the pivotField items don't list values explicitly, but instead reference a shared item value in  
31 the pivotCacheDefinition part.

32 Note that the first instance of x has no attribute value v associated with it, so v's default value of "0" is  
33 implied.

34 The first i collection represents all item values for the first column in the column axis area of the PivotTable.  
35 The first x in the first i corresponds to the first field in the columns area of the PivotTable, namely "Year". The  
36 implied index value of '0' on this x indicates that the item value for this first item in the column is the 0th item

1 for this pivotField. The 0th item for this pivotField is itself an index to an item value into this field's shared  
 2 items collection in the pivotCacheDefinition part, namely "2001".  
 3 The item values corresponding to the second and third x elements can be found in the same way, arriving at  
 4 "3" for the second item value, and arriving at "July" for the third item value for this first column.  
 5 The second i collection expresses all three item values for the second column in the column axis area. The  
 6 r value of '2' indicates that the first two item values from the previous column will be repeated here, which  
 7 means that the first item value for this second column will be "2001" again and the second item value for this  
 8 second column will be "3". The third item value is expressed by the only x element under this second  
 9 i element, and without further explanation is understood to reference the item value "August".

#### 10 3.9.3.4.8 Report Filter Area Fields

11 This collection describes which fields are found in the report filter area of the PivotTable.

```
12 <pageFields count="3">
13   <pageField fld="2" hier="0"/>
14   <pageField fld="4" hier="0"/>
15   <pageField fld="5" hier="0"/>
16 </pageFields>
```

17 In the context of pageField:

- 18 • fld is a zero-based index indicating the field to be on the report filter area.
- 19 • hier is an index of the OLAP hierarchy to which this belongs.

#### 20 3.9.3.4.9 Values Area Fields

21 This collection describes which fields are found in the values area of the PivotTable.

```
22 <dataFields count="1">
23   <dataField name="Sum of Sales Amount" fld="25" baseField="0" baseItem="0"/>
24 </dataFields>
```

25 In the context of dataField:

- 26 • name is the name of the values field.
- 27 • fld is the index of the field being summarized.
- 28 • baseField is the index of the base field when showDataAs calculation is in use.
- 29 • baseItem is the index of the base item when showDataAs calculation is in use.

#### 30 3.9.3.4.10 PivotTable Style Information

31 Styles information is discussed in the informative section on spreadsheetML styles. Therefore the XML is  
 32 provided for completeness, but will not be discussed here.

```

1 <pivotTableStyleInfo name="PivotStyleDark8" showRowHeaders="1"
2   showColHeaders="1" showRowStripes="0" showColStripes="0"
3   showLastColumn="1"/>
4 </pivotTableDefinition>
```

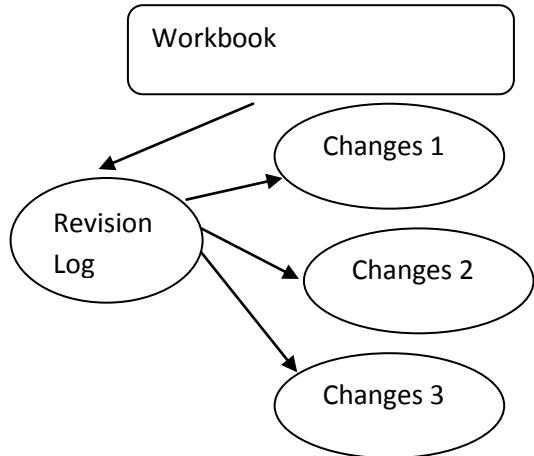
## 5 **3.10 Shared Workbook Revisions**

### 6 **3.10.1 Overview**

7 The Shared Workbooks architecture enables a spreadsheet application to record revisions made to a workbook  
 8 (e.g., track changes), and is designed to enable multiple users to edit the same workbook at the same time.  
 9 Therefore, the application needs to support the ability to read changes made by another user, and update its  
 10 own state of the same workbook with those changes, even when those changes are made concurrently with  
 11 other changes made by other users. Inevitably, there will be conflicts, and therefore merge conflict resolution  
 12 should be supported by the runtime application.

13 This architecture supports the ability to track changes made by a single user as well.

### 14 **3.10.2 How It Works**



22 Relationship diagram

23 A Shared Workbook must have shared mode turned on. For unsaved workbooks, this will require a save,  
 24 because revisions will be stored in the file.

25 Changes to the workbook are saved as Shared Workbook Revision Header parts within the document at each  
 26 save or time interval specified.

27 A table summarizing the revision logs (revisionHeaders.xml) tracks when changes are made, who made them,  
 28 and lists the relationship id to the specific Shared Workbook Revision Log part.

29 The application scans the summary table for new change logs and merges them into the workbook.

### 3.10.3 Example

Consider a series of edits made by different users.

#### 3.10.3.1 First Edit

Starting with a blank workbook, the first user types "A, B, C" into A1:C1, and "1, 2, 3; 4, 5, 6" into A2:C3, like this:

	A	B	C
1	A	B	C
2	1	2	3
3	4	5	6
4			

Once the file is saved to disk after these edits, the summary table is updated and the revision log for this change is written.

#### 3.10.3.1.1 Summary Revision Table

Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

Inside the summary table there is a revision header definition corresponding to the time of the edit:

```

12 <header guid="{902054C2-C7B5-48BA-BFB2-4D439D9758D6}"
13   dateTime="2006-04-14T10:33:16" maxSheetId="4" userName="User 1"
14   r:id="rId2" minRId="1" maxRId="11">
15   <sheetIdMap count="3">
16     <sheetId val="1"/>
17     <sheetId val="2"/>
18     <sheetId val="3"/>
19   </sheetIdMap>
20 </header>
```

Notice that the user name, userName, and date and time stamp, dateTime, for the edit is stored along with an outline of the sheet structure. Use the r:id value of rId2 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way, the corresponding Shared Workbook Revision Log can be located.

#### 3.10.3.1.2 First Edit Revision Log

Inside the corresponding Shared Workbook Revision Log part is the following content:

```
1   <revisions xmlns="..." xmlns:r="...">
2     <rcc rId="1" sId="1">
3       <nc r="A1" t="inlineStr">
4         <is>
5           <t>A</t>
6         </is>
7       </nc>
8     </rcc>
9     <rcc rId="2" sId="1">
10      <nc r="B1" t="inlineStr">
11        <is>
12          <t>B</t>
13        </is>
14      </nc>
15    </rcc>
16    <rcc rId="3" sId="1">
17      <nc r="C1" t="inlineStr">
18        <is>
19          <t>C</t>
20        </is>
21      </nc>
22    </rcc>
23    <rrc rId="4" sId="1" eol="1" ref="A2:XFD2" action="insertRow"/>
24    <rcc rId="5" sId="1">
25      <nc r="A2">
26        <v>1</v>
27      </nc>
28    </rcc>
29    <rcc rId="6" sId="1">
30      <nc r="B2">
31        <v>2</v>
32      </nc>
33    </rcc>
34    <rcc rId="7" sId="1">
35      <nc r="C2">
36        <v>3</v>
37      </nc>
38    </rcc>
39    <rrc rId="8" sId="1" eol="1" ref="A3:XFD3" action="insertRow"/>
```

```

1      <rcc rId="9" sId="1">
2          <nc r="A3">
3              <v>4</v>
4          </nc>
5      </rcc>
6      <rcc rId="10" sId="1">
7          <nc r="B3">
8              <v>5</v>
9          </nc>
10     </rcc>
11     <rcc rId="11" sId="1">
12         <nc r="C3">
13             <v>6</v>
14         </nc>
15     </rcc>
16 </revisions>
```

17 rId is the revision Id, and indicates the order in which the particular revision should be applied.

18 sId indicates the sheet to which this revision applies.

19 rcc means "revision cell change"

20 nc means new cell, and is of type CT\_Cell (see §3.2 for more information on the cell definition). Note that  
21 instead of using a shared string table, strings are expressed inline for these cells.

22 rrc means "revision row/column". Note that rrc can have an associated action, like insertRow (or deleteRow),  
23 which would cause a row to be inserted (or deleted) at that step in the series of revisions.

### 24 3.10.3.2 Second Edit

25 During the second edit, bold facing has been applied to A1:C1, and a formula has been applied to A4:C4 to  
26 sum the data in the table. For example, A4 contains =SUM(A2:A3).

	A	B	C
1	A	B	C
2	1	2	3
3	4	5	6
4	5	7	9

27  
28 Once the file is saved to disk after these edits, the summary table is updated and the revision log for this  
29 change is written.

### 30 3.10.3.2.1 Summary Revision Table

31 Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

1 Inside the summary table there is a revision header definition corresponding to the time of the edit:

```

2   <header guid="{A3A5EE09-2092-433C-895D-77D5A15DC847}"
3     dateTime="2006-04-14T10:34:10" maxSheetId="4" userName="User 2"
4     r:id="rId3" minRId="12" maxRId="15">
5     <sheetIdMap count="3">
6       <sheetId val="1"/>
7       <sheetId val="2"/>
8       <sheetId val="3"/>
9     </sheetIdMap>
10    </header>
```

11 This time the user name has been updated. Use the r:id value of rId3 and then follow the relationship  
 12 expressed in revisionHeaders.xml.rels. In this way, the corresponding Shared Workbook Revision Log can be  
 13 located.

#### 14 3.10.3.2.2 Second Edit Revision Log

15 Inside the corresponding Shared Workbook Revision Log part is the following content:

```

16   <revisions xmlns="..." xmlns:r="...">
17     <rfmt sheetId="1" sqref="A1:C1" start="0" length="2147483647">
18       <dxf>
19         <font>
20           <b/>
21           </font>
22       </dxf>
23     </rfmt>
24     <rrc rId="12" sId="1" eol="1" ref="A4:XFD4" action="insertRow"/>
25     <rcc rId="13" sId="1">
26       <nc r="A4">
27         <f>SUM(A2:A3)</f>
28       </nc>
29     </rcc>
30     <rcc rId="14" sId="1">
31       <nc r="B4">
32         <f>SUM(B2:B3)</f>
33       </nc>
34     </rcc>
```

```

1   <rcc rId="15" sId="1">
2     <nc r="C4">
3       <f>SUM(C2:C3)</f>
4     </nc>
5   </rcc>
6   <rcv guid="{34804977-BBD3-40C9-87A7-1779BEE2183C}" action="add"/>
7 </revisions>

```

8 rfmt indicates a formatting revision

9 start and length indicate where to apply the formatting on the string

10 eol indicates that an insert is happening at the end of a list of data (end row)

11 rcv means "revision custom view", and indicates that a custom view is to be added.

### 12 3.10.3.3 Third Edit

13 During this editing session, column A has been inserted, and columns have been inserted between the data.

14 Additionally, a row has been inserted between the data and the summary formula row, and at the top of the  
15 worksheet.

	A	B	C	D	E	F
1						
2	A		B		C	
3		1		2		3
4		4		5		6
5						
6		5		7		9
7						

16 Once the file is saved to disk after these edits, the summary table is updated and the revision log for this  
change is written.

#### 19 3.10.3.3.1 Summary Revision Table

20 Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

```

21 <header guid="{894981D2-DACF-4C1B-951C-EB199EA01DBF}"
22   dateTime="2006-04-14T10:36:10" maxSheetId="4" userName="User 2"
23   r:id="rId4" minRId="16" maxRId="20">
24   <sheetIdMap count="3">
25     <sheetId val="1"/>
26     <sheetId val="2"/>
27     <sheetId val="3"/>
28   </sheetIdMap>
29 </header>

```

1 Use the r:id value of rId4 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way,  
 2 the corresponding Shared Workbook Revision Log part can be located.

3 **3.10.3.3.2 Third Edit Revision Log**

4 Inside the corresponding Shared Workbook Revision Log part is the following content:

```
5   <revisions xmlns="..." xmlns:r="...">
6     <rrc rId="16" sId="1" ref="A1:XFD1" action="insertRow"/>
7     <rrc rId="17" sId="1" ref="A1:A1048576" action="insertCol"/>
8     <rrc rId="18" sId="1" ref="C1:C1048576" action="insertCol"/>
9     <rrc rId="19" sId="1" ref="E1:E1048576" action="insertCol"/>
10    <rrc rId="20" sId="1" ref="A5:XFD5" action="insertRow"/>
11    <rcv guid="{34804977-BBD3-40C9-87A7-1779BEE2183C}" action="delete"/>
12    <rcv guid="{34804977-BBD3-40C9-87A7-1779BEE2183C}" action="add"/>
13  </revisions>
```

14 rrc indicates a "revision to row/column". There are several row inserts and column inserts expressed here.

15 **3.10.3.4 Fourth Edit**

16 During this edit, a double-underscore cell border was applied to B4 and D4. Also, column F (the data titled "C")  
 17 was deleted.

18

20 Once the file is saved to disk after these edits, the summary table is updated and the revision log for this  
 21 change is written.

22 **3.10.3.4.1 Summary Revision Table**

23 Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

```
24  <header guid="{A478A962-DEB9-43AA-BB25-2C54AFA155F1}">
25    <dateTime>2006-04-14T10:37:14</dateTime>
26    <maxSheetId>4</maxSheetId>
      <userName>User 2</userName>
      <r:id>rId5</r:id>
      <minRId>21</minRId>
```

```

1   <sheetIdMap count="3">
2     <sheetId val="1"/>
3     <sheetId val="2"/>
4     <sheetId val="3"/>
5   </sheetIdMap>
6 </header>
```

7 Use the r:id value of rId5 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way,  
8 the corresponding Shared Workbook Revision Log part can be located.

### 9 3.10.3.4.2 Fourth Edit Revision Log

10 Inside the corresponding Shared Workbook Revision Log part is the following content:

```

11   <revisions xmlns="..." xmlns:r="...">
12     <rrc rId="21" sId="1" ref="F1:F1048576" action="deleteCol">
13       <rfmt sheetId="1" xfDxf="1" sqref="F1:F1048576" start="0" length="0"/>
14       <rcc rId="0" sId="1" dxf="1">
15         <nc r="F2" t="inlineStr">
16           <is>
17             <t>C</t>
18           </is>
19         </nc>
20         <ndxf>
21           <font>
22             <b/>
23             <sz val="11"/>
24             <color theme="1"/>
25             <name val="Calibri"/>
26             <scheme val="minor"/>
27           </font>
28         </ndxf>
29       </rcc>
30       <rcc rId="0" sId="1">
31         <nc r="F3">
32           <v>3</v>
33         </nc>
34       </rcc>
35       <rcc rId="0" sId="1">
36         <nc r="F4">
37           <v>6</v>
38         </nc>
39       </rcc>
40       <rcc rId="0" sId="1">
41         <nc r="F6">
```

```

1          <f>SUM(F3:F4)</f>
2      </nc>
3      </rcc>
4  </rrc>
5  <rfmt sheetId="1" sqref="B4" start="0" length="0">
6      <dxf>
7          <border>
8              <left/>
9              <right/>
10             <top/>
11             <bottom style="double">
12                 <color auto="1"/>
13             </bottom>
14         </border>
15     </dxf>
16 </rfmt>
17 <rfmt sheetId="1" sqref="D4" start="0" length="0">
18     <dxf>
19         <border>
20             <left/>
21             <right/>
22             <top/>
23             <bottom style="double">
24                 <color auto="1"/>
25             </bottom>
26         </border>
27     </dxf>
28 </rfmt>
29 </revisions>

```

30 The first rrc element, with action="deleteCol" expresses that column F was deleted. Additionally, child  
 31 collections of rrc contain all the column, formatting, and cell information (values and formulas) that was  
 32 deleted as part of deleting column F.

33 xfDxf true means a whole row/column of formatting was affected.

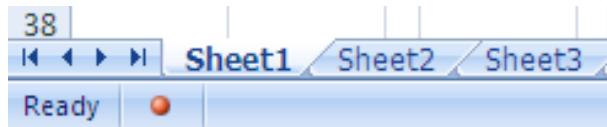
34 dxf true means cell change includes format change

35 The rfmt collections at the bottom of this XML indicate that borders were applied to B4 and D4.

### 36 3.10.3.5 Fifth Edit

37 Rename "Sheet1" to "Published Numbers".

38 Before:



2 After:



4 Once the file is saved to disk after these edits, the summary table is updated and the revision log for this  
5 change is written.

### 6 3.10.3.5.1 Summary Revision Table

7 Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

```
8 <header guid="{B0CB8BC9-63A4-4830-8821-E03C053BD326}"
9   dateTime="2006-04-14T10:40:24" maxSheetId="4" userName="User 2"
10  r:id="rId6" minRId="22">
11  <sheetIdMap count="3">
12    <sheetId val="1"/>
13    <sheetId val="2"/>
14    <sheetId val="3"/>
15  </sheetIdMap>
16 </header>
```

17 Use the r:id value of rId6 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way  
18 the corresponding Shared Workbook Revision Log part can be located.

### 19 3.10.3.5.2 Fifth Edit Revision Log

20 Inside the corresponding Shared Workbook Revision Log part is the following content:

```
21 <revisions xmlns="..." xmlns:r="...">
22   <rsnm rId="22" sheetId="1" oldName="[SharedWorkbook.xlsx]Sheet1"
23     newName="[SharedWorkbook.xlsx]Published Numbers"/>
24 </revisions>
```

25 rsnm means "revision: sheet name".

26 oldName indicates the name of the sheet before renaming it.

27 newName indicates the name of the sheet after renaming it.

## 1    3.11    Query Tables

### 2    3.11.1    Overview

3    A *QueryTable object* is a range that is bound to an external data source. It is a cohesive range of cells in a sheet  
 4    that share a common collection of properties and behaviors, separate from the connection itself. A QueryTable  
 5    object can be associated with a cell range or a table definition.

### 6    3.11.2    Web Query Example

7    This example illustrates a range, B2:D6, QueryTable rendering, which is data-bound to a table found on  
 8    <http://www.msn.com>, specifically the financial information table usually found on that page.

	A	B	C	D	E	F
1						
2	This table charts the key U.S. financial indices by their last					
3	Index	Last	Change			
4	Dow	11,642.65	2.88			
5	NASDAQ	2,320.74	-17.51			
6	S&P	1,322.85	-2.29			

#### 10    3.11.2.1    QueryTable XML

```
11 <queryTable xmlns="..." name="www.msn" preserveFormatting="0"
12   connectionId="1"
13   autoFormatId="16" applyNumberFormats="0" applyBorderFormats="0"
14   applyFontFormats="1" applyPatternFormats="1" applyAlignmentFormats="0"
15   applyWidthHeightFormats="0"/>
```

16    In the context of queryTable:

- 17    • name is the name of the QueryTable.
- 18    • preserveFormatting indicates whether to retain user-applied formatting after refresh or re-apply
- 19    source data formatting.
- 20    • connectionId is the workbook connection's id.
- 21    • autoFormatId identifies (by implied index) the auto-format applied to the QueryTable.

22    All remaining attributes beginning with apply... indicate whether to apply this particular aspect of the auto-  
 23    format definition.

### 24    3.11.3    Text Import Example

25    This example illustrates a range (B2:D3) QueryTable rendering which is data-bound to a text file. Notice that  
 26    formulas are entered in E2:E3, the column directly to the right of the QueryTable range.

	A	B	C	D	E	F	G
1							
2		1	2	3	6		
3		4	5	6	15		
4							

### 3.11.3.1 QueryTable XML

```

3   <queryTable xmlns="..." name="Text" refreshOnLoad="1" fillFormulas="1"
4     removeDataOnSave="1" connectionId="3" autoFormatId="16"
5     applyNumberFormats="0" applyBorderFormats="0" applyFontFormats="1"
6     applyPatternFormats="1" applyAlignmentFormats="0"
7     applyWidthHeightFormats="0"/>

```

8 In the context of queryTable:

- 9 • refreshOnLoad value of 1 indicates that this QueryTable should be refreshed when the workbook is  
10 opened.
- 11 • fillFormulas indicates that this QueryTable has immediately adjacent columns (which are not part of  
12 the QueryTable range) containing formulas that need to be filled down as the QueryTable grows and  
13 shrinks in size after refresh.
- 14 • removeDataOnSave indicates that the data in the worksheet resulting from the QueryTable refresh  
15 should be removed from the worksheet when saved and closed.

### 3.11.4 Access Table Example

17 This example demonstrates a QueryTable that is applied to a Table object. This data came from connecting to  
18 an Access database table with four fields: "ID", "Field1", "Field2", and "Field3". "Field3" has been deleted from  
19 the QueryTable in the worksheet below. Notice that a calculated column has been added to the Table, in the  
20 column titled "CustomClientColumn", which concatenates the values from "Field1" and "Field2".

	A	B	C	D	E
1					
2	ID	Field1	Field2	CustomClientColumn	
3	4	A	B	AB	
4	5	D	E	DE	
5	6	G	H	GH	
6					

### 3.11.4.1 QueryTable XML

```

23   <queryTable xmlns="..." name="Database1.accdb" connectionId="2"
24     autoFormatId="16" applyNumberFormats="0" applyBorderFormats="0"
25     applyFontFormats="0" applyPatternFormats="0" applyAlignmentFormats="0"
26     applyWidthHeightFormats="0">

```

```

1   <queryTableRefresh nextId="6" unboundColumnsRight="1">
2     <queryTableFields count="4">
3       <queryTableField id="1" name="ID" tableColumnId="1"/>
4       <queryTableField id="2" name="Field1" tableColumnId="2"/>
5       <queryTableField id="3" name="Field2" tableColumnId="3"/>
6       <queryTableField id="5" dataBound="0" tableColumnId="4"/>
7     </queryTableFields>
8     <queryTableDeletedFields count="1">
9       <deletedField name="Field3"/>
10    </queryTableDeletedFields>
11  </queryTableRefresh>
12 </queryTable>

```

13 In the context of queryTableRefresh:

- 14 • nextId is the next available Id that can be assigned to a field. This is an optimization done for  
load/save, to avoid recalculating the value.
- 15 • unboundColumnsRight are the number of columns on the right side of the QueryTable that aren't  
data bound (don't come from the external data)

18 In the context of queryTableFields:

- 19 • Each of the queryTableField elements expresses information about one of the columns in the Table  
20 that is part of the QueryTable. For example, the right-most column's dataBound is set to 0, indicating  
21 that this column is not bound to external data.

22 In the context of queryTable:

- 23 • queryTableDeletedFields collection expresses which fields returned by the connection have been  
24 deleted from the QueryTable. This is tracked so that the connection information does not have to be  
25 updated with which columns are no longer required.

## 26 3.12 External Connection

### 27 3.12.1 Overview

28 Many spreadsheet users want to be able to access data from external sources: databases, text files, web  
29 pages, XML web services, OLAP cubes. Typically, a spreadsheet application will provide abilities for the user to  
30 locate, browse, connect to, and query external data sources. Once the data source has been located,  
31 connected to, and queried, the resulting data must be rendered in the spreadsheet application, and made  
32 available for further analysis.

33 Data sources such as databases are made available for browsing and consumption via data-provider  
34 technologies. Typically, the data provider provides a standard interface for accessing the data, and removes  
35 the complexity introduced due to each database application's providing non-standard data access APIs. In this

1 way, OLEDB providers, for example, can be written for myriad database implementations, and a consumer can  
2 always use a single interface (defined by OLEDB) to access these disparate data sources.

3 A live connection to a data source is established by the application at runtime, and can only exist as a live  
4 connection while the application is running. There are two types of information about a particular connection:

- 5     • The information used to establish the connection.  
6     • The information and properties about how the connection should be used and how the connection  
7         should behave in conjunction with the application.

8 Information about a connection can be supplied by the user as the connection is being established—for  
9 example, providing a password, picking a table, applying a filter, or setting behavioral properties such as  
10 whether to refresh the data when the workbook is opened and whether to store refreshed data in the  
11 worksheet when the workbook is saved.

12 Information about a connection can also be persisted in a connection file separately from the workbook file. In  
13 this way a directory or file share containing a variety of these connection files can be considered a library of  
14 data connections, for example.

15 Any time a connection is established, whether by using information from a connection file or by gathering the  
16 connection information directly from the user, a copy of the connection information is stored in the workbook.

17 Data providers and connection types discussed below are:

- 18     • ODBC  
19     • OLEDB  
20     • ADO  
21     • DAO  
22     • Text Import  
23     • Web

24 The corresponding features in SpreadsheetML that render and analyze the data are:

- 25     • Query Table  
26     • Table  
27     • XML Map  
28     • Pivot Table  
29     • The CUBE\* Functions

### 30 **3.12.2 OLAP Connection**

31 Below is a PivotTable that is rendering data from an OLAP source:

A	B	C
1		
2	Row Labels	Amount
3	Current Year's Actuals	398755.69
4	Adjustment for Budget input	565238.13
5	Current Year's Budget	565238.13
6	Forecast	565238.13
7	Grand Total	398755.69
8		

### 3.12.3 Pivot XML fragment

In this example, the PivotTable data cache records /cacheSource@connectionId="2", which associates this PivotTable to the connection whose id="2" in the workbook connections part.

```

5   <pivotCacheDefinition ... saveData="0" refreshedBy="Chad Rothschilder"
6     refreshedDate="2006-04-13T16:02:14" backgroundQuery="1"
7     createdVersion="3"
8     refreshedVersion="3" minRefreshableVersion="3" recordCount="0">
9     <cacheSource type="external" connectionId="2"/>
10    <cacheFields count="2">
11      <cacheField name="[Category].[Category Description]"
12        caption="Category
13        Description" numFmtId="0" hierarchy="1" level="1">
14          <sharedItems count="4">
15            <s v="[Category].[All Category].[Current Year's Actuals]"
16              c="Current Year's Actuals"/>

```

### 3.12.4 Connection XML

Looking at the XML in the workbook connections part that describes the connection whose id="2":

```

19   <connection id="2" odcFile="
20     c:\...\externalData.odc" keepAlive="1" name="externalData"
21     description="FoodMart 2000 - Budget Planing" type="5"
22     refreshedVersion="3" background="1">
23     <dbPr connection="Provider=MSOLAP.2;Integrated Security=SSPI;Persist
24       Security Info=True;Data Source=xlextdat8;Initial
25         Catalog=AdamTest;Client
26         Cache Size=25;Auto Synch Period=10000;MDX Compatibility=1"
27         command="Budget" commandType="1"/>
28     <olapPr sendLocale="1" rowDrillCount="1000"/>
29   </connection>

```

Attributes on the connection element express properties about the connection.

- odcFile indicates the location of the data connection file which was used to create this connection.  
Data connection files can be created on the local machine, on any network share, or any web location

1 whenever the connection is created so that the connection information can be reused if desired. The  
 2 connection information is copied from the connection file into the spreadsheet. When a connection  
 3 cannot be established, the spreadsheet application can check the external connection file to see if a  
 4 newer definition of the connection is available.

- 5 • name indicates the friendly name of the connection. This name must be unique within a workbook.
- 6 • keepAlive indicates that the application should hold the connection open once established, instead of  
 closing the connection after retrieving the data.
- 7 • type value of 5 indicates that this connection type is OLEDB.
- 8 • refreshedVersion indicates the version of the application which last refreshed this connection.
- 9 • background value of 1 indicates that background refresh (asynchronous refresh) is enabled. Note that  
 10 this is not a guarantee that the connection will be refreshed asynchronously. Certain objects may  
 11 require a connection to be refreshed either synchronously or asynchronously regardless of this setting.

13 Attributes on the dbPr element express additional properties on the connection.

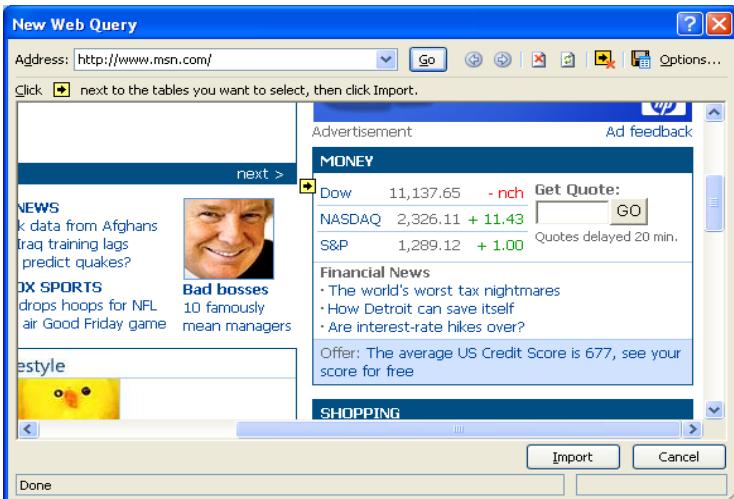
- 14 • connection expresses the connection string that is needed to establish a connection to the external  
 15 data source.
- 16 • command can indicate a table name, a cube name, or an SQL expression requesting data.
- 17 • commandType indicates what kind of information is found in command: 1 means the value of  
 18 command is the name of an OLAP cube.

19 Attributes on the olapPr element express properties that apply to connections to OLAP data sources.

- 20 • sendLocale value of 1 indicates that the client application should send its user interface language  
 21 locale to the OLAP data provider in order to receive back from the server localized OLAP cube member  
 22 string values.
- 23 • rowDrillCount is number of rows to return on a drill through request.

### 24 3.12.5 Web Query

25 A possible user interface for picking a web source is:



26

1 A possible rendering in the spreadsheet grid might be:

B	C	D
<b>Web Query to MSN Money</b>		
Dow	11,137.65	7.68
NASDAQ	2,326.11	11.43
S&P	1,289.12	1

2

### 3.12.6 QueryTable XML

4 The XML expressing the definition of the QueryTable indicates that it is using the connection whose Id value  
 5 is 1 (connectionId):

```
6 <queryTable ... name="msn" connectionId="1" autoFormatId="16"
7   applyNumberFormats="0" applyBorderFormats="0" applyFontFormats="1"
8   applyPatternFormats="1" applyAlignmentFormats="0"
9   applyWidthHeightFormats="0"/>
```

### 10 3.12.7 Connection XML

11 The workbook connection whose Id is 1 is expressed below.

```
12 <connection id="1" name="Connection" type="4" refreshedVersion="3"
13   background="1" saveData="1">
14   <webPr sourceData="1" parsePre="1" consecutive="1" xl2000="1"
15     url="http://www.msn.com" htmlTables="1">
16     <tables count="1">
17       <x v="2"/>
18     </tables>
19   </webPr>
20 </connection>
```

21 Attributes and elements that have been previously discussed are not discussed here.

- 22 • type value of 4 indicates the connection is a web query connection.
- 23 • saveData value of 1 indicates that refreshed data will be kept in the sheet when saving the workbook.
- 24 0 indicates to remove the data from the workbook when saving the workbook.
- 25 • sourceData value of 1 indicates to import and parse the XML data rather than consume the web page's
- 26 HTML definition.
- 27 • parsePre value of 1 indicates that text in <PRE> tags is interpreted as tables.
- 28 • consecutive value of 1 means consecutive delimiters are treated as one delimiter
- 29 • url is the address indicating where to retrieve data for this query.
- 30 • htmlTables true means only import html tables

31 tables indicates which HTML table to import from the web page.

- 32 • v value of 2 indicates that the second table is the one to import.

### 1    3.12.8    Unused Connection

2    A connection can be expressed in a workbook, but not currently used. It remains until deleted by the user  
 3    explicitly. This simply means that there is no object or feature in the workbook; that is referencing the  
 4    connection.

### 5    3.12.9    ODBC

6    A database table imported to the grid, where the data provider is ODBC:

	A	B	C	D	
1	ID	Field1	Field2	Field3	
2	3	Foo	Bar	Goo	
3	4	The	Quick	Brown	
4	5	Fox	Jumped	Over	
5	6	The	Lazy	Dog	
6					

8    When a table object is used to render external data, it is associated with a QueryTable object to store the  
 9    properties used when a range is associated with external data. Therefore, the Table object references the  
 10   QueryTable name, which in turns references connectionId to identify the connection in the workbook  
 11   connections part.

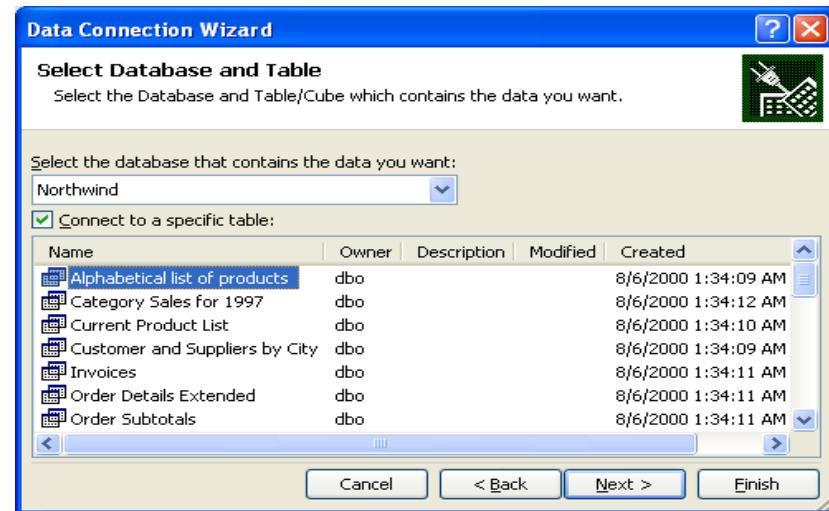
### 12    3.12.10    Connection XML

```
13 <connection id="4" name="Query from MS Access Database" type="1"
14   refreshedVersion="3" background="1" saveData="1">
15   <dbPr connection="DSN=MS Access"
16     Database;DBQ=E:\...\Database1.accdb;DefaultDir=E:\Documents and
17     Settings\chadroth\Desktop;DriverId=25;FIL=MS
18     Access;MaxBufferSize=2048;PageTimeout=5;" 
19     command="SELECT Table1.ID,
20       Table1.Field1, Table1.Field2,
21       Table1.Field3_x000d__x000a_FROM
22       `E:\...\Database1.accdb`.Table1 Table1"/>
23 </connection>
```

- 24     • type value of 1 indicates ODBC connection type.
- 25     • command contents are an SQL select statement.

### 26    3.12.11    SQL

27    An implementation might use a data connection wizard to connect to a SQL table; for example:



1

- 2 The resulting data is rendered in the grid:

	A	B	C	D
1				
2	<b>SQL Connection</b>			
3	ShippedDate	OrderID	Subtotal	
4	7/16/1996 0:00	10248	440	
5	7/10/1996 0:00	10249	1863.4	
6	7/12/1996 0:00	10250	1552.6	
7	7/15/1996 0:00	10251	654.06	
8	7/11/1996 0:00	10252	3597.9	
9	7/16/1996 0:00	10253	1444.8	
10	7/23/1996 0:00	10254	556.62	
11	7/15/1996 0:00	10255	2490.5	
12	7/17/1996 0:00	10256	517.8	
13	7/22/1996 0:00	10257	1119.9	
14	7/23/1996 0:00	10258	1614.88	
15	7/25/1996 0:00	10259	100.8	
16	7/29/1996 0:00	10260	1504.65	
17	7/30/1996 0:00	10261	448	
18	7/25/1996 0:00	10262	584	

- 4 In this example, a table object is used to render external data, it is associated with a QueryTable object to store  
5 the properties used when a range is associated with external data. Therefore, the Table object references the  
6 QueryTable name, which in turns references connectionId to identify the connection in the workbook  
7 connections part.

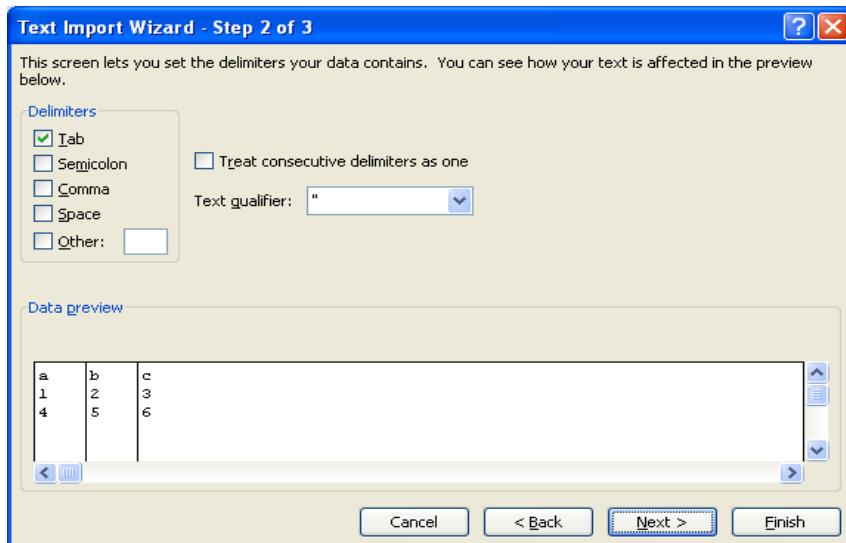
## 3.12.12 Connection XML

```
2 <connection id="6" odcFile="c:\...\xlextdat8_Northwind_Summary_of_Sales_by  
3 Year.odc" keepAlive="1"  
4 name="xlextdat8_Northwind_Summary_of_Sales_by_Year"  
5 type="5" refreshedVersion="3" background="1" saveData="1">  
6 <dbPr connection="Provider=SQLOLEDB.1;Integrated Security=SSPI;  
7 Persist Security Info=True;Initial Catalog=Northwind;Data  
8 Source=xlextdat8;Use Procedure for Prepare=1;Auto  
9 Translate=True;Packet Size=4096;Workstation ID=CHADROTH012;Use  
10 Encryption for Data=False;Tag with column collation when  
11 possible=False" command=""Northwind".&quot;dbo&quot;  
12 .&quot;Summary of Sales by Year"" commandType="3"/>  
13 </connection>
```

- type value of 5 indicates that this connection is using an OLEDB data provider.
  - commandType value of 3 specifies that a table name is in command
  - command specifies a table name.

## 17    3.12.13    Text Import

## 18 Text Import settings:



- 20 Note that there are additional settings not pictured here.

- 21 The resulting data in the grid:

	A	B	C	D	
1					
2					Text Import
3		a	b		
4		1	2		
5		4	5		
6					

- 2 The range is associated with a QueryTable object. This query table definition references the connectionId used  
 3 to retrieve the data.

#### 4 **3.12.14 Connection XML**

```

5   <connection id="5" name="Text" type="6" refreshedVersion="3"
6     background="1" saveData="1">...
7     <textPr codePage="437" sourceFile="E:\ ...\\Text.txt">
8       <textFields count="3">
9         <textField type="text"/>
10        <textField position="5"/>
11        <textField type="skip" position="10"/>
12      </textFields>
13    </textPr>
14  </connection>
```

15 connection defines the connection

- 16 • type value of 6 indicates that this is a text import type of connection.

17 textPr expresses properties which are specific to text import connections.

- 18 • codePage value of 437 indicates that the text file is using the IBM PC (OEM) code page 437 character  
 19 set.  
 20 • sourceFile indicates where the file is located.

21 textFields expresses information about the particular fields in the text file.

- 22 • delimited value of 1 (default) indicates that the text is delimited (variable length). Since this example  
 23 uses the default value, it is not saved as part of the connection information.  
 24 • type indicates the data type (user-specified) of the particular field.  
 25 • position indicates the starting position of the field for fixed-width fields.  
 26 • thousands specifies the thousands separator character (not in this example, but of enough interest to  
 27 mention).  
 28 • tab, space, comma attributes with values of 1 would flag these characters as delimiters (not in this  
 29 example, but of enough interest to mention).

## 1    3.13    External Links

### 2    3.13.1    Overview

3    An *external link* is used to link a workbook to other workbook or to external data. The most frequent  
 4    occurrence for linking a workbook to other workbooks has to do with formulas. In this case, a formula  
 5    references a range or name defined in another workbook. Hyperlinks on cells and other spreadsheet objects  
 6    are also considered an external link. OLE links are yet another technology used to link the workbook to another  
 7    object. Finally, Dynamic Data Exchange (DDE) servers can be used to access external data. DDE servers are  
 8    accessed through formulas in the workbook.

9    The goal of the way in which external links are saved is to always write the target source in a relationship file,  
 10   so that external resources are easily discoverable in lightweight relationship XML rather than deep in the  
 11   application's XML.

### 12    3.13.2    Formula Example

13   Consider cells B2 and C2 in the following worksheet, Sheet1:

	A	B	C
1		Link to an external workbook range:	Link to an external workbook name:
2		=SUM('C:\[Source.xlsx]Sheet1'!\$A\$1:\$A\$3)	='C:\Source2.xlsx'!NameInExternalWorkbook
3			
4		<a href="#">W3C Hyperlink</a>	
5			

14   Here, the formulas themselves are displayed in the cells.

	A	B	C	D
1		Link to an external workbook range:	Link to an external workbook name:	
2			6	2
3				
4		<a href="#">W3C Hyperlink</a>		
5				

16   Here, the results of the formulas are displayed in the cells.

17   The formula is expressed in Sheet1's XML, as shown in the following subclause.

### 19    3.13.3    Sheet XML

20   The corresponding content from Sheet1.xml is:

```

1   <worksheet ...>
2     <dimension ref="B1:C4"/>
3     <sheetViews>
4       <sheetView tabSelected="1" workbookViewId="0">
5         <selection activeCell="B2" sqref="B2"/>
6       </sheetView>
7     </sheetViews>
8     <sheetFormatPr defaultRowHeight="15"/>
9     <cols>
10       <col min="1" max="1" width="1.7109375" customWidth="1"/>
11     </cols>
12     <sheetData>
13       <row r="1" spans="2:3" customFormat="1" ht="9" customHeight="1"/>
14       <row r="2" spans="2:3" customFormat="1">
15         <c r="B2">
16           <f>SUM([1]Sheet1!$A$1:$A$3)</f>
17           <v>6</v>
18         </c>
19         <c r="C2">
20           <f>[2]!NameInExternalWorkbook</f>
21           <v>2</v>
22         </c>
23       </row>
24       <row r="4" spans="2:3" customFormat="1">
25         <c r="B4" s="1" t="s">
26           <v>0</v>
27         </c>
28       </row>
29     </sheetData>
30     <hyperlinks>
31       <hyperlink ref="B4" r:id="rId1"/>
32     </hyperlinks>
33     <printOptions/>
34     <pageMargins left="0.7" right="0.7" top="0.75" bottom="0.75"
35       header="0.3" footer="0.3"/>
36     <headerFooter/>
37   </worksheet>

```

### 3.13.3.1 Cell B2

39 The formula expressed in cell B2 (cell B2 is the c element whose r="B2") is this:

40 SUM([1]Sheet1!\$A\$1:\$A\$3)

1 The external reference to another workbook in this case is tokenized to [1]. The value inside the brackets is a  
 2 1-based index to the externalReferences collection in the workbook part.

3 **3.13.3.2 Cell C2**

4 The formula expressed in cell C2 (cell C2 is the c element whose r is C2) is this:

5 [2]!NameInExternalWorkbook

6 The external reference to another workbook in this case is tokenized to [2]. The value inside the brackets is a  
 7 1-based index to the externalReferences collection in the workbook part.

8 **3.13.3.3 Workbook XML**

9 The corresponding content from workbook.xml is

```
10 <workbook ...>
11   <fileVersion lastEdited="4" lowestEdited="4" rupBuild="4012"/>
12   <workbookPr defaultThemeVersion="123820"/>
13   <bookViews>
14     <workbookView xWindow="360" yWindow="270" windowWidth="18735"
15       windowHeight="11445"/>
16   </bookViews>
17   <sheets>
18     <sheet name="Sheet1" sheetId="1" r:id="rId1"/>
19     <sheet name="Sheet2" sheetId="2" r:id="rId2"/>
20     <sheet name="Sheet3" sheetId="3" r:id="rId3"/>
21   </sheets>
22   <externalReferences>
23     <externalReference r:id="rId4"/>
24     <externalReference r:id="rId5"/>
25   </externalReferences>
26   <calcPr calcId="122211"/>
27   <webPublishing codePage="1252"/>
28 </workbook>
```

29 The workbook part's externalReferences collection indicates that there are two external workbook references  
 30 in this workbook. The first supporting external workbook data cache, also stored in this workbook, can be  
 31 found by following the relationship from the workbook whose Id value is rId4. The second supporting external  
 32 workbook data cache, also stored in this workbook, can be found by following the relationship from the  
 33 workbook whose Id value is rId5.

34 **3.13.4 Workbook Relationships**

35 The corresponding content from workbook.xml.rels is:

```

1   <Relationships xmlns="http://.../package/2006/relationships">
2     <Relationship Id="rId8" Type="http://.../sharedStrings"
3       Target="sharedStrings.xml"/>
4     <Relationship Id="rId3" Type="http://.../worksheet"
5       Target="worksheets/sheet3.xml"/>
6     <Relationship Id="rId7" Type="http://.../styles" Target="styles.xml"/>
7     <Relationship Id="rId2" Type="http://.../worksheet"
8       Target="worksheets/sheet2.xml"/>
9     <Relationship Id="rId1" Type="http://.../worksheet"
10    Target="worksheets/sheet1.xml"/>
11    <Relationship Id="rId6" Type="http://.../theme" Target="theme/theme1.xml"/>
12    <Relationship Id="rId5" Type="http://.../externalLink"
13      Target="externalLinks/externalLink2.xml"/>
14    <Relationship Id="rId4" Type="http://.../externalLink"
15      Target="externalLinks/externalLink1.xml"/>
16    <Relationship Id="rId9" Type="http://.../calcChain" Target="calcChain.xml"/>
17  </Relationships>
```

18 These relationship expressions indicate that cell B2 is supported by the external workbook data cache located  
19 at `externalLinks/externalLink1.xml` in the package. These relationship expressions also indicate that  
20 cell C2 is supported by the external workbook data cache located at `externalLinks/externalLink2.xml`  
21 in the package.

### 22 3.13.5 Supporting Workbook Cache (Cell C2)

23 The corresponding content from `externalLink2.xml` is:

```

24  <externalLink ...>
25    <externalBook xmlns:r="http://schemas.openxmlformats.org
26      /officeDocument/2006/relationships" r:id="rId1">
27      <sheetNames>
28        <sheetName val="Sheet1"/>
29        <sheetName val="Sheet2"/>
30        <sheetName val="Sheet3"/>
31      </sheetNames>
32      <definedNames>
33        <definedName name="NameInExternalWorkbook"
34          refersTo="='Sheet1'!$B$1"/>
35      </definedNames>
```

```

1   <sheetDataSet>
2     <sheetData sheetId="0">
3       <row r="1">
4         <cell r="B1">
5           <v>2</v>
6         </cell>
7       </row>
8     </sheetData>
9     <sheetData sheetId="1"/>
10    <sheetData sheetId="2"/>
11  </sheetDataSet>
12 </externalBook>
13 </externalLink>
```

14 Supporting workbook data caches store the top-level structure of the workbook (sheet names, defined names,  
15 cell table). Only the cells referenced are cached. This supporting workbook data cache indicates that the  
16 workbook being referenced by C2 has three sheets, whose names are "Sheet1", "Sheet2", and "Sheet3", and  
17 has a defined name of "NameInExternalWorkbook". Additionally, the cell table shows that cell B1 in this  
18 workbook is the cell being referenced. A copy of the cell table is stored locally, inside the workbook containing  
19 the external link.

20 The r:id="rId1" on the top level externalLink element indicates the Id of the relationship from the  
21 externalLink2.xml part, which indicates the location of the actual external workbook.

### 22 **3.13.6 External Link (Cell C2)**

23 The corresponding content from externalLink2.xml.rels is

```

24   <Relationships ...>
25     <Relationship Id="rId1" Type="http://.../externalLinkPath"
26       Target="file:///C:/Source2.xlsx" TargetMode="External"/>
27   </Relationships>
```

28 This relationship indicates that the supporting workbook that C2 references resides on the local drive, at  
29 c:\source2.xlsx.

### 30 **3.13.7 Supporting Workbook Cache (Cell B2)**

31 The corresponding content from externalLink1.xml is:

```

1   <externalLink ...>
2     <externalBook xmlns:r="http://.../relationships" r:id="rId1">
3       <sheetNames>
4         <sheetName val="Sheet1"/>
5         <sheetName val="Sheet2"/>
6         <sheetName val="Sheet3"/>
7       </sheetNames>
8       <sheetDataSet>
9         <sheetData sheetId="0">
10        <row r="1">
11          <cell r="A1">
12            <v>1</v>
13          </cell>
14        </row>
15        <row r="2">
16          <cell r="A2">
17            <v>2</v>
18          </cell>
19        </row>
20        <row r="3">
21          <cell r="A3">
22            <v>3</v>
23          </cell>
24        </row>
25      </sheetData>
26      <sheetData sheetId="1"/>
27      <sheetData sheetId="2"/>
28    </sheetDataSet>
29  </externalBook>
30 </externalLink>
```

31 This supporting workbook data cache indicates that the workbook being referenced by B2 has three sheets,  
32 whose names are "Sheet1", "Sheet2", and "Sheet3". Additionally, the cell table shows that cells A1, A2, and A3,  
33 whose values are 1, 2, and 3, respectively, in this workbook are being referenced. A copy of the cell table is  
34 stored locally, inside the workbook containing the external link.

35 The r:id="rId1" on the top level externalLink element indicates the Id of the relationship from the  
36 externalLink1.xml part, which indicates the location of the actual external workbook.

### 37 3.13.8 External Link (Cell B2)

38 The corresponding content from externalLink1.xml.rels is

```

1   <Relationships ...>
2     <Relationship Id="rId1" Type="http://.../externalLinkPath"
3       Target="file:///C:/Source.xlsx" TargetMode="External"/>
4   </Relationships>

```

5 This relationship indicates that the supporting workbook that C2 references resides on the local drive, at  
6 c:\source.xlsx.

### 7 3.13.9 Hyperlink Example

8 Consider the following worksheet:

A	B	C
1	Link to an external workbook range: =SUM('C:\[Source.xlsx]Sheet1'!\$A\$1:\$A\$3)	Link to an external workbook name: ='C:\Source2.xlsx'!NameInExternalWorkbook
3		
4	<a href="http://www.w3.org/">W3C Hyperlink</a>	
5		

10 Cell B4 contains a hyperlink, whose friendly name is "W3C Hyperlink", and whose target is  
11 "http://www.w3.org/".

### 12 3.13.10 Worksheet XML

13 See §3.13.3 for the full XML. Here is the snippet expressing the hyperlink information, whose collection  
14 appears immediately after the sheetData collection in this example.

```

15   <hyperlinks>
16     <hyperlink ref="B4" r:id="rId1"/>
17   </hyperlinks>

```

18 The hyperlink XML indicates that cell B4 of this sheet has a hyperlink, whose target can be found by following  
19 the relationship whose Id="rId1" from the current sheet. The 'friendly' name of the hyperlink is stored in the  
20 cell definition.

### 21 3.13.11 Relationship

22 The corresponding content from sheet1.xml.rels is:

```

23   <Relationships ...>
24     <Relationship Id="rId1" Type="http://.../hyperlink"
25       Target="http://www.w3.org/" TargetMode="External"/>
26   </Relationships>

```

27 This hyperlink points external to the workbook (TargetMode="External"), and the URL is found in the value of  
28 Target to be "http://www.w3.org/".

## 1    3.14    Volatile Dependencies

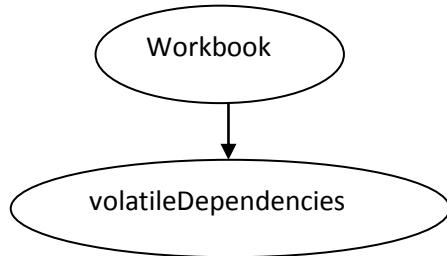
### 2    3.14.1    Overview

3    The volatile dependencies part provides a supporting cache of data for Real Time Data (RTD) and CUBE  
 4    functions in the workbook. Both of these types of functions require connectivity to external servers to retrieve  
 5    their data. For RTD functions, an RTD interface has been defined for how to provide (on the server side) and  
 6    retrieve (on the client side) external data. Similarly, CUBE functions are able to access data in OLAP cubes. Each  
 7    type of function has its own function syntax resulting in a specific piece of information being returned.

8    In the event that the server providing the data is unavailable, a spreadsheet application may want to cache the  
 9    most recently retrieved values so that when recalculating the spreadsheet, calculated results may be acquired  
 10   instead of errors.

11   The volatile dependencies part provides that cache of data and supporting information about these functions  
 12   and their data servers and connections.

### 13   3.14.2    File Architecture - Relationships



18   The workbook holds the relationship to the volatile dependencies part.

### 19   3.14.3    Example

20   In this example, both a Real Time Data (RTD) function and CUBE functions are in use.

#### 21   3.14.3.1    Illustration

	A	B
1	aaa: 672	
2		
3	Corporate	
4		\$80,450,596.98
5	Set	
6		#N/A
7		#N/A
8	Not Applicable	
9	Growth in Customer Base Goal	
10		

23   (values shown)

	A
1	=RTD("jrtdx.rtd","aaa")
2	
3	=CUBEMEMBER("xlextdat9 Adventure Works DW Adventure Works", "[Department].[Departments].[Corporate]")
4	=CUBEVALUE("xlextdat9 Adventure Works DW Adventure Works", A3)
5	=CUBESET("xlextdat9 Adventure Works DW Adventure Works", "[Customer].[Customer Geography].[All Customers].[United Kingdom].children", "Set")
6	=CUBERANKEDMEMBER("xlextdat9 Adventure Works DW Adventure Works", \$A\$3,ROW(A1))
7	=CUBESETCOUNT(A3)
8	=CUBEMEMBERPROPERTY("xlextdat9 Adventure Works DW Adventure Works", "[Product].[Product].[All Products].[Blade]", "Class")
9	=CUBEKPIMEMBER("xlextdat9 Adventure Works DW Adventure Works", "Growth in Customer Base", 2)
10	

2 (functions shown)

### 3 3.14.3.2 volatileDependencies.xml

```

4 <volTypes xmlns="...">
5   <volType type="realTimeData">
6     <main first="jrtdx.rtd">
7       <tp t="s">
8         <v>aaa: 4447</v>
9         <stp/>
10        <stp>aaa</stp>
11        <tr r="A1" s="1"/>
12      </tp>
13    </main>
14  </volType>
15  <volType type="olapFunctions">
16    <main first="xlextdat9 Adventure Works DW Adventure Works">
17      <tp t="e">
18        <v>#N/A</v>
19        <stp>1</stp>
20        <tr r="A6" s="1"/>
21        <tr r="A9" s="1"/>
22        <tr r="A8" s="1"/>
23        <tr r="A5" s="1"/>
24        <tr r="A4" s="1"/>
25        <tr r="A3" s="1"/>
26      </tp>
27    </main>
28  </volType>
29 </volTypes>
```

### 30 3.14.3.2.1 RTD Supporting Data

31 /voltypes/volType@type indicates that the supporting information pertains to an RTD function call. Valid  
32 values are realTimeData and olapFunctions

1    /volTypes/volType/main@first indicates the ProgId of the RTD server. This value corresponds to the  
 2    first argument of an RTD function in a worksheet.

3    /volTypes/volType/main/tp contains a listing of topics within the main topic. For the RTD function, this  
 4    collection will express the remaining parameters of the function, and indicate the last known value and data  
 5    type of that value.

6    /volTypes/volType/main/tp@t indicates the data type of the value associated with this topics. For this  
 7    RTD example, the value is "aaa: 4447" whose data type is string.

8    /volTypes/volType/main/tp/v expresses the last known value of this RTD function, "aaa: 4447".

9    /volTypes/volType/main/tp/stp expresses the remaining topics, or function parameters, for this RTD  
 10   function. Notice that in the example, the second parameter is left empty, and the third parameter is "aaa".

11   /volTypes/volType/main/tp/tr expresses the cells which are dependent on this particular set of topics,  
 12   and which are associated with this supporting information.

### 13    3.14.3.2.2      Cube Function Supporting Data

14   Cube functions use the same persistence structure as the RTD supporting data, but the information is  
 15   interpreted slightly differently. At a high level, main@first indicates the connection name, and the  
 16   tr elements spell out the cells with cube function calls dependent on this connection. In most cases (when  
 17   the <stp> value is equal to "1") the remaining information can be ignored.

18   /volTypes/volType/main@first indicates the connection name for the related cube functions.

19   /volTypes/volType/main/tp@t can be ignored when stp value is 1.

20   /volTypes/volType/main/tp/v contains an error value of "#N/A", which can be ignored when stp value  
 21   is 1.

22   /volTypes/volType/main/tp/stp value of 1 indicates that all of the related cells with calling cube  
 23   functions have been refreshed.

24   /volTypes/volType/main/tp/tr expresses the cells contain cube functions which are dependent on this  
 25   connection, and which are associated with this supporting information.

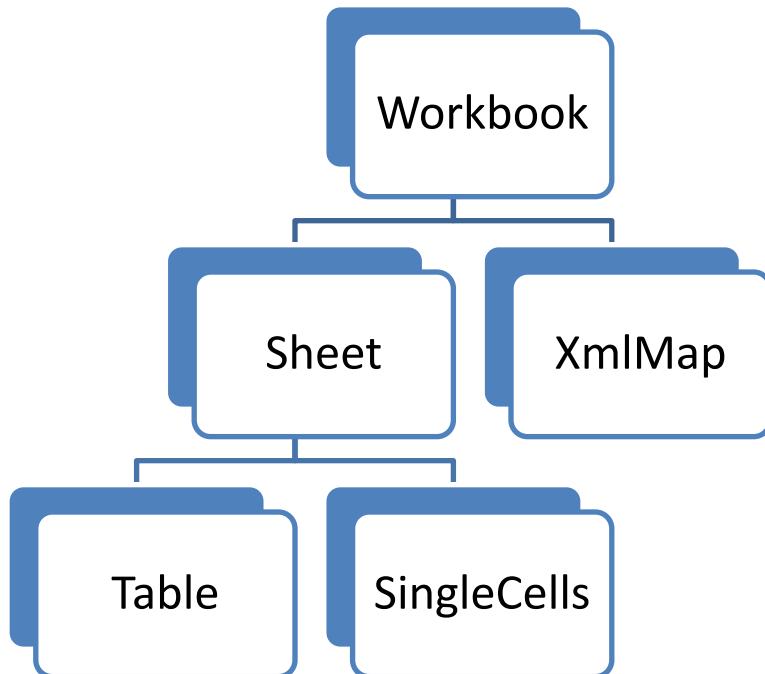
## 26    3.15   Custom XML Mappings

### 27    3.15.1   Overview

28   With the pervasiveness of XML data structures and XML web services, it is appropriate for a spreadsheet  
 29   application to consume XML data structures and render the data in the sheet grid. Furthermore it is  
 30   appropriate and desirable for the spreadsheet application to be able to generate XML data structures. Finally,  
 31   since XML is extensible, the kinds of XML structures that can be consumed or produced by a spreadsheet  
 32   application should be as varied as the number of XML schemas that exist.

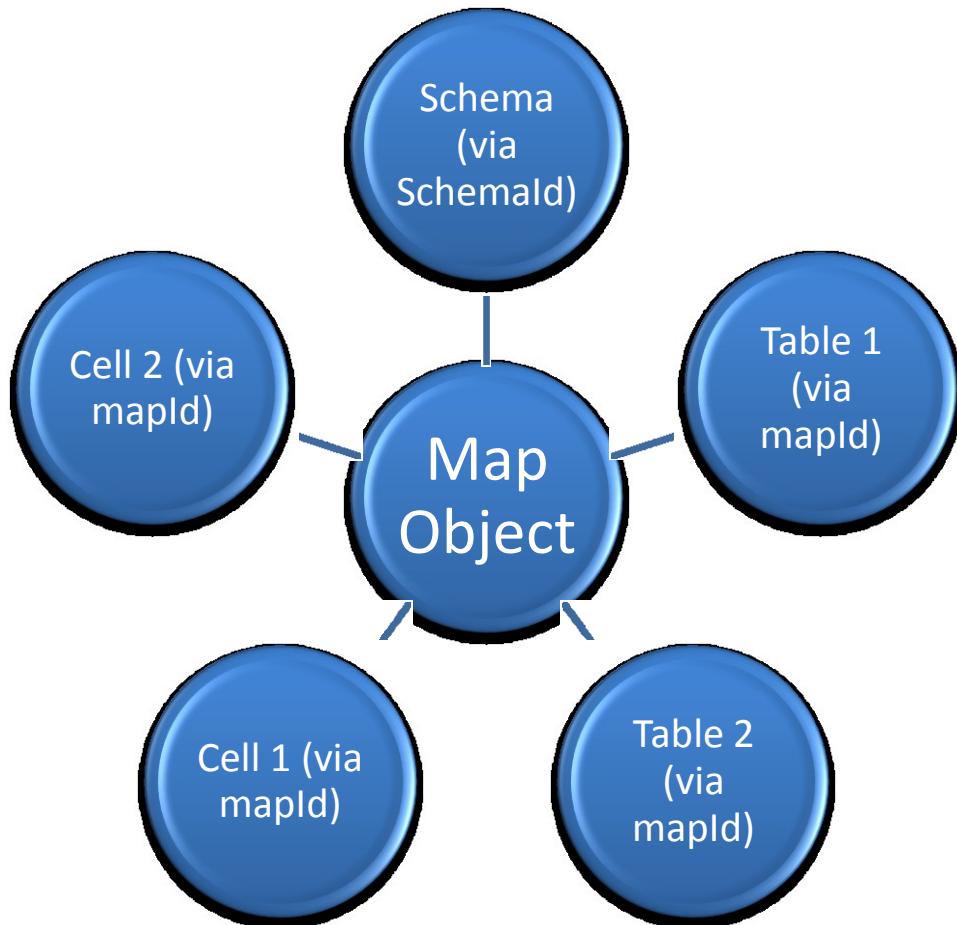
- 1 The XML Mapping feature enables adding arbitrary XML data structures and arbitrary XML schema definitions to the workbook, then mapping the various XML nodes to cells and ranges in the workbook. Once an XML Mapping is set up, the application is able to import and export XML instance structures according to the schema definition.
- 5 While the original schema or XML definition may reside on disk or at some file location outside the workbook, a copy of the schema is stored in the workbook.
- 7 Every time an XML instance or schema is added to the workbook, a new map object is created which ties together the schemas and where the various elements are mapped in the workbook.
- 9 Additional properties are stored on each cell and each column of a Table that has an XML map association.

#### 10 **3.15.2 File Architecture - Relationships**



- 11
- 12 The workbook owns sheets and the xmlMap definitions. Each sheet references Tables and single cells which
- 13 are mapped to XML structures.

1    **3.15.3    Conceptual Model**



2

3

- 4    Conceptually all the objects reference a common Map Object, by @id. It is in this way that they come together  
5    into a single working feature that can import and export custom XML data.

6    **3.15.4    Example**

- 7    In this example both single cells and a Table are shown to have a data binding to an XML structure.

1    3.15.4.1      Illustration

A	B	C	D	E	F
1					
2	currency	detailed	total-sum		
3	USD	FALSE		556.9	
4					
5	First	Last	Email		
6	Fred	Landis	f.landis@nanonull.com		
7					
8	type	expto	Date	expense	description
9	Lodging	Sales	1/1/2003	122.11	
10	Lodging	Development	1/2/2003	122.12	Played penny arcade
11	Lodging	Marketing	1/2/2003	299.45	Treated Clients
12	Entertainment	Development	1/2/2003	13.22	Bought signed "XMLSPY Handbo
13					
14					
15					

- 3    The table in B8:G12 is also data bound to an XML mapping object. The first column, titled "type", is associated  
4    with the XML Map named "expense-report\_Map", specifically the attribute identified by the xpath expression  
5    /expense-report/expense-item@type pointing into the corresponding XML structure. In similar fashion,  
6    each of the columns in the Table correspond with elements or attributes in the related XML Map structure.
- 7    Additionally, cells B3:D3 and B6:D6 are each bound to a single, non-repeating element or attribute from the  
8    same XML Map structure. For example, cell B3 corresponds to /expense-report@currency.
- 9    In this way XML instance structures can be refreshed into the cells and Table region, and XML instance  
10   structures can be generated from the data in those ranges of the spreadsheet. In other words, XML structures  
11   can be imported and exported to and from the worksheet via the XML Mapping feature.

12    3.15.4.2      The xmlMap XML

- 13    The xmlMaps part stores the custom schema that has been added to the workbook, and also stores the  
14    xmlMap definitions. There can be multiple schemas and xmlMaps in a single workbook.

```

15 <MapInfo SelectionNamespaces="">
16   <Schema ID="Schema1">
17     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
18       <xsd:element nillable="true" name="expense-report">
19         <xsd:complexType>
20           <xsd:sequence minOccurs="0">
21             <xsd:element minOccurs="0" nillable="true"
22               name="Person" form="unqualified">
23               <xsd:complexType>
24                 <xsd:sequence minOccurs="0">
25                   <xsd:element minOccurs="0" nillable="true"
26                     type="xsd:string" name="First" form="unqualified"></xsd:element>
```

```

1                               <xsd:element minOccurs="0" nillable="true"
2 type="xsd:string" name="Last" form="unqualified"></xsd:element>
3                               <xsd:element minOccurs="0" nillable="true"
4 type="xsd:string" name="Title" form="unqualified"></xsd:element>
5                               <xsd:element minOccurs="0" nillable="true"
6 type="xsd:string" name="Phone" form="unqualified"></xsd:element>
7                               <xsd:element minOccurs="0" nillable="true"
8 type="xsd:string" name="Email" form="unqualified"></xsd:element>
9 </xsd:sequence>
10 </xsd:complexType></xsd:element>
11 <xsd:element minOccurs="0" maxOccurs="unbounded"
12 nillable="true" name="expense-item" form="unqualified">
13     <xsd:complexType>
14         <xsd:all>
15             <xsd:element minOccurs="0" nillable="true"
16 type="xsd:date" name="Date" form="unqualified"></xsd:element>
17             <xsd:element minOccurs="0" nillable="true"
18 type="xsd:double" name="expense" form="unqualified"></xsd:element>
19             <xsd:element minOccurs="0" nillable="true"
20 type="xsd:string" name="description" form="unqualified"></xsd:element>
21             <xsd:element minOccurs="0" nillable="true"
22 name="Misc" form="unqualified">
23             <xsd:complexType>
24                 <xsd:attribute name="misctype"
25 form="unqualified" type="xsd:string"></xsd:attribute>
26             </xsd:complexType>
27         </xsd:all>
28         <xsd:attribute name="type" form="unqualified"
29 type="xsd:string"></xsd:attribute>
30             <xsd:attribute name="expto" form="unqualified"
31 type="xsd:string"></xsd:attribute>
32             </xsd:complexType>
33         </xsd:element>
34     </xsd:sequence>
35     <xsd:attribute name="currency" form="unqualified"
36 type="xsd:string"></xsd:attribute>
37         <xsd:attribute name="detailed" form="unqualified"
38 type="xsd:boolean"></xsd:attribute>
39         <xsd:attribute name="total-sum" form="unqualified"
40 type="xsd:double"></xsd:attribute>
41         </xsd:complexType>
42     </xsd:element>
43 
```

```

1          </xsd:schema>
2      </Schema>
3      <Map ID="1" Name="expense-report_Map" RootElement="expense-report"
4 SchemaID="Schema1" ShowImportExportValidationErrors="false" AutoFit="true"
5 Append="false" PreserveSortAFLLayout="true" PreserveFormat="true">
6          <DataBinding FileBinding="true" DataBindingLoadMode="1"/>
7      </Map>
8  </MapInfo>

9  • /MapInfo@SelectionNamespaces ties the prefix to the actual namespace. This is used when
10 writing xpath expressions at runtime against the XML instance structures, because the xpath
11 expressions use namespace prefixes instead of the fully spelled out namespace.
12 • /MapInfo/Schema stores the schemas for a particular XML map object. There can be multiple
13 <Schema> elements in a workbook, one for each XML map.
14 • /MapInfo/Schema@ID identifies the schema collection used to define a particular XML map object.
15 • /MapInfo/Map/@ID identifies the map object.
16 • /MapInfo/Map@Name is the friendly name of the map object.
17 • /MapInfo/Map@RootElement is the name of the root element of the XML instance (schemas can
18 define more than one root node).
19 • /MapInfo/Map@SchemaID identifies which schema collection the map uses.
20 • /MapInfo/Map@ShowImportExportValidationErrors indicates that when an XML instance is
21 imported or exported, the schema should be used to validate the instance, and schema errors should
22 be shown to the user.
23 • /MapInfo/Map@AutoFit indicates that after refresh, all the cells should be 'best fitted'.
24 • /MapInfo/Map@Append means that when refreshed, don't discard existing data, but append new
25 data to it.
26 • /MapInfo/Map@PreserveSortAFLLayout indicates whether to keep filters on (Tables).
27 • /MapInfo/Map@PreserveFormat indicates whether to keep the cell formatting applied or re-apply
28 based on schema data type.

```

### 3.15.4.3 The Table XML

The only difference with table definitions that are bound to XML is that @tableType="xml" and each column has an additional set of xml-specific properties, contained in the <xmlColumnPr> collection, which appears once for every column in the Table which has an XML data binding.

```

33  <table xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/5/main"
34  id="7" name="Table7" displayName="Table7" ref="B8:G12" tableType="xml"
35  totalsRowShown="0" connectionId="1">
36      <autoFilter ref="B8:G12"/>
37      <tableColumns count="6">
38          <tableColumn id="1" uniqueName="type" name="type">
39              <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/@type"
40  xmlDataType="string"/>

```

```

1      </tableColumn>
2      <tableColumn id="2" uniqueName="expto" name="expto">
3          <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/@expto"
4      xmlDataType="string"/>
5      </tableColumn>
6      <tableColumn id="3" uniqueName="Date" name="Date">
7          <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/Date"
8      xmlDataType="date"/>
9      </tableColumn>
10     <tableColumn id="4" uniqueName="expense" name="expense">
11         <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/expense"
12     xmlDataType="double"/>
13     </tableColumn>
14     <tableColumn id="5" uniqueName="description" name="description">
15         <xmlColumnPr mapId="1" xpath="/expense-report/expense-
16 item/description" xmlDataType="string"/>
17     </tableColumn>
18     <tableColumn id="6" uniqueName="misctype" name="misctype">
19         <xmlColumnPr mapId="1" xpath="/expense-report/expense-
20 item/Misc/@misctype" xmlDataType="string"/>
21     </tableColumn>
22     </tableColumns>
23     <tableStyleInfo name="TableStyleMedium7" showFirstColumn="0"
24 showLastColumn="0" showRowStripes="1" showColumnStripes="0"/>
25 </table>
```

26 The column in the Table titled "type" is bound to an XML mapping, whose map object Id @mapId is "1". The  
27 @xpath value indicates an xpath expression to which this Table column is associated. In this example the Table  
28 column "type" corresponds to @type, which is an attribute of the <expense-item> collection. The  
29 corresponding custom schema definition for @type indicates a data type of string. This is stored as an xml  
30 column property as well, in @xmlDataType. This is used for interpreting the data on import and export, and is  
31 also used to format the cells for proper rendering in the range.

32 The remaining columns have similar properties set and can be understood from the discussion above.

### 3.15.4.4 Single Cell XML

34 Contents of tableSingleCells.xml

```

35     <singleXmlCells
36     xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/5/main">
37         <singleXmlCell id="1" name="Table1" displayName="Table1" r="B3"
38     connectionId="1">
39         <xmlCellPr id="1" uniqueName="currency">
```

```

1          <xmlPr mapId="1" xpath="/expense-report/@currency"
2      xmlDataType="string"/>
3      </xmlCellPr>
4  </singleXmlCell>
5  <singleXmlCell id="2" name="Table2" displayName="Table2" r="C3"
6 connectionId="1">
7      <xmlCellPr id="1" uniqueName="detailed">
8          <xmlPr mapId="1" xpath="/expense-report/@detailed"
9      xmlDataType="boolean"/>
10     </xmlCellPr>
11  </singleXmlCell>
12  <singleXmlCell id="3" name="Table3" displayName="Table3" r="D3"
13 connectionId="1">
14      <xmlCellPr id="1" uniqueName="total-sum">
15          <xmlPr mapId="1" xpath="/expense-report/@total-sum"
16      xmlDataType="double"/>
17      </xmlCellPr>
18  </singleXmlCell>
19  <singleXmlCell id="4" name="Table4" displayName="Table4" r="B6"
20 connectionId="1">
21      <xmlCellPr id="1" uniqueName="First">
22          <xmlPr mapId="1" xpath="/expense-report/Person/First"
23      xmlDataType="string"/>
24      </xmlCellPr>
25  </singleXmlCell>
26  <singleXmlCell id="5" name="Table5" displayName="Table5" r="C6"
27 connectionId="1">
28      <xmlCellPr id="1" uniqueName="Last">
29          <xmlPr mapId="1" xpath="/expense-report/Person/Last"
30      xmlDataType="string"/>
31      </xmlCellPr>
32  </singleXmlCell>
33  <singleXmlCell id="6" name="Table6" displayName="Table6" r="D6"
34 connectionId="1">
35      <xmlCellPr id="1" uniqueName="Email">
36          <xmlPr mapId="1" xpath="/expense-report/Person/Email"
37      xmlDataType="string"/>
38      </xmlCellPr>
39  </singleXmlCell>
40 </singleXmlCells>
```

41 A single cell which has been mapped to an XML node is expressed in much the same way that an entire table is  
42 expressed.

- 1 The `<singleXmlCell>` collection is the top level object, like the Table, which identifies the cell in question.
- 2 The `<xmlCellPr>` collection identifies the name for the only 'column' in this structure, the single cell. In this way it is much like a table column definition and the table column-level properties.
- 4 The `<xmlPr>` collection expresses the xml properties for this cell.

## 3.16 Formulas

### 3.16.1 Introduction

7 A SpreadsheetML *formula* is an equation that performs a calculation that typically involves the values of one or  
8 more cells in one or more worksheets.

9 A formula is an expression that can contain the following: constants (§3.16.2), operators (§3.16.3), cell  
10 references (§3.16.4), calls to functions (§3.16.5), and names (§3.16.6).

11 Consider the formula `PI()*(A2^2)`. In this case,

- 12 • `PI()` results in a call to the function PI, which returns the value of  $\pi$ .
- 13 • The cell reference `A2` returns the value in that cell.
- 14 • 2 is a numeric constant.
- 15 • The caret (`^`) operator raises its left operand to the power of its right operand.
- 16 • The parentheses, `(` and `)`, are used for grouping.
- 17 • The asterisk (`*`) operator performs multiplication of its two operands.

### 3.16.2 Constants

19 A *constant* is a predefined value that is not calculated, and, therefore, does not change. A constant can be any  
20 of the following:

- 21 • A real number
- 22 • The logical values TRUE and FALSE
- 23 • A string literal
- 24 • An array constant
- 25 • An error constant

### 3.16.3 Operators

27 An *operator* is a symbol that specifies the type of operation to perform on one or more operands. There are  
28 arithmetic, comparison, text, and reference operators.

Operators			
Family	Operator	Description	Precedence

Operators			
Reference operators	:	Binary range operator, which takes two cell reference (§3.16.3) operands, and results in one reference to the cells inclusive of, and between, those references. For example, SUM(B5:C15), which references 11 cells.	highest
	,	Binary union operator, which takes two cell reference (§3.16.3) operands, and results in one reference to all those, possibly non-contiguous, cells. For example, SUM((B5:B15,D5:D15)), which references 22 cells, 11 from column B, and 11 from column D. The grouping parentheses are necessary to indicate that the comma is an operator rather than a punctuator separating two arguments.	
	space	Binary intersection operator, which takes two cell reference (§3.16.3) operands, and results in one reference to those, possibly non-contiguous, cells that are common. If the intersection is empty, the result is #NULL!. For example, COUNT((B1:C1) (C1:D1)), which results in a reference to C1, while COUNT((B1:D1) (B1,D1)) results in a single reference to B1 and D1.	
Arithmetic operators	-	Unary minus	
	%	Percentage (unary postfix), which divides its operand by 100. For example, 10.5%, which results in 0.105.	
	^	Exponentiation	
	*	Multiplication	
	/	Division	
	+	Addition	
	-	Subtraction	
Text operator	&	Text concatenation (Each of the two operands is converted to text, if necessary, before concatenation.)	lowest
Comparison operators	=	Equal-to	
	<>	Not-equal-to	
	<	Less-than	
	<=	Less-than or equal-to	
	>	Greater-than	

Operators		
	>=	Greater-than-or-equal-to

1

2 Given that cell E38 contains the value 4, and cell F38 contains the value 2, the formula

3  $((-1+E38^2)*3-F38)/2$

4 produces the result 21.5.

## 5 3.16.4 Cell References

6 Each set of horizontal cells in a worksheet is a *row*, and each set of vertical cells is a *column*. A cell's row and  
7 column combination designates the location of that cell.

8 A *cell reference* designates one or more cells on the same worksheet. Using references, one can:

- 9 • Use data contained in different parts of the same worksheet in a single formula.
- 10 • Use the value from a single cell in several formulas.
- 11 • Refer to cells on other sheets in the same workbook, and even to other workbooks. (References to  
12 cells in other workbooks are called *links*.)

13 There are two cell reference styles: A1 and R1C1.

- 14 • In the A1 reference style, each row has a numeric heading numbered sequentially from the top down,  
15 starting at 1. Each column has an alphabetic heading named sequentially from left-to-right, A–Z, then  
16 AA–AZ, BA–BZ, ..., ZA–ZZ, AAA–AAZ, ABA–ABZ, and so on. Column letters are not case-sensitive.

17  
18 A relative reference to a single cell is written as its column letter immediately followed by its row  
19 number. A relative reference to a whole row is written as its row number. A relative reference to a  
20 whole column is written as its column letter. A reference to a range of two or more cells is written as  
21 two single-cell references separated by the binary range operator (:). An absolute A1 reference is  
22 made up of a cell's column letter followed by its row number, with each being preceded by a dollar  
23 character (\$). For example, A2, B34, and B5:D8 are relative A1 references. \$A\$2, \$B\$34, and  
24 \$B\$5:\$D\$8 are absolute A1 references. \$A2, B\$34, and \$B5:D\$8 are mixed A1 references.

- 25 • In the R1C1 reference style, each row has a numeric heading numbered sequentially from the top down,  
26 starting at 1. Each column has a numeric heading numbered sequentially from left-to-right,  
27 starting at 1.

28  
29 A whole row is referenced by omitting the column, and a whole column is referenced by omitting the  
30 row. An absolute row or column reference uses absolute row or column numbers, respectively. A  
31 relative row or column reference uses, respectively, row or column offsets from the cell containing the  
32 formula, with a negative offset indicating a row to the left or a column above, and a positive offset  
33 indicateing a row to the right or a column below. Specifying an offset of zero is equivalent to omitting

1 that offset and its delimiting brackets. For example, R[-2]C refers to the cell two rows up and in the  
 2 same column, R[2]C[2] refers to the cell two rows down and two columns to the right, R2C2 refers  
 3 to the cell in the second row and in the second column, R[-1] refers to the entire row above the  
 4 active cell, and R refers to the current row.

5 The R1C1 alternate reference style can only be used at runtime.

### 6 **3.16.5 Functions**

7 A *function* is a named formula that takes zero or more arguments, performs an operation, and, optionally,  
 8 returns a result. Some examples of function calls are: PI(), POWER(A1,B3), and SUM(C6:C10).

9 There are more than 300 predefined functions defined by this Office Open XML specification. User-defined  
 10 functions are also permitted.

### 11 **3.16.6 Names**

12 A *name* is an alias for a constant, a cell reference, or a formula. A name in a formula can make it easier to  
 13 understand the purpose of that formula. For example, the formula SUM(FirstQuarterSales) is easier to  
 14 identify than SUM(C20:C30).

### 15 **3.16.7 Types and Values**

16 Each *expression* has a type. SpreadsheetML formulas support the following types: array, error, logical, number,  
 17 and text.

18 An array value or constant represents a collection of one or more elements, whose values can have any type  
 19 (i.e., the elements of an array need not all have the same type).

### 20 **3.16.8 Error values**

21 The evaluation of an expression can result in an error having one of a number of *error values*. These error  
 22 values are:

Error Value	Reason for Occurrence
#DIV/0!	Intended to indicate when any number, including zero, is divided by zero.
#N/A	Intended to indicate when a designated value is not available. For example, Some functions, such as SUMX2MY2, perform a series of operations on corresponding elements in two arrays. If those arrays do not have the same number of elements, then for some elements in the longer array, there are no corresponding elements in the shorter one; that is, one or more values in the shorter array are not available. This error value can be produced by calling the function NA.
#NAME?	Intended to indicate when what looks like a name is used, but no such name has been defined. For example, XYZ/3, where XYZ is not a defined name. Total is & A10, where neither Total nor is is a defined name. Presumably, "Total is " & A10 was intended. SUM(A1:C10), where the range A1:C10 was intended.

Error Value	Reason for Occurrence
#NULL!	Intended to indicate when two areas are required to intersect, but do not. For example, In the case of $\text{SUM}(\text{B1} \text{ } \text{C1})$ , the space between B1 and C1 is treated as the binary intersection operator, when a comma was intended.
#NUM!	Intended to indicate when an argument to a function has a compatible type, but has a value that is outside the domain over which that function is defined. (This is known as a <i>domain error</i> .) For example, Certain calls to ASIN, ATANH, FACT, and SQRT might result in domain errors.
	Intended to indicate that the result of a function cannot be represented in a value of the specified type, typically due to extreme magnitude. (This is known as a <i>range error</i> .) For example, FACT(1000) might result in a range error.
#REF!	Intended to indicate when a cell reference is invalid. For example, If a formula contains a reference to a cell, and then the row or column containing that cell is deleted, a #REF! error results. If a worksheet does not support 20,001 columns, $\text{OFFSET}(\text{A1}, 0, 20000)$ will result in a #REF! error.
#VALUE!	Intended to indicate when an incompatible type argument is passed to a function, or an incompatible type operand is used with an operator. For example, In the case of a function argument, a number was expected, but text was provided. In the case of $1 + "ABC"$ , the binary addition operator is not defined for text.

### 3.16.9 Dates and Times

- Each unique instant in SpreadsheetML time is represented as a distinct non-negative numeric *serial value*, which is made up of an integer date component and a fractional time component. As dates and times are numeric values, they can take part in arithmetic operations.
- Numerous functions take as arguments one or more serial values or strings representing dates and/or times. Functions that care only about the date shall ignore any time information that is provided. Functions that care only about the time shall ignore any date information that is provided.

#### 3.16.9.1 Date Representation

Going forward in time, the date component of a serial value increases by 1 each day.

There are two different bases for serial values:

- In the *1900 date base system*, the lower limit is January 1, 1900, which has serial value 1. The upper-limit is December 31, 9999, which has serial value 2,958,465.
- In the *1904 date base system*, the lower limit is January 1, 1904, which has serial value 0. The upper-limit is December 31, 9999, which has serial value 2,957,003.

As to which date base system an implementation uses by default or whether it allows its users to switch between date base systems, is unspecified.

For the 1900 date base system:

1 DATEVALUE("01-Jan-1900") results in the serial value 1.0000000...  
 2 DATEVALUE("03-Feb-1910") results in the serial value 3687.0000000...  
 3 DATEVALUE("01-Feb-2006") results in the serial value 38749.0000000...  
 4 DATEVALUE("31-Dec-9999") results in the serial value 2958465.0000000...

5 For the 1904 date base system:

6  
 7 DATEVALUE("01-Jan-1904") results in the serial value 0.0000000...  
 8 DATEVALUE("03-Feb-1910") results in the serial value 2225.0000000...  
 9 DATEVALUE("01-Feb-2006") results in the serial value 37287.0000000...  
 10 DATEVALUE("31-Dec-9999") results in the serial value 2957003.0000000...

### 11 3.16.9.2 Time Representation

12 The time component of a serial value ranges in value from 0–0.9999999, and represents times from 0:00:00  
 13 (12:00:00 AM) to 23:59:59 (11:59:59 P.M.), respectively.

14 Going forward in time, the time component of a serial value increases by 1/86,400 each second. (As such, the  
 15 time 12:00 has a serial value time component of 0.5.)

16 TIMEVALUE("00:00:00") results in the serial value 0.0000000...  
 17 TIMEVALUE("00:00:01") results in the serial value 0.0000115...  
 18 TIMEVALUE("10:05:54") results in the serial value 0.4207639...  
 19 TIMEVALUE("12:00:00") results in the serial value 0.5000000...  
 20 TIMEVALUE("23:59:59") results in the serial value 0.9999884...

### 21 3.16.9.3 Combined Date and Time Representation

22 Any date component can be added to any time component to produce a serial value for that date/time  
 23 combination.

24 For the 1900 date base system:

25  
 26 DATE(1910,2,3)+TIME(10,5,54) results in the serial value 3687.4207639...  
 27 DATE(1900,1,1)+TIME(12,0,0) results in the serial value 1.5000000...  
 28 DATE(9999,12,31)+TIME(23,59,59) results in the serial value 2958465.9999884...

29 For the 1904 date base system:

30  
 31 DATE(1910,2,3)+TIME(10,5,54) results in the serial value 2225.4207639...  
 32 DATE(1904,1,1)+TIME(12,0,0) results in the serial value 0.5000000...  
 33 DATE(9999,12,31)+TIME(23,59,59) results in the serial value 2957003.9999884...

### 3.16.10 XML Representation

A formula is represented in a worksheet's XML by an `f` element that contains the text of the formula, and a `v` element that contains the text version of the last computed value for that formula. This pair of elements is inside a `c` element, which is, in turn, is inside a `row` element. Consider the scalar formula  $\text{SQRT}(\text{C2}^2+\text{D2}^2)$ , where C2 refers to a cell containing the number 12.5, and D2 refers to a cell containing the number 9.6. The corresponding XML might be as follows:

```

7   <row r="2" spans="2:4">
8     <c r="B2" s="40">
9       <f>SQRT(C2^2+D2^2)</f>
10      <v>15.761027885261798</v>
11    </c>
12    <c r="C2" s="0">
13      <v>12.5</v>
14    </c>
15    <c r="D2" s="0">
16      <v>9.6</v>
17    </c>
18  </row>
```

In the scalar formula `CONCATENATE("The total is ",C7," units")`, C7 refers to a cell containing the number 23. The corresponding XML might be as follows:

```

21   <row r="7" spans="2:4" ht="285">
22     <c r="B7" s="4" t="str">
23       <f>CONCATENATE("The total is ",C7," units")</f>
24       <v>The total is 23 units</v>
25     </c>
26     <c r="C7" s="0">
27       <v>23</v>
28     </c>
29   </row>
```

As the function `CONCATENATE` returns a string, the value for the cell's `t` attribute is `str`.

# 4. Introduction to PresentationML

2     **This clause is informative.**

3     This clause contains a detailed introduction to the structure of a PresentationML document.

4     The PresentationML file format can be broken down into the following subjects:

- 5       • Presentation
- 6       • Slides
- 7       • Slide Content
- 8       • Animation

9     There are other schemas—most notably DrawingML—that make up a sizeable chunk of the PresentationML  
10    file format. These schemas are addressed separately in §5.

## 4.1 Basics

### 4.1.1 Introduction

13    This subclause provides a high-level overview of the content described in the following schemas: pml-  
14    baseTypes.xsd, pml-presentation.xsd, pml-presentationProperties.xsd, and pml-viewProperties.xsd.

15    The PresentationML file format can be broken down into the following subjects:

- 16       • Presentation
- 17       • Slides
- 18       • Slide Content
- 19       • Animation

20    The best way to understand the content in each of these subjects is to cover them in that particular order.

21    The eight schemas that collectively represent the PresentationML file format can be grouped by subject as  
22    follows:

<b>Presentation</b>	<b>Slide</b>	<b>Slide content</b>	<b>Animation</b>
pml-baseTypes.xsd	pml-slide.xsd	pml-embedding.xsd	pml-animationInfo.xsd
pml-presentation.xsd		pml-comments.xsd	
pml-presentationProperties.xsd			
pml-viewProperties.xsd			

1

2 There are other schemas—most notably DrawingML—that make up a sizeable chunk of the PresentationML  
 3 file format. These schemas are addressed separately.

4 This subclause introduces the first subject, “Presentation”. Other subclauses build on this foundation.

## 5 **4.1.2 Basic Utilities**

6 The schema pml-baseTypes.xsd contains a set of complex types and simple types that are used by other  
 7 schemas. The types, or utilities, are used in a variety of cases. Their single implementation provides for rapid  
 8 and less error-prone changes throughout an implementation.

9 To provide some insight into the type of information that is being repurposed, here is a sample of these  
 10 utilities:

- 11 • Empty Element
- 12 • Name
- 13 • Direction
- 14 • Index and Index Range
- 15 • Slide Show ID
- 16 • Slide List Choice
- 17 • Slide Relationship
- 18 • Customer Data
- 19 • Future Extensibility

20 Each of these is discussed in the following subclauses.

### 21 **4.1.2.1 Empty Element**

22 Sometimes, the simple presence of an element is sufficient to convey meaning. That is, in some cases, you do  
 23 not necessarily need information to be a Boolean, an integer, or complex type.

24 A simple example is the Show Type element group. In this case, a slide show can be one of three types:  
 25 present, browse, or kiosk. The schema for this element group is as follows:

```
26 <xsd:group name="EG_ShowType">
27   <xsd:choice>
28     <xsd:element name="present" type="CT_Empty">
29     </xsd:element>
30     <xsd:element name="browse" type="CT_ShowInfoBrowse">
31     </xsd:element>
32     <xsd:element name="kiosk" type="CT_Empty">
33     </xsd:element>
34   </xsd:choice>
35 </xsd:group>
```

1    4.1.2.2        Name

2    Many constructs within a presentation have names associated with them. In some cases, the names are  
 3    machine-generated, such as shape names (e.g., rectangle1), while others are user-defined, such as slide shows  
 4    (e.g., customer-ready).

5    In one implementation the name simple type is simply an xsd:string. The intent is to restrict this to the  
 6    appropriate pattern allowed for named constructs. The tentative restriction pattern is:

7    [ \t]\*[^ \t].\*

8    4.1.2.3        Direction

9    This multi-purpose simple type is used to convey horizontal versus vertical direction of a variety of types. Such  
 10   usage can be found in the definition of slide transitions and various shape effects.

11   4.1.2.4        Index and Index Range

12   These two utilities are generally used to denote a contiguous set of items within a list. The classic example of  
 13   usage would be the selection of a set of slides to print.

14   From a schema-perspective, there is no way to enforce that the start index be equal to or less than the end  
 15   index.

16   4.1.2.5        Slide Show ID

17   This defines the ID for a slide show (also called a custom show). Because slide shows can be named, and that  
 18   name can change, an implementation needs a method of referring to a slide show that can withstand name  
 19   changes made by the user. In many cases, for example, with a slide, we can leverage the fact that each slide  
 20   has a part within the package, in which case we can use the relationship ID. However, since there is no part for  
 21   each slide show, we are forced to generate an unsigned integer for each slide show and use that.

22   There is nothing in the schema that prevents two or more slide shows from having the same ID.

23   4.1.2.6        Slide List Choice

24   There are many cases in which a user needs to specify a set of slides for an operation. The canonical example  
 25   is what slides to include in your slide show. Because this operation is frequently required in the file format,  
 26   one implementation has provided a utility to facilitate this:

```
27   <xsd:group name="EG_SlideListChoice">
  28     <xsd:choice>
  29       <xsd:element name="sldAll" type="CT_Empty" />
  30       <xsd:element name="sldRg" type="CT_IndexRange" />
  31       <xsd:element name="custShow" type="CT_CustomShowId" />
  32     </xsd:choice>
  33   </xsd:group>
```

- 1 As the schema above declares, when selecting a set of slides, the user can select all of the slides, a slide range  
 2 (by declaring a pair of start and end indices) or a particular custom show.

3 **4.1.2.7 Slide Relationship**

- 4 As described in the Slide Show ID paragraphs above, there are many situations where the format needs to  
 5 store an ordered list of slides, and does so by storing their slide IDs. This is implemented using two types: a list  
 6 entry complex type and a list complex type:

```

7   <xsd:complexType name="CT_SlideRelationshipListEntry">
8     <xsd:attribute ref="r:id" use="required"/>
9   </xsd:complexType>
10  <xsd:complexType name="CT_SlideRelationshipList">
11    <xsd:sequence>
12      <xsd:element name="sld" type="CT_SlideRelationshipListEntry"
13        minOccurs="0" maxOccurs="unbounded"/>
14    </xsd:sequence>
15  </xsd:complexType>
```

16 **4.1.2.8 Customer Data**

- 17 There is a set of utilities that facilitate the storage of customer XML data within the file format. Although a  
 18 topic for a separate paper, essentially, this functionality comes down to the ability to store customer-defined  
 19 XML in the file format in a way that it can be easily queried, modified and/or surfaced in the presentation.  
 20 Suffice it to say, the data is stored in a separate part within the package, and hence the utility pairs the object  
 21 using it with the part within the package.

22 **4.1.2.9 Future Extensibility**

- 23 There is functionality that provides the ability to extend a subset of objects within the file format for inclusion  
 24 of additional data over the lifetime of the file format. The utilities provide both the ability to add an  
 25 alternative representation (e.g., provide a raster image in addition to the XML data for a diagram) as well as  
 26 additional properties to the objects.

27 **4.1.3 The Presentation Object**

- 28 The schema pml-presentation.xsd defines the content of the principal or start part for a PresentationML  
 29 document. This content includes both structural and presentation-level data for the presentation.

- 30 Astute readers will quickly identify an apparent duplication of presentation-level data, as there is also a  
 31 separate schema file, pml-presentationProperties.xsd, which contains presentation-level data. That being said,  
 32 there is actually no duplication. Rather, the differentiation of what presentation-level data goes into which  
 33 part is based on two user scenarios: document signatures and document sanitization.

- 34 In a document signature scenario, assume a user digitally signs a presentation. There exist two types of data  
 35 within the presentation package: data which changes the “content” of the presentation and data which is  
 36 intended to configure an editor or the behavior of an editor. In the first case, any modification to data which

1 changes the “content” of the presentation must invalidate the signature; in the second case, any modification  
2 to that data should not invalidate the signature.

3 A classic example of this scenario deals with Kinsoku information and the publish path in the HTML settings. If  
4 the user changes the Kinsoku information in a file, the file will look (and potentially mean) something different.  
5 This is in contrast to a user setting a new HTML publish path for their particular computer.

6 In a document sanitization scenario, users want to remove all non-necessary information from the file. A  
7 typical usage case would be posting a presentation to a company’s Internet site. In this case, you don’t want  
8 certain configuration information publicly available. The ideal manner of removal would be to remove an  
9 entire part from the presentation package as opposed to editing a part from a package.

10 Going back to our Kinsoku and HTML publish path example above, the Kinsoku information needs to remain  
11 with the file. The HTML publish path could give away internal information about web servers that could be  
12 used to facilitate an attack or, more likely, simply provide information about the author to the public (e.g., the  
13 path c:\documents and settings\shawnv\webpages strongly implies that “shawnv” published this document).

14 Going back to the original question—what presentation-level data goes in which part—we see that data that  
15 will not invalidate a digital signature or data that should be removed during a sanitization pass should be  
16 stored in the part associated with the pml-presentationProperties.xsd schema and other presentation-level  
17 data should be stored in the part associated with the pml-presentation.xsd schema.

18 In addition to structural and presentation-level data defined by this schema, there are also definitions for  
19 handling customer data and future extensibility. Again, both of these will be addressed in additional papers.

#### 20     4.1.3.1       Structural Information

21 From a structural information perspective, there are two sets of data defined in this schema: core lists and  
22 sizes.

23 The schema first defines a number of lists that serve as the foundation for most objects in the presentation.  
24 These lists are as follows:

- 25           • Slide IDs
- 26           • Slide Masters
- 27           • Notes Masters
- 28           • Handout Masters
- 29           • Custom Shows

30 It is essential that the reader fully understand the implementation of usage of these lists as they are the  
31 foundation for almost all solutions that operate—open, interrogate, modify, write—against the  
32 PresentationML file format.

33 As mentioned above, the lists are defined as a part of list entry and list complex types. The slide master list is  
34 defined as follows:

```

1   <xsd:complexType name="CT_SlideMasterIdListEntry">
2     <xsd:attribute ref="r:id" use="required" />
3   </xsd:complexType>
4   <xsd:complexType name="CT_SlideMasterIdList">
5     <xsd:sequence>
6       <xsd:element name="sldMasterId" type="CT_SlideMasterIdListEntry"
7         minOccurs="0" maxOccurs="unbounded" />
8     </xsd:sequence>
9   </xsd:complexType>
```

- 10 Although not complex or difficult to understand, the lists are called out because they are vital to any solution.
- 11 The next pieces of structural information are the sizes for the slides and the notes slides. By storing this  
12 information at the presentation level, the implication is that all slides (or all notes slides) in a presentation  
13 have the same size. This further implies that all slides in a presentation share the same orientation (i.e., they  
14 are all landscape-oriented or all portrait-oriented).

#### 15 4.1.3.2 Presentation-Level Properties

- 16 The presentation-level properties defined in this schema can be grouped into the following groupings:

- 17 • Text-Related
- 18 • Save-Related
- 19 • Editor-Related
- 20 • Content-Related

- 21 A description for each property within each group follows.

##### 22 4.1.3.2.1 Text-Related Properties

- 23 The first property stores information related to the Kinsoku settings. Kinsoku settings define the list of  
24 characters that are not allowed to start or end a line of text for a given East Asian language.

- 25 The schema definition of the Kinsoku settings is relatively straightforward: identify the language, the set of  
26 invalid start characters, and the set of invalid end characters:

```

27   <xsd:complexType name="CT_Kinsoku">
28     <xsd:attribute name="lang" type="xsd:string" use="optional">
29   </xsd:attribute>
30     <xsd:attribute name="INVALStChars" type="xsd:string" use="required">
31   </xsd:attribute>
32     <xsd:attribute name="INVALEndChars" type="xsd:string" use="required">
33   </xsd:attribute>
34 </xsd:complexType>
```

- 35 The second property stores a flag to use strict characters for starting and ending a line of Japanese text.  
36 Naturally, this is a simple Boolean attribute:

```

1      <xsd:attribute name="strictFirstAndLastChars"
2          type="xsd:boolean" use="optional" default="true"/>
3
4      The final text-related property stores information related to any fonts that are embedded in the presentation.
5      To do this, we need to store a list of embedded fonts that reference each part that stores font data (generally,
6      there is a one-font-to-one-part mapping, although this is not a strict rule). This information is defined using
7      three complex types:
8
9          <xsd:complexType name="CT_EMBEDDEDFONTLIST">
10         <xsd:sequence>
11             <xsd:element name="embeddedFont" type="CT_EMBEDDEDFONTLISTENTRY"
12                 minOccurs="0" maxOccurs="unbounded" />
13         </xsd:sequence>
14     </xsd:complexType>
15     <xsd:complexType name="CT_EMBEDDEDFONTLISTENTRY">
16         <xsd:sequence>
17             <xsd:element name="font" type="a:CT_TextFont" minOccurs="1"
18                 maxOccurs="1" />
19             <xsd:element name="regular" type="CT_EMBEDDEDFONTDATAID"
20                 minOccurs="0" maxOccurs="1"/>
21             <xsd:element name="bold" type="CT_EMBEDDEDFONTDATAID"
22                 minOccurs="0" maxOccurs="1"/>
23             <xsd:element name="italic" type="CT_EMBEDDEDFONTDATAID"
24                 minOccurs="0" maxOccurs="1"/>
25             <xsd:element name="boldItalic" type="CT_EMBEDDEDFONTDATAID"
26                 minOccurs="0" maxOccurs="1" />
27         </xsd:sequence>
28     </xsd:complexType>
29     <xsd:complexType name="CT_EMBEDDEDFONTDATAID" >
30         <xsd:attribute ref="r:id" use="required"/>
31     </xsd:complexType>

```

#### 30      4.1.3.2.2        Save-Related Properties

31      There is a set of properties that indicate to the editor what should be saved as part of the presentation.

32      The first such property controls the inclusion of Personally Identifiable Information (“PII”). PII is any  
33      information that can be used to identify the author or contributor to a presentation. And while there are cases  
34      where this information is exposed visually to the user (e.g., author name in a comment shape), there are other  
35      cases where the information is not immediately evident to the user (e.g., the document author name in the list  
36      of document properties).

37      An implementation can provide a mechanism by which the author of a presentation can configure a file to  
38      always remove any PII that might otherwise be normally included during a regular save operation. While not a  
39      guarantee that no PII is stored in the file (e.g., consider a shape with my name in it—in some cases it describes

1 content in the file [my position in my group's organization chart] whereas in others it is an editorial directive  
2 ["check with ShawnV on this point"]. Given this ambiguity, we cannot solve all cases of this. As a result, this is  
3 more a convenience feature than a privacy management feature.

4 The second set of save-related properties has two groupings of properties. The first controls whether or not  
5 fonts will be embedded into the package representing the presentation. The second, enabled by setting the  
6 first, allows an implementation to optimize such font embedding to keep the size minimal, at the cost of future  
7 editing on other machines.

```
8 <xsd:attribute name="embedTrueTypeFonts" type="xsd:boolean"  
9   use="optional" default="false"/>  
10 <xsd:attribute name="saveSubsetFonts" type="xsd:boolean"  
11   use="optional" default="false"/>
```

12 The user scenario behind these properties is as follows. Assume you are putting together a presentation to  
13 distribute to external customers. You happen to use an East Asian font with an on-disk file size of around  
14 5 megabytes.

15 Assuming that this font is not a standard font that is widely distributed, not including this font will cause font  
16 substitution when the presentation is opened on machines that don't have a copy of the font. In any case, this  
17 can radically change the visual appearance of the presentation; in some cases, it can render the presentation  
18 unreadable.

19 Because you cannot afford the presentation to be unreadable or to look unprofessional, you decide to embed  
20 the font. By default, the implementation will set embedTrueTypeFonts to true and embed the entire  
21 5 megabyte font file in the presentation package. This will clearly bloat the file, but will ensure that anyone  
22 viewing or editing this file will have the same font experience as you originally had (subject to licensing  
23 restrictions, of course).

24 Since you are distributing the presentation, and your primary purpose is for people to view the presentation,  
25 you can reduce the amount of font data embedded in the presentation package. By setting the second  
26 property (saveSubsetFonts) to true, only those characters in the font that were actually used to create the  
27 presentation are saved. This yields less font data stored in the file at the cost of not being able to use unused  
28 characters in future edits of the presentation on different machines.

29 The third property related to saving, controls whether or not an implementation can automatically compress  
30 pictures contained in the presentation. This is particularly important given the proliferation of digital cameras  
31 and scanners and the increasing importance of small files (e.g., to save network bandwidth, reduce storage  
32 required for mail and file servers, etc.).

33 The final property in this set specifies a password that is required to enable editing of the file using the  
34 implementation. Because this is a convenience feature intended to prevent accidental changes to information,  
35 it is stored in clear text as an xsd:string.

1 By storing this information in the file, the implementation will prompt the user for this password in order to  
2 open the file read/write; if the user does not provide the correct modify password, the implementation will  
3 open the file read-only.

4 **4.1.3.2.3 Editor-Related Properties**

5 The presentation file itself contains data that provide configuration information for the implementation's  
6 editor.

7 For example, the presentation can define a set of smart tags for use while editing the particular presentation.  
8 Because Smart Tags are stored in a separate part, the presentation object contains the relationship ID of the  
9 Smart Tag part.

10 In a fully functioning OLE server, PresentationML objects can be embedded into OLE containers, during which  
11 time a customer can set a zoom scale. This is stored in the file as a percentage called serverZoom.

12 An internationalized application might support configuring the editor to respect different screen orientations.  
13 For example, in regions of the world where Complex Scripts are in use, it is customary to orient the screen  
14 right-to-left. As such, a presentation can request the editor reconfigure itself for such usage scenarios.

15 Finally, due to changes in the file format and functionality (e.g., graphics and text engines), PresentationML  
16 introduces some end-user complexity when working collaboratively with other customers using older versions.  
17 To help remedy this, an implementation might support a Compatibility Mode, which restricts the functionality  
18 exposed by the editor to optimize the output for the best cross-version collaboration story possible. As a  
19 result, each presentation needs to opt-into this mode.

20 **4.1.3.2.4 Content-Related Properties**

21 This set of properties is related to the actual content in the presentation.

22 End-users can define the starting slide number for numbered slides in each presentation. While it typically  
23 starts at one, the user can select any positive number to begin slide numbering. The primary user scenario is  
24 when compiling a mega-presentation that is a collection of multiple presentations. A secondary user scenario  
25 is when including a presentation in the middle of or at the end of a printed document where you want the  
26 slide/page numbers to continue.

27 Another content-related property controls whether or not header/footer placeholders are to be shown on title  
28 slides. In many cases users will use special shapes called header and footer placeholder that contain built-in  
29 field codes that control the display of various sorts of information like the date/time and slide number.

30 In most cases, users like to keep their title slides as simple as possible (much like in the printed world where  
31 you want your first page to be clean and streamlined) and hence do not want data like date/times and slide  
32 numbers to show up on such slides. This attribute defines this presentation-wide.

33 The final property relates to creating photo albums. The implementation has a feature that allows the user to  
34 generate automatically a presentation based on a set of pictures. During this process, the user can select from  
35 a variety of settings, including, but not limited to, what pictures to include, the layout of the pictures on the

1 slides (e.g., one picture per slide, two pictures per slide, etc.), what type of frame shape to use, etc. All of this  
2 information is stored in the presentation for future photo album creation.

### 3 **4.1.4 Presentation Properties**

4 Those properties that apply to the presentation as a whole, and that are likely to be removed during document  
5 sanitization, or are not going to invalidate a digital signature, are defined in pml-presentationProperties.xsd.  
6 These properties can be grouped into three primary groupings.

- 7 • HTML Publish Properties
- 8 • Print Properties
- 9 • Slide Show Properties
- 10 • View Properties

11 In addition to this grouping, there are properties that define a Most Recently Used (“MRU”) list of colors as  
12 well as providing for future extensibility. (The MRU will be discussed in a DrawingML paper and the  
13 extensibility will be discussed in a similar paper.)

#### 14 **4.1.4.1 HTML Publish Properties**

15 An implementation must have the ability to save (and publish) a presentation to a web-friendly format like  
16 HTML or MHTML. Various parameters are used to configure the application for saving such formats as well as  
17 to control what content gets generated. The parameters that configure the application are the HTML Publish  
18 properties whereas the content properties are the Web Properties.

19 The HTML Publish properties provide the author with the ability to control what content gets displayed in the  
20 browser when the resulting file—either HTML or MHTML—is viewed using that type of an application. For  
21 example, the speaker notes can either be displayed in the frameset or can be hidden from view. This is  
22 particularly useful when a speaker’s notes are not necessarily in a customer-ready format. It’s useful but not  
23 necessarily secure.

24 The author can also specify the title to be displayed in the browser. Although this defaults to the actual file  
25 name, or if that is missing, to the content of the first slide’s title placeholder, it can be overridden by the  
26 author.

27 Finally the author can specify a publish path to use when saving this file in this format. This is particularly  
28 useful for two reasons.

29 First, because there is a transformation happening, it sometimes takes a few iterations of publishing to get the  
30 browser-based experience to be exactly what you want. A classic example of this is the differing animation  
31 capabilities between the implementation and certain browsers: it is important to verify that the change in  
32 animation behavior continues to work after publishing; if you are not satisfied with the experience, sometimes  
33 you need to change the animation in the implementation and republish.

34 The second reason storing the path is useful is that web server paths can be cumbersome and are often not on  
35 the tip of each user’s tongue. This allows the user to specify the path once and then publish using the same

1 location without having to re-specify it. Naturally, being stored in the file format, this allows this data to  
 2 persist across session.

3 Indirectly, the HTML Publish properties can prime the Web Properties by defining a target web browser  
 4 generation (i.e., third, fourth or third and fourth). This is done by setting the appropriate  
 5 ST\_HtmlPublishWebBrowserSupport attribute:

```
6   <xsd:complexType name="CT_HtmlPublishProperties">
7     <xsd:sequence>
8       <xsd:group ref="EG_SlideListChoice" minOccurs="1" maxOccurs="1">
9         </xsd:group>
10        </xsd:sequence>
11        <xsd:attribute name="showSpeakerNotes" type="xsd:boolean">
12          use="optional" default="true" />
13        <xsd:attribute name="pubBrowser">
14          type="ST_HtmlPublishWebBrowserSupport"
15          use="optional" default="v3v4" />
16        <xsd:attribute name="title" type="xsd:string" use="optional">
17          default="">
18        </xsd:attribute>
19        <xsd:attribute ref="r:id" use="required">
20        </xsd:attribute>
21      </xsd:complexType>
```

22 By providing a target generation, the Web Properties will be set to a predefined package defined for the  
 23 specified browser generation. Naturally, the user can override the individual Web Property settings.

#### 24 4.1.4.1.1 Web Properties

25 As mentioned in the previous subclause, these properties configure the output of the presentation when saved  
 26 using the HTML or MHTML formats. In this case, a number of parameters can be controlled.

27 In all multi-slide cases where the presentation is saved using one of these formats, the implementation will  
 28 create a frameset to bring the various parts of a presentation—the slide content, the speaker notes and the  
 29 outline—together as well as provide for simple navigation. The color of the HTML frames, the background  
 30 used and the user interface controls can be controlled to leverage browser settings, use high contrast, etc.

31 The author can also control how much interactivity will be exposed in the resulting output. For example, the  
 32 user may elect to disable slide animations and transitions and opt for a more static presentation. Similarly, the  
 33 author may elect to disable certain scripting features like the ability to resize dynamically the output to match  
 34 the size of the browser window.

35 Somewhat related to this is the ability to specify the target screen size which is especially important when  
 36 targeting the earlier browser generations or user environments where features like JavaScript are disabled.

1 For an internationalized implementation, there is the ability to control the encoding of text used in the  
2 generation of the HTML or MHTML output.

3 Finally there are a set of parameters that configure the on-disk storage of the resulting output. For example, if  
4 the customer knows something about the machine configurations of her audience, she can opt to use better  
5 raster graphic formats like PNG that support alpha transparency or elect to include Vector Markup Language  
6 ("VML") representations only for vector images.

7 The customer can also provide some indication as to how the output will be used. If the customer knows that  
8 the output will be used like regular files (perhaps passed around on CDs or moved between file shares) the  
9 user may elect to store the files in a folder to ensure that a straggling file is not lost; if, however, the target  
10 scenario is to put the files on a web server, the user can skip the folder and save the individual files in a flat  
11 directory. Similarly, if the customer knows that they are using a web server that only handles "8.3" file names,  
12 they can configure the implementation to generate files using names that are "8.3" compliant, as opposed to  
13 using long file names that may otherwise cause such web servers problems.

#### 14 4.1.4.2 Print Options Properties

15 There is also a set of properties that control the default print behavior for a presentation. The inclusion of this  
16 information in the file format simply primes the Print dialog when this presentation is used. It does not force  
17 options nor does it represent the last-used set of print options for a presentation.

18 Using these properties, the author can control the type of output printed. For example, in some cases, authors  
19 need to print their slides (one slide per printed page) while in other cases, they want to provide printed  
20 handouts for the audience on which to take their own notes (handout pages that can contain anywhere from  
21 three to nine slides per printed page, as well as option lines for note taking). In other cases, the author would  
22 like to print out notes where each printed page has one slide (anchored at the top) and a text box (anchored at  
23 the bottom) with the speaker notes included or simply print the textual outline of the presentation.

24 The author can also control whether or not hidden slides are included in the printed output, as well as whether  
25 or not the output is sent to the printer in color, in grayscale, or in pure black and white.

26 There is also a set of properties that the author can set that determine if slides are framed on the printed  
27 page, if the slides are scaled up to the printed page (e.g., consider non 4x3 aspect ratio slides), etc.

#### 28 4.1.4.3 Slide Show Properties

29 This set of presentation-level properties controls the default slide show.

30 Among the parameters that can be controlled is one that defines the type of slide show. Generally, the classic  
31 slide show is characterized by a presenter presenting the presentation to an audience. The presenter controls  
32 the flow of the presentation, etc. This is referred to as a "present" slide show. In some cases, however, the  
33 presentation is distributed and individuals walk themselves through the slide show. This is referred to as a  
34 "browse" slide show. Finally, there are cases where a slide show is prepackaged and used as a kiosk; naturally,  
35 it is referred to as a "kiosk" slide show.

1 Furthermore, the customer can control which slides are to be included in the slide show, what color the pen  
 2 should be, etc.  
 3 Finally, the customer can control various interactivity settings that are to be used for the slide show. This  
 4 provides the customer the ability to configure their slide show outside the typical settings for a particular slide  
 5 show type. For example, the user may create a slide show that has a pre-configured animation built with  
 6 timings (i.e., the time between particular builds or the time between slide transitions), even though she is  
 7 going to be presenting the content to an audience.

8 **4.1.4.4 View Properties**

9 The schema pml-viewProperties.xsd defines the properties on all of the views found in the implementation.  
 10 MS's implementation currently supports the following views:

- 11 • Slide View
- 12 • Slide Master View
- 13 • Notes View
- 14 • Handout View
- 15 • Notes Master View
- 16 • Outline View
- 17 • Slide Sorter View

18 Additionally, the default view, Normal View, is a composite view that pulls from three multiple view property  
 19 sets.

20 In general, there is a significant amount of commonality among views. For example, each view contains four  
 21 common components:

<b>Scale</b>	<b>The zoom scale for the view</b>
Origin	The origin of the view
Variable Scaling	A special zoom scale that configures the application to fit the content of the view into whatever view size is provided
Draft Mode	Controls whether or not a view is in draft mode which is a mode designed to provide the fastest editing/redraw possible by dropping properties like font face, certain colors, pictures, etc.

22

23 For those views based on a slide (e.g., slide master view) there are additional common components:

<b>Guide List</b>	<b>Represents the list of drawing guides for this view</b>
Guide Properties	Represents guide properties like direction and position of each guide in the view

Guide List	Represents the list of drawing guides for this view
Guide Settings	Determines if guides should be shown for this view
Snap Settings	Determines if shapes should be snapped to the grid and/or snapped to other shapes for this view

## 1    4.2     Slides, Masters, Layouts, and Placeholders

### 2    4.2.1     Introduction

3    This subclause provides a high-level overview of the content described in the pml-slide.xsd schema.

4    The important aspects of the PresentationML Slides file format are introduced in the following order.

- 5        •    Masters
- 6        •    Presentation Slide
- 7        •    Slide Notes
- 8        •    Slide Layouts

9    This subclause provides a structured introduction to the slides portion of the PresentationML file format. Other  
10    subclauses build on this foundation and explain more about topics such as animation, comments, and the  
11    presentation object.

### 12    4.2.2     Masters

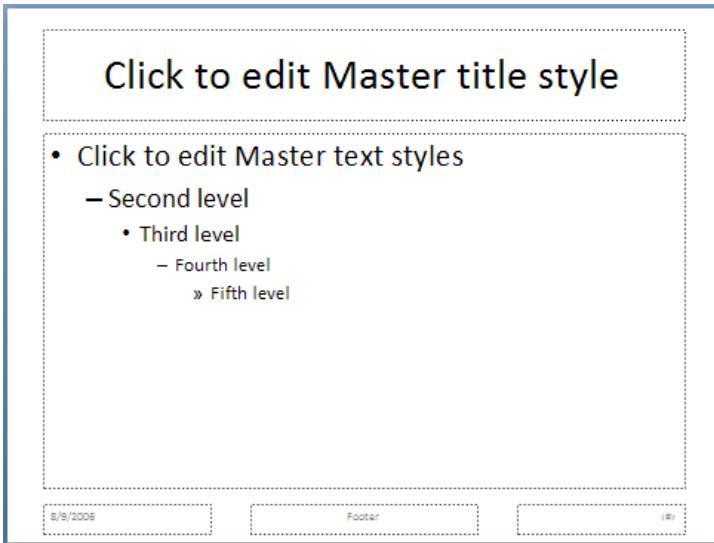
13    For slides the notion of hierarchy and inheritance applies. A master represents a common layout for the page  
14    type in question. For instance, if a slide master had a background set to a gradient fill then all slides referencing  
15    to that slide master would have the same background. In addition to setting common attributes of the slides  
16    such as background and styling information, the slide master also provides numerous layouts within itself in  
17    order to make a presentation that both follows a layout theme and incorporates a high level of variety. The  
18    variety is supported through slide layouts which will be discussed in a later section.

#### 19    4.2.2.1     Slide Master

20    A *slide master* is a master that is tied specifically to presentation slides. The presentation slides are those that  
21    are shown during a presentation. These will be discussed in more detail in a later section on the Presentation  
22    Slide. Within a slide master are some common structural elements that should be understood, namely:

- 23        •    *Common Data* - Common properties that will be inherited by the other slides as well as layout  
24              information for presentation slides based on the master slide.
- 25        •    *Header and Footer* - Header and footer properties for the presentation slides to inherit.
- 26        •    *Color Map* - Color Mapping for the presentation slides to inherit.
- 27        •    *Text Styles* - Text Styling information to be used within each placeholder on a presentation slide.
- 28        •    *Slide Layout List* - A list of slide layouts that provide the variety needed within any presentation. These  
29              are applied to a presentation slide which will inherit both the layout of the slide layout in addition to  
30              the slide design of the slide master.
- 31        •    *Timing Information* - Common timing properties used for animation, controls, etc.

- 1 • *Transition Information* - Slide transitioning information to be inherited by each presentation slide.

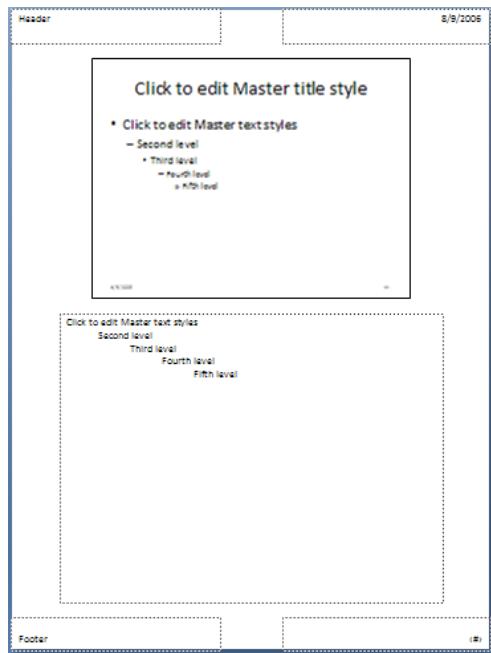


- 2  
3 Slides inheriting information from a slide master do have the ability to specify properties that override those  
4 specified in the slide master.

5 **4.2.2.2 Notes Master**

6 A *notes master* is a master that specifies properties for slide notes pages. The notes page associated with a  
7 presentation slide stores a thumbnail of the presentation slide as well as the presenter's notes about the slide.  
8 These will be discussed in more detail in a later section. Within a notes master the important common  
9 structural elements are:

- 10 • *Common Data* - Common properties that will be inherited by other notes pages as well as the layout  
11 information for notes pages based on this master slide. The notes master serves as the pattern for all  
12 notes pages.  
13 • *Color Map* - Color Mapping for the notes pages to inherit.

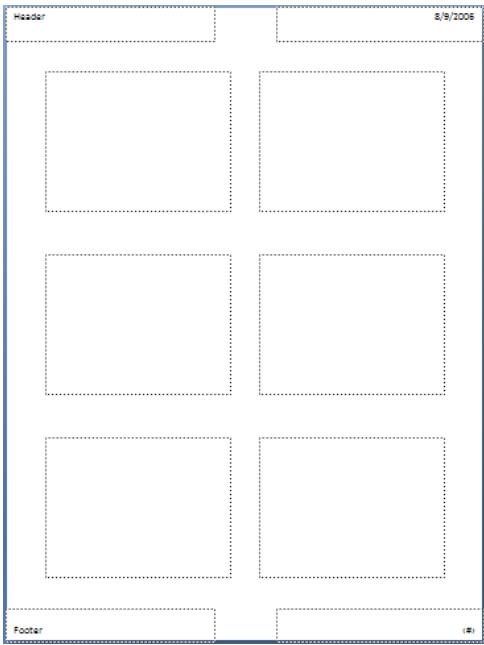


2 Notes pages inheriting information from a notes master do have the ability to specify properties that override  
3 those specified in the notes master.

#### 4 4.2.2.3 Handout Master

5 A handout master determines the layout for all handout pages. The handout pages consist of a place to store a  
6 thumbnail of each slide with additional elements such as header, footer or graphical information. These will be  
7 discussed in more detail in a later section. Within a handout master are some common structural elements  
8 that should be understood, namely the following.

- 9     • *Common Slide Data* - Common properties and layout information that will be used by all handout  
10        pages. The handout master represents how each handout page will look.
- 11     • *Header and Footer* - Header and footer properties for all handout pages.
- 12     • *Color Map* - Color Mapping for all handout pages.



1

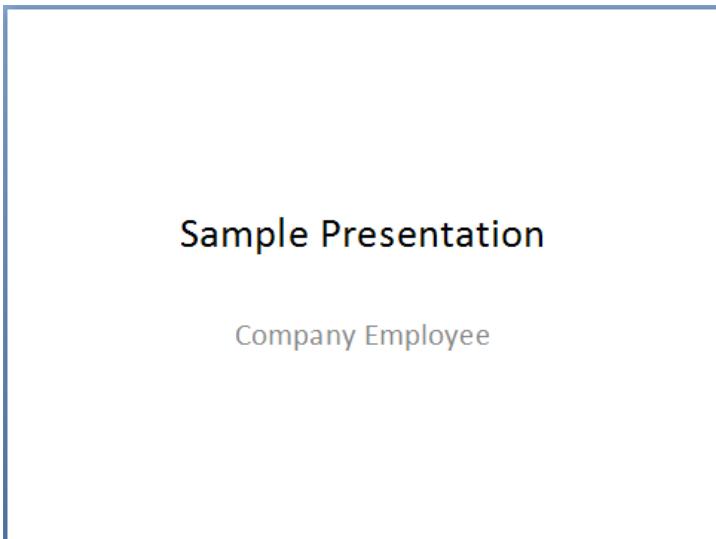
2

### 4.2.3 Presentation Slide

A presentation slide is a slide that inherits slide properties from the corresponding slide master and layout information from the corresponding slide layout. Each presentation slide has the ability to override any of this information that it chooses by specifying local attribute values within the presentation slide. Much like the master slide, the presentation slide contains some common structural elements, namely the following.

- *Common Slide Data* - Common properties and layout information for this presentation slide. Properties listed here that conflict with existing elements specified in the slide master will override those specified in the slide master.
- *Color Map Override* - Color Mapping that will override the inherited color mapping for this presentation slide.
- *Timing Information* - Common timing properties used for animation, controls, etc.
- *Transition Information* - Slide transitioning information for this presentation slide.

The above list defines the areas that can be used to override inherited components from the master slide and the layout slide. That is, these can be specifically defined on a per-slide basis via the above elements.

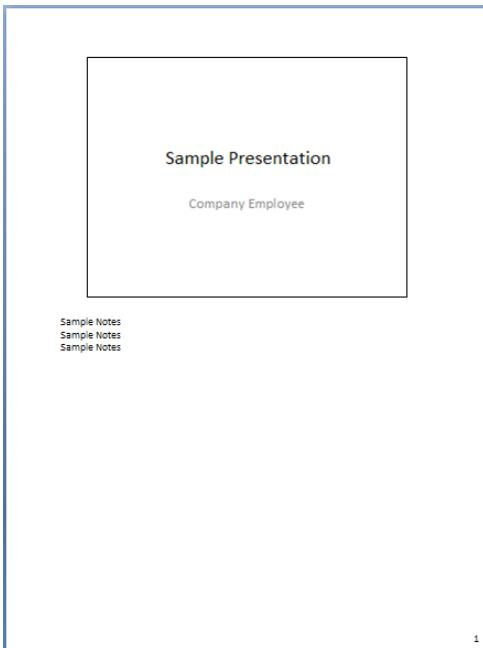


#### 2    4.2.4    Notes Page

3    A notes page inherits slide properties from the corresponding notes master. The initial layout for a notes page  
4    is defined by the single notes master slide. Each notes page has the ability to override any of this information  
5    that it chooses by specifying local attribute values within the notes slide. Much like the notes master, the notes  
6    page contains some common structural elements, namely the following.

- 7         • *Common Slide Data* - Common properties and layout information for this notes page.  
8         • *Color Map Override* - Color Mapping to override the inherited color mapping for this notes page.

9    The above list defines the areas that can be used to override inherited components from the notes master.  
10   That is, these can be specifically defined on a per-slide basis via the above elements.

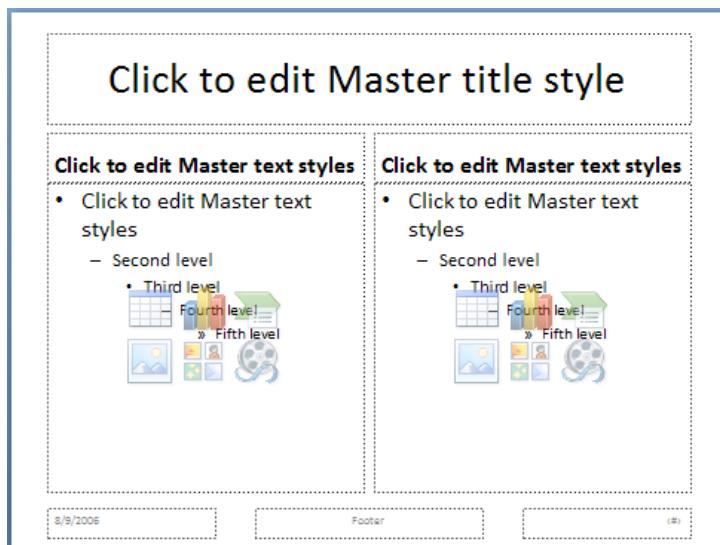


## 1    4.2.5     Slide Layouts

2    A *slide layout* inherits slide properties from the corresponding slide master and sets layout information for all  
 3    presentation slides that utilize this layout. Each presentation slide has the ability to override any of this  
 4    information that it chooses by specifying local attribute values within the presentation slide. Much like the  
 5    slide master, the slide layout contains some common structural elements:

- 6       • *Common Slide Data* - Common properties and layout information that will override properties set  
     7       within the slide master but will be inherited by all presentation slides that utilize this layout.
- 8       • *Color Map Override* - Color Mapping that will override the inherited color mapping from the slide  
     9       master but will be inherited by all presentation slides that utilize this layout.
- 10      • *Header and Footer* - Header and footer properties that will override properties set within the slide  
     11      master but will be inherited by all presentation slides that utilize this layout.
- 12      • *Timing Information* - Common timing properties used for animation, controls, etc. These will override  
     13      properties set within the slide master but will be inherited by all presentation slides that utilize this  
     14      layout.
- 15      • *Transition Information* - Slide transitioning information to be inherited by each presentation slide.  
     16      These will override properties set within the master slide but will be inherited by all presentation slides  
     17      that utilize this layout

18     The above list defines the areas that can be used to override inherited components from the master slide. That  
 19     is, these can be specifically defined on a per-layout basis via the above elements.



20

## 21    4.3     Comments

### 22    4.3.1     Introduction

23     This document describes the commenting feature for presentations as expressed in PresentationML. The  
 24     schema that defines this feature is pml-comments.xsd.

1 Note that it is important to keep in mind that comments are not shapes. The representation of them within the  
2 document is left entirely up to the generating application and are thus implementation specific.

### 3 **4.3.2 Functional Overview**

4 Readers of a presentation can provide feedback to the presentation author in the form of *comments*.  
5 Comments can only be applied to slides; they cannot be applied to masters of any type or to notes slides.

6 At first glance, comments appear to be shapes on the slide surface; however, they are not. Comments differ  
7 from regular shapes in two ways:

- 8     • Comments cannot be formatted or resized  
9     • The text contained within a comment cannot be formatted

### 10 **4.3.3 Comment Author List**

11 Presentations contain a list of all authors who have comments in the presentation. This list is commonly  
12 referred to as the *Comment Author List* (CAL). The CAL contains one entry for each author. Each entry is made  
13 up of five pieces of data: ID, Author Name, Author Initials, Last Index, and Color Index.

14 Each author that comments on a presentation is assigned an ID, which is a simple integer. This ID is unique  
15 within the presentation, and is assigned by the application itself.

16 The Author Name and Author Initials are taken from the application itself. If no initials are known to the  
17 application, the comment author is prompted upon the insertion of the initial comment. Both the Author  
18 Name and Author Initials are simple strings; that is, there is no association of the values with an identity (from  
19 a security or authentication perspective).

20 The Last Index (lastIdx) is an integer that documents how many comments the associated author has made in  
21 this presentation. When the author makes another comment, that comment is numbered using the next  
22 integer, and then this value is updated once again.

23 The Color Index (clrIdx) is an integer into a color table that is used to provide the solid background fill for the  
24 comment shape. The utility that this provides is that all of the comments by a particular author share the  
25 same color.

26 Here is an example of such a CAL:

```
27 <p:cmAuthorLst>
28   <p:cmAuthor id="0" name="Shawn" initials="SV" lastIdx="3" clrIdx="0" />
29   <p:cmAuthor id="1" name="Brian" initials="BJ" lastIdx="7" clrIdx="1" />
30 </p:cmAuthorLst>
```

31 To determine if an author is already in the CAL, one must consider only the Author Name and Author Initials  
32 data. If they both match an entry in the CAL, the author is already considered to be in the CAL; otherwise, the  
33 author is considered unique, and a separate entry is added for that author in the CAL.

1 When the presentation is saved using PresentationML, a separate Comment Authors part is created that  
 2 contains the CAL.

### 3 **4.3.4 Comment List**

4 Each slide within a presentation may contain zero or more comments. Each slide with at least one comment  
 5 starts a list of comments for that slide. Each entry in that list is made up of the following pieces of data:

- 6 • Author ID: This represents the ID of the author who created the comment. It matches an entry in the  
 7 CAL.
- 8 • Date/Time: This represents the date and time of the last modification of this particular comment.  
 9 Although expressed in UTC, its accuracy is dependent on the state of the machine making the edits.
- 10 • Index: This is the number assigned to this particular comment, and is one of the comments associated  
 11 with the specified author. This number should be equal to, or less than, the Last Index value for the  
 12 author in the CAL. There cannot be duplicate Indexes for the same author.
- 13 • Position: This defines the 2D coordinate for the location at which the comment shows up on the slide  
 14 surface. This is the position of the upper left point of the comment shape.
- 15 • The Text data includes all of the text that makes up the body of the comment. Note that this text is  
 16 expressed differently than other text as expressed in DrawingML. As this text contains no formatting,  
 17 and is strictly limited to text input, there is no additional data that needs to be stored.

18 Here is an example of a comment list for a slide:

```
19 <p:cmLst>
20   <p:cm authorId="0" dt="2006-01-30T22:45:13.597" idx="3">
21     <p:pos x="10" y="10" />
22     <p:text>Need to check with Mary on exact data values</p:text>
23   </p:cm>
24   <p:cm authorId="1" dt="2006-01-30T22:46:22.082" idx="1">
25     <p:pos x="106" y="106" />
26     <p:text>This chart is hard to read from afar</p:text>
27   </p:cm>
28 </p:cmLst>
```

29 When the presentation is saved using PresentationML, a separate Comments part is created for each comment  
 30 list.

## 31 **4.4 Animation**

### 32 **4.4.1 Introduction**

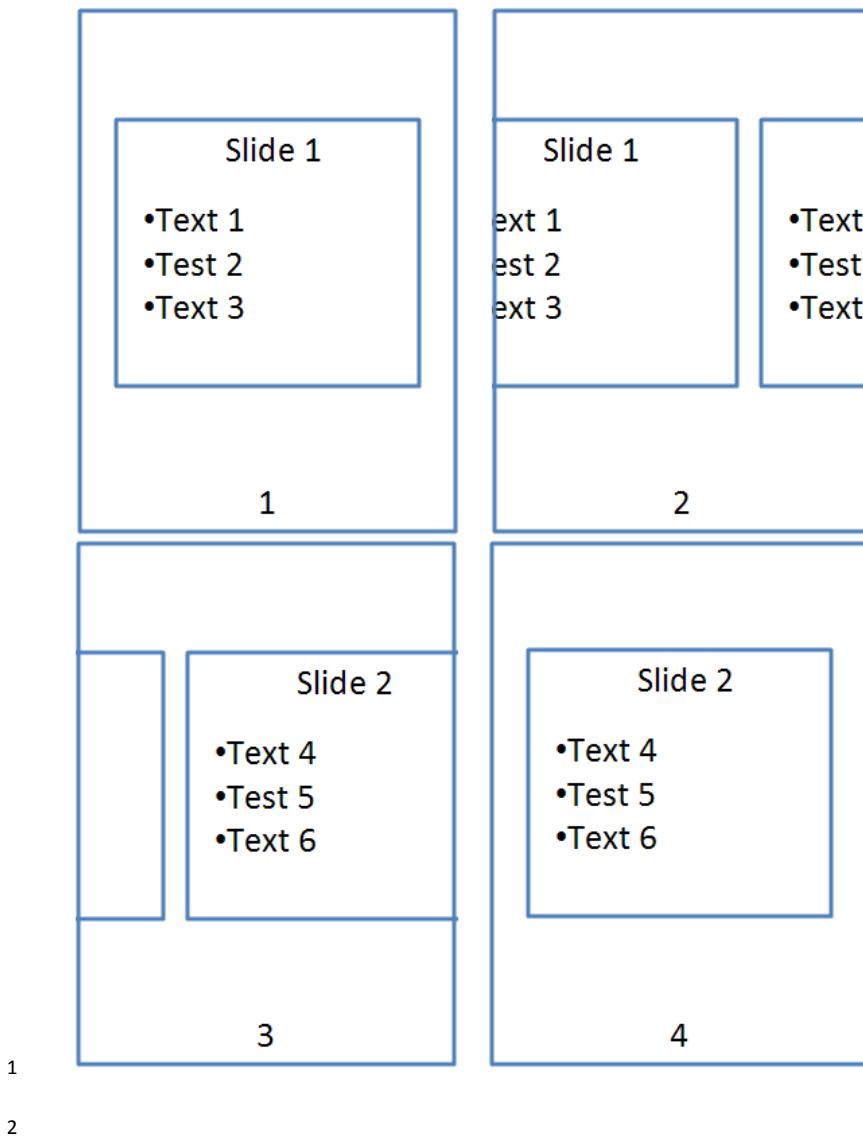
33 This subclause provides a high-level overview of the animation settings in PresentationML. This schema is  
 34 loosely based on the syntax and concepts from the Synchronized Multimedia Integration Language (SMIL), a  
 35 W3C Recommendation for describing multimedia presentations using XML.

- 1 The schema describes all the animations effects on that reside on a slide; it also describes the animation that
- 2 occurs when going from slide to slide (slide transition).
- 3 Animations on a slide are inherently time-based and consist of an animation effects on an object or text.
- 4 However, slide transitions do not follow this concept and always appear before any animation on a slide.
- 5 All elements described in this schema are contained within the slide XML file. Superficially, they are in the
- 6 transition and the timing element as shown below:

```
7   <p:sld>
8     <p:cSld> ...
9     <p:clrMapOvr> ...
10    <p:transition> ...
11      <p:timing> ...
12    </p:sld>
```

#### 13 **4.4.2 Slide Transitions**

- 14 Slide transitions are the animation effects that displayed in between slides. They are specified in the transition
- 15 element in the slide XML file. For example, consider a slide with a "push" slide transition as shown below:



3 The push element should be used as follows:

```

4   <p:transition>
5     <p:push dir="r"/>
6   </p:transition>
```

### 7 4.4.3 Timeline Overview

8 The timeline is an important aspect for animations on a slide. It moderates the amount of time that the  
9 animations are run from beginning to end. For example, it allows animation to be started when the slide is  
10 loaded or based on an event.

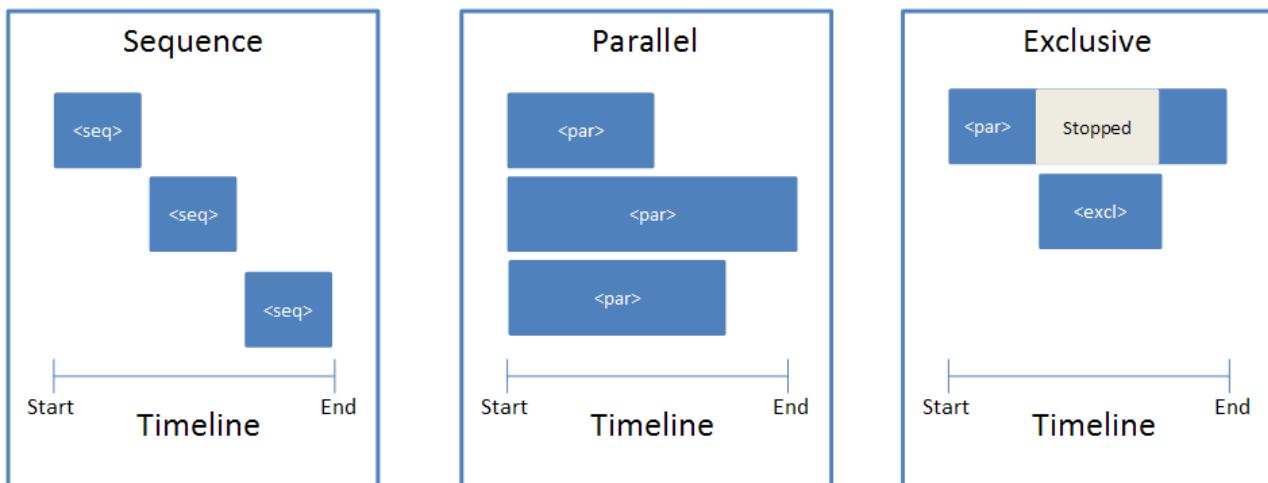
11 A timeline is composed of timing nodes that dictate at which point a certain animation is shown. A timeline  
12 can contain unlimited number of timing nodes; it can also have time nodes nested within them.

- 1 There are three types of time nodes:

Element	Name	Description
par	Parallel	This is a parallel time node and can be activated along with other parallel time node containers.
seq	Sequence	This is a sequence time node and it can only be activated when the one before it finishes.
excl	Exclusive	This time node is used to pause all other timelines when it is activated.

2

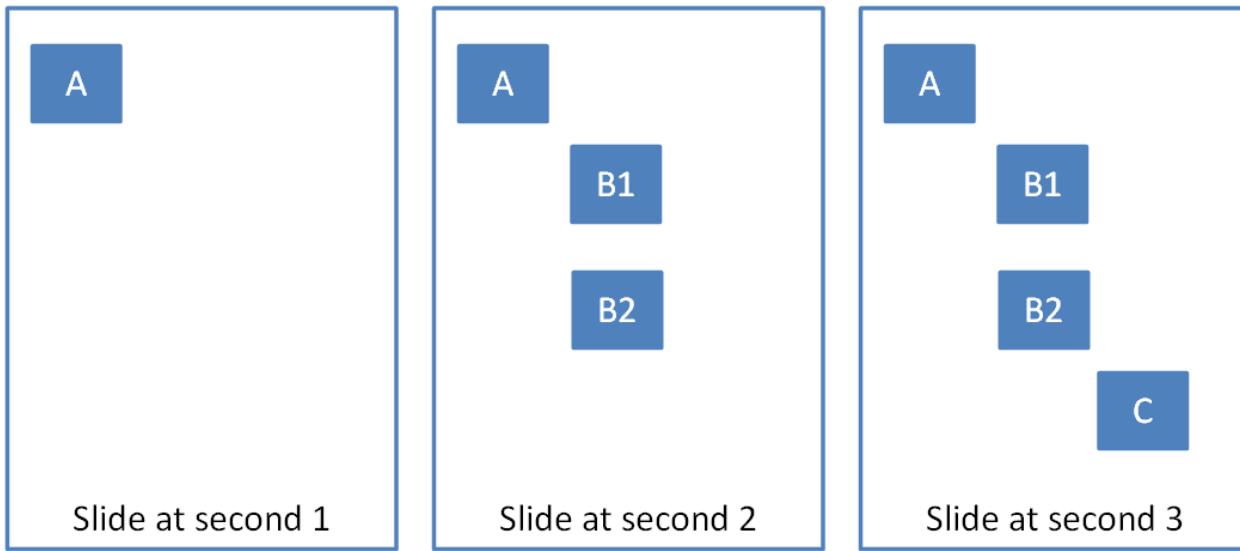
- 3 A conceptual diagram of this is shown below:



4

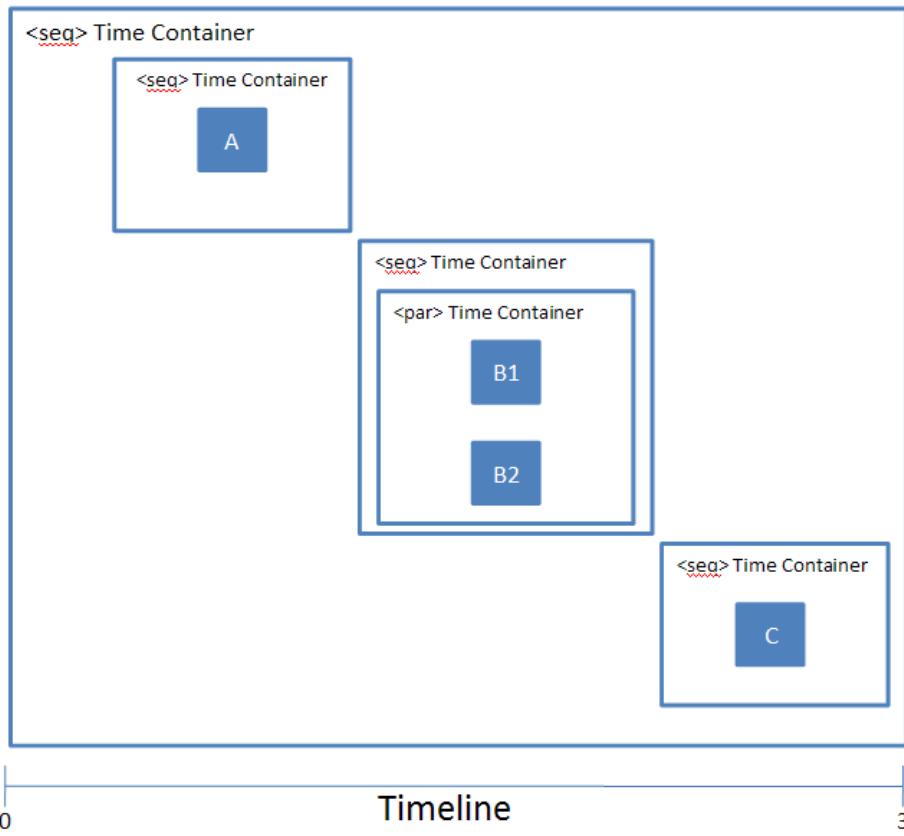
#### 4.4.4 Timeline Construction

- 5 To illustrate what the timeline looks like in the slide XML file, suppose we have four rectangles named A, B1, B2, and C that appear on a timeline three seconds long. Rectangle A appears at second 1, B1 and B2 appear together at second 2, and C appears at second 3, as shown below:



1

2 The timeline and time containers could look something like:



3

4 A typical timeline consists of the following structure:

```

1   <p:timing>
2     <p:tnLst>
3       <p:seq concurrent="1" nextAc="seek">
4         <p:stCondLst> ...
5         <p:cTn id="2" dur="indefinite" nodeType="mainSeq">
6           <p:childTnLst>
7             <p:seq> ... // Square A
8             <p:seq>
9               <par>... // Square B1
10              <par>... // Square B2
11            </p:seq>
12            <p:seq> ... // Square C
13            </p:childTnLst>
14          </p:cTn>
15          <p:prevCondLst> ...
16          <p:nextCondLst> ...
17        </seq>
18      </p:tnLst>
19      <p:bldLst> ...
20    </p:timing>

```

21 As show, this timeline starts with a timing element that represents the timeline. Within this timeline, there is a  
22 child element tnList, which contains a list of time nodes.

23 In this case, there is one main timing container, which is the seq element. Within this element there are a  
24 three of conditional elements, namely stCondList, nextCondList, and prevCondList. These elements contain  
25 condition properties that allow for the starting/stopping of the particular time node. This is explained in more  
26 detail in §4.4.6.

27 Following the stCondList element is the cTn element, which describes the properties for this node. Within this  
28 element is the childTnList, which contains the nested time nodes that describe the animation sequence  
29 mentioned above.

30 Finally, we have the bldList element, which is used to specify how objects with sub-shapes should be  
31 animated. More information can be found in §4.4.7.

#### 32 4.4.5 Animation Behaviors

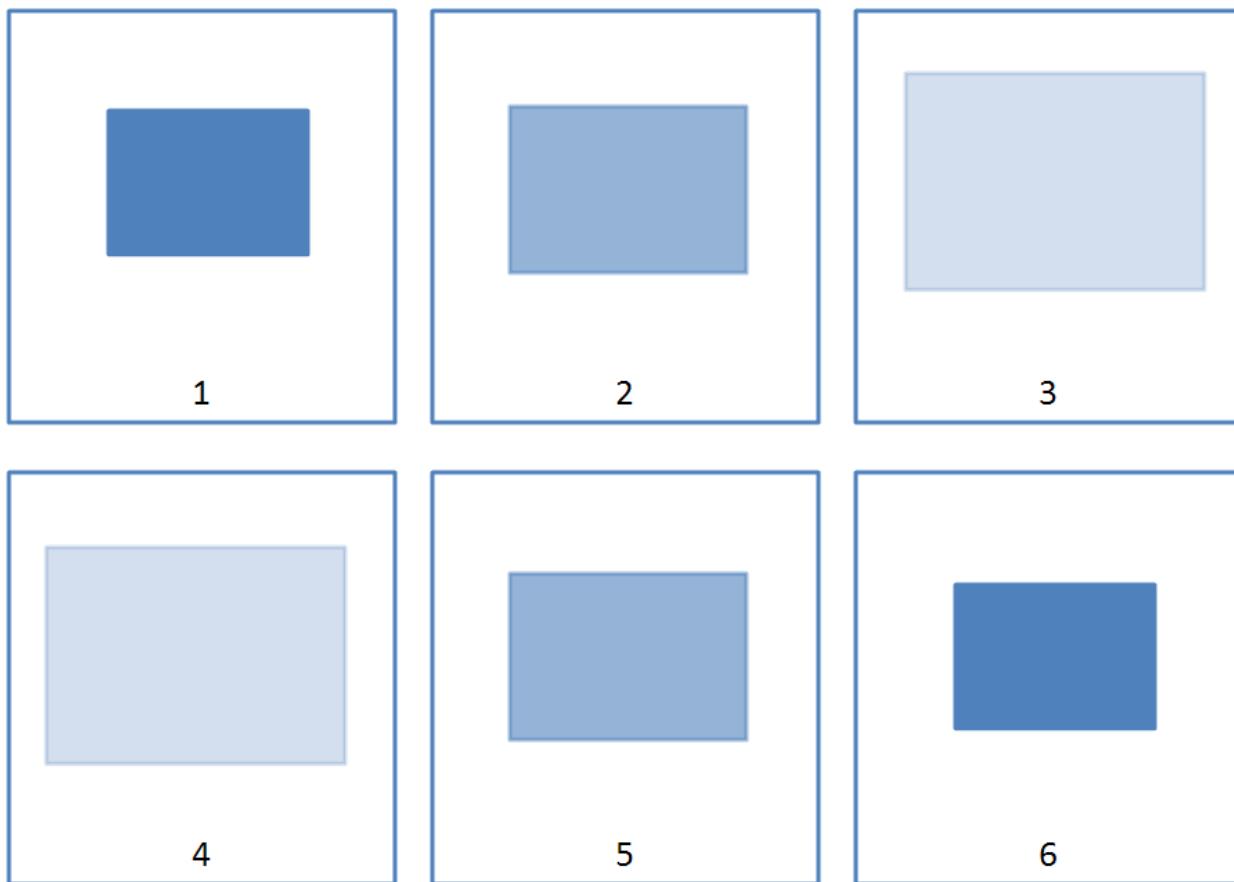
33 All animation consists of the following basic animation behaviors:

Element	Name	Description
anim	Animate	The animate behavior introduces a generic attribute animation that requires no semantic understanding of the attribute being animated. It can animate numbers.
animColor	Animate Color	This behavior animates the color of a particular element.

Element	Name	Description
animEffect	Animate Effect	This behavior provides the ability to do image transform/filter effects on elements.
animMotion	Animate Motion	Animate motion provides an abstracted way to move positioned elements. It provides the ability to specify from/to/by type motion as well as to use more detailed path descriptions for motion over polylines or bezier curves.
animRotation	Animate Rotation	This behavior allows rotation of an element.
animScale	Animate Scale	Allows animation of the width and/or height of an element over time.

1

- 2 A time node can combine multiple animations for a range of effects. For example, the "flash bulb" animation  
 3 which scales a shape larger while at the same time having it fade uses two animation behavior elements. An  
 4 example is shown below:



5

- 6 The representation for this animation effect in the time node element appears like:

```

1   <p:par>
2     <p:cTn id="5">
3       <p:stCondLst>....
4       <p:childTnLst>
5         <p:animEffect transition="out" filter="fade"> ...
6         <p:animScale>
7           <p:cBhvr>
8             <p:cTn id="7" dur="500" autoRev="1" fill="hold"/>
9             <p:tgtEl>
10               <p:spTgt spid="9"/>
11             </p:tgtEl>
12           </p:cBhvr>
13           <p:by x="105000" y="105000"/>
14         </p:animScale>
15       </p:childTnLst>
16     </p:cTn>
17   </p:par>

```

18 In this time node, we have two animation effects. One is creating a "fade" effect on the shape using the  
 19 animEffect element and the other is creating a "scale" effect using the animScale element. All animation  
 20 behaviors have a cBhvr and cTn element, which stores properties for the animation. For example, we can give  
 21 the animation behaviors an ID and attributes that set the duration of the animation. The spTgt specifies the  
 22 target shape to which this animation effect will be applied.

#### 23 **4.4.6 Conditional Properties**

24 Another important aspect of time nodes is conditional properties. There are four such conditions:

Element	Name	Description
stCondLst	Start Condition	Conditions that must be met for a time node to start.
prevCondLst	Previous Condition	Conditions that must be met for a time node to go back to the previous time node.
nextCondLst	Next Conditions	Conditions that must be met for a time node to advance to the next time node.
endCondLst	End Conditions	Conditions that must be met for a time node to end.

25

26 Conditional properties are useful for providing finer granularity as to exactly when a time node should be  
 27 activated. For example, suppose we have a shape with an entrance appearance after five seconds. The  
 28 stCondLst element should be used as follows:

```

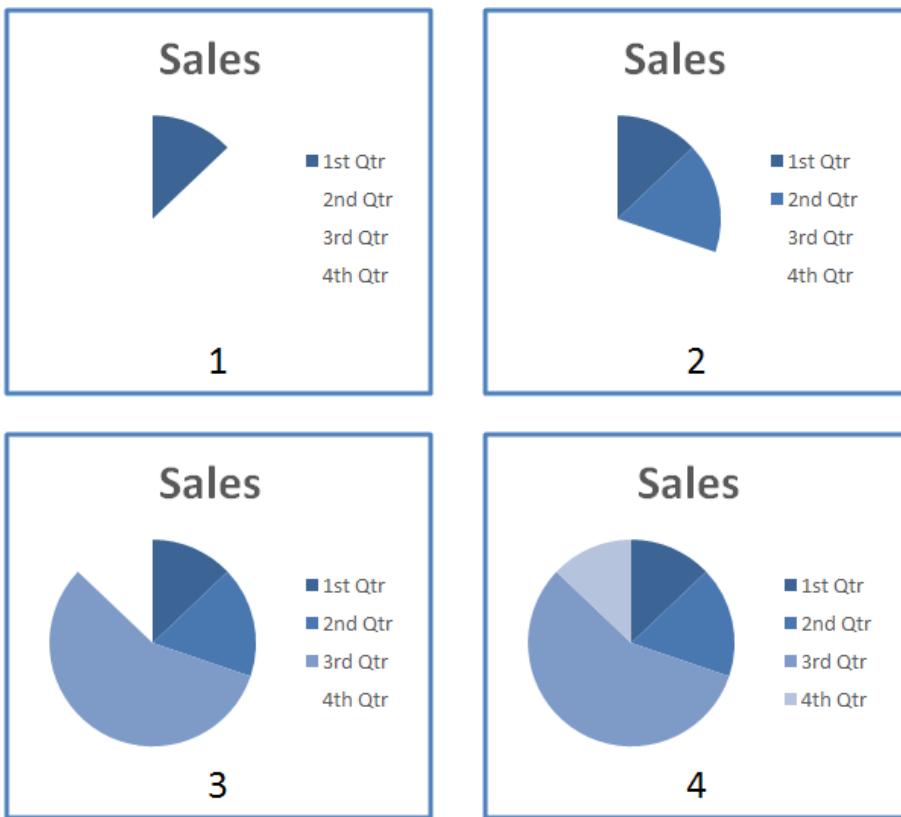
1   <p:par>
2     <p:cTn id="5">
3       <p:stCondLst>
4         <p:cond delay="5000"/>
5       </p:stCondLst>
6     </p:cTn>
7   </p:par>
```

#### 8    4.4.7     Build Animations

9    Another important aspect of animations is how they are built. This refers to how the different sub-shapes or  
10   sub-components of an object are displayed. The different objects that can have build properties are text,  
11   diagrams, and charts.

12   This is specified in the bldLst element.

13   For example, suppose we want to animate a pie chart, but based on category as shown below:



14

15   The representation of this in the slide XML looks like:

```

1   <p:bldLst>
2     <p:bldGraphic spid="4" grpId="0">
3       <p:bldSub>
4         <a:bldChart bld="category"/>
5       </p:bldSub>
6     </p:bldGraphic>
7   </p:bldLst>
```

8 The bldLst element contains children elements that describe how the different objects should be built. In this  
 9 case, there is only one graphic to be build, that with id 4. The bldGraphic element contains the bldSub  
 10 element, which describes how the object should be built. This element then contains the bldChart element  
 11 with the attribute bld set to category.

12 **End of informative text.**

## 13 4.5 Slide Synchronization

14 This subclause provides an overview of the p:SldSyncPr element in pml-slideSynchronizationData.xsd.

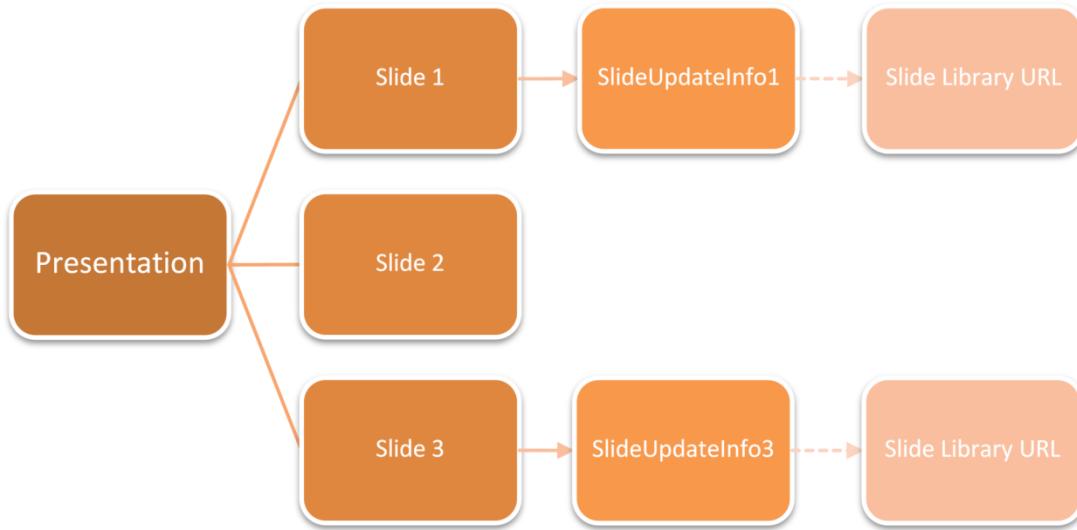
### 15 4.5.1 Introduction

16 A *Slide Library* is a library type in SharePoint Server that exclusively contains single-slide presentations. Users  
 17 are able to publish and reuse slides to/from these libraries. Furthermore, when a user inserts a slide from the  
 18 library into a presentation, she is able to create an update relationship so that she is notified when the original  
 19 slide on the server changes.

20 It is worth noting that the SlideUpdateInfo part in itself does not define the complete slide update  
 21 functionality. That part requires a SharePoint Slide Library or compatible server (e.g., a webdav server that  
 22 emulates SharePoint SOAP methods).

### 23 4.5.2 Slide Update Info

24 For each slide in a presentation that has an update relationship with its counterpart in a Slide Library, a Slide  
 25 Update Info part is created. The diagram below provides an overview of this relationship.



1

2

3 Each Slide Update Info part is stored under its own folder. For example:

4     /ppt/slidesUpdateInfo/slidesUpdateInfo1.xml  
 5     /ppt/slidesUpdateInfo/slidesUpdateInfo3.xml

6 The part is identified for each slide by a relationship with the following characteristics:

7     Type: http://.../slidesUpdateInfo  
 8     TargetMode: Internal  
 9     Target= "<Uri of the slidesUpdateInfo part for the slide>"

10 The content type of the update info part is application/vnd.openxmlformats-officedocument.presentationml.slidesUpdateInfo+xml.

12 It contains:

- 13     • Modified time of the slide on the server when it was inserted (stored in ISO 8061 format).
- 14     • Time the slide was inserted into the presentation.
- 15     • Regular ID of the slide on the server (saved as a string)

16 These Slide Update Info parts themselves have an external relationship to the Slide Library Url from which the  
 17 Slide was inserted.

18     Type: http://.../slidelibraryUrl  
 19     TargetMode: External  
 20     Target = "<Url of the Slide Library>"

21 Every Slide Update Info part should have exactly one occurrence of this relationship.

22 Samples:

```
1  slideupdateinfo1.xml  
2      <p:sldUpdatePr ... serverSldId="7991" serverSldModifiedTime="2006-03-08T18:48:33"  
3          clientInsertedTime="2006-03-10T06:02:33.975" />  
4  slideupdateinfo1.xml.rels  
5      <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>  
6          <Relationships xmlns="http://.../relationships">  
7              <Relationship Id="rId1" Type="http://.../slideUpdateUrl  
8                  Target="http://content/slides" TargetMode="External" />  
9          </Relationships>  
10     End of informative text.
```

# 1    5. Introduction to DrawingML

2    **This clause is informative.**

3    This clause contains a detailed introduction to the components of DrawingML.

4    **5.1 Basics**

5    **5.1.1 Introduction**

6    This subclause provides a high-level overview of the content described in the dml-baseTypes.xsd, dml-  
 7    compatibility.xsd and dml-lockedCanvas.xsd schemas. The aggregation of the elements within these schemas  
 8    encompass what has been labeled the DrawingML – Basics sub-clause. That is, the elements contained here  
 9    are considered to be commonly shared elements among the DrawingML framework.

10    **5.1.2 Overview**

11    This sub-clause is made up of four distinct pieces: Basic Elements, Colors, Compatibility, and Locked Canvas.  
 12    Together these make up the common elements that are shared across the DrawingML framework. These  
 13    elements are described below.

14    **5.1.3 Basic Elements**

15    When the common elements of the DrawingML framework are aggregated it can be seen that the most widely  
 16    used elements are property elements. These reside within every object and allow for the setting of both visual  
 17    and non-visual object-specific properties. The visual properties are those that affect the appearance of the  
 18    object when it is rendered on the screen. The non-visual properties on the other hand, do not affect the  
 19    object's appearance. Instead, these properties are used to store information normally hidden such as  
 20    identification numbers, human readable names for the objects and specific behaviour that should be obeyed  
 21    within the UI when manipulating the corresponding object.

22    **5.1.4 Colors**

23    The notion of color plays a vital role in the presentation of DrawingML objects within a document. Virtually all  
 24    objects are specified to have a corresponding color or set of colors. The notion of color is a common one  
 25    among graphically-inclined applications. Because of this and the fact that there are many different types of  
 26    graphic objects available today, we introduce the notion of several different types of color models. The  
 27    following color models can be used to specify color within a DrawingML based document.

- 28        1. RGB – Red, Green, Blue Color Model  
 29        2. HSL – Hue, Saturation, Luminance Color Model  
 30        3. Scheme – Scheme Based Color Model  
 31        4. Preset – Color Presets Color Model

1        5. System – Operating System Color Model

2        These different models allow document authors the choice as to which color model would be appropriate for  
3        their particular application. Each of these is detailed within the DrawingML Basics reference material.

4        **5.1.5      Compatibility**

5        Compatibility deals with the notion of legacy drawings. Legacy drawings are objects that were supported by  
6        previous versions of a generating application, but are no longer provided as an option. In order to store these  
7        drawing objects correctly, we introduce the notion of legacy drawing compatibility. This allows for the  
8        specification of information used to identify this legacy object and thus allow for full rendering support within  
9        current versions of the generating application.

10        **5.1.6      Locked Canvas**

11        Locked Canvas is a minor topic that is similar to compatibility in that it is used to render drawing objects that  
12        would otherwise not be recognized due to a lack of information. Locked Canvas, however, goes in the opposite  
13        direction from compatibility, and deals with objects that have been created and saved in the current version of  
14        a generating application and are being opened in a previous version of the generating application. The locked  
15        canvas element acts as a container for more advanced drawing objects. The notion of a locked canvas comes  
16        from the fact that the generating application opening the file cannot create this object and thus cannot  
17        perform edits either. Thus, the drawing object is locked from all UI adjustments that would normally take  
18        place.

19        **5.2        Audio and Video**

20        **5.2.1      Introduction**

21        DrawingML contains support for basic audio and video capabilities. The definitions for these structures can be  
22        found in dml-audioVideo.xsd.

23        **5.2.2      Functional Overview**

24        Presentation authors can specify that both audio and video can play while a slide is shown in slide show.  
25        When such media is inserted into a presentation, a presentation author can specify that the media is to play  
26        automatically (that is, in accordance with the slide's animation timeline) or in response to a mouse-click. In  
27        either case, media only plays for the duration of time specified, or until the slide changes, whichever is shorter.

28        Sources for audio content include CD-based files as well as the more traditional disc- or server-based files.

29        When inserting audio content stored on a CD, the author can specify a start and end track, as well as an index  
30        into each track. This information identifies the content from the CD to be played for the specified slide during a  
31        slide show.

32        In cases where the audio or video content is stored on a hard drive or server, the author can only specify the  
33        file itself; it will be played in its entirety, or until the slide changes.

### 5.2.3 DrawingML Syntax

- 1 In all three cases, the media objects themselves are stored on the slides using a picture shape. The picture  
 2 shape uses a blipFill to show the media object on the slide's surface. In both audio cases, the picture used is  
 3 the icon image, whereas in the video case, the picture used is the poster frame for the video file.  
 4
- 5 To express this information, the standard blipFill element is used to refer to the image file; and because this  
 6 refers to a file within the package, a relationship ID is used:

```

7   <p:pic>
8     <p:nvPicPr> ...</p:nvPicPr>
9     <p:blipFill>
10       <a:blip r:embed="rId4" r:link="" />
11       <a:stretch>
12         <a:fillRect />
13       </a:stretch>
14     </p:blipFill>
15     <p:spPr> ... </p:spPr>
16   </p:pic>
```

- 17 As the media objects are related to the slide's timeline—in both the automatic and mouse-click cases—they  
 18 must have interactivity information stored in the form of a hyperlink.
- 19 To express this information, a hyperlink is added to the non-visual shape properties; and because it is a  
 20 hyperlink, it, too, uses a relationship ID:

```

21   <p:pic>
22     <p:nvPicPr>
23       <p:cNvPr id="15" name="Rectangle 15" descr="">
24         <a:hlinkClick r:id="rId3" tgtFrame="" tooltip="" />
25       </p:cNvPr>
26     <p:cNvPicPr />
27       <p:nvPr> ... </p:nvPr>
28     </p:nvPicPr>
29     <p:blipFill> ... </p:blipFill>
30     <p:spPr> ... </p:spPr>
31   </p:pic>
```

- 32 The final piece of information required for each media type is the source bits. In both file-based media cases,  
 33 DrawingML needs to provide a mechanism to specify the location and file name for the media; this is in  
 34 contrast to the CD-based audio where only track information is required. Regardless of the type of source  
 35 information required, all of this is stored in application-specific non-visual properties. This is illustrated in the  
 36 following three XML islands representing each of the three media cases:

CD-Audio	<pre> &lt;p:pic&gt;   &lt;p:nvPicPr&gt; ... &lt;p:cNvPicPr /&gt;   &lt;p:nvPr&gt;     &lt;a:audioCd&gt;       &lt;a:st track="2" time="50" /&gt;       &lt;a:end track="3" time="22" /&gt;     &lt;/a:audioCd&gt;   &lt;/p:nvPr&gt;   ... &lt;/p:pic&gt;</pre>
File-Audio	<pre> &lt;p:pic&gt;   &lt;p:nvPicPr&gt; ... &lt;p:cNvPicPr /&gt;   &lt;p:nvPr&gt;     &lt;a:audioFile r:embed="" r:link="rId1" /&gt;   &lt;/p:nvPr&gt; &lt;/p:nvPicPr&gt; ... &lt;/p:pic&gt;</pre>
File-Video	<pre> &lt;p:pic&gt;   &lt;p:nvPicPr&gt; ... &lt;p:cNvPicPr /&gt;   &lt;p:nvPr&gt;     &lt;a:videoFile r:embed="" r:link="rId1" /&gt;   &lt;/p:nvPr&gt; &lt;/p:nvPicPr&gt; ... &lt;/p:pic&gt;</pre>

1

2 In the CD-Audio case, there is no capability to choose the particular CD drive that will contain the source. This  
 3 is a functional limitation.

4 While the default case is that media is linked, file-based media can also have the source bits included in the  
 5 package as a separate part. In this case, the relationships point not to an external file but, rather, to a part  
 6 inside the package.

## 7 5.3 Styles

### 8 5.3.1 Introduction

9 This piece of DrawingML deals with the definition of the shared aspects contained within a document theme.  
 10 The shared-style sheet defines an application-independent set of styling that can be applied to objects within a  
 11 document and which affects the look of the document and the information and objects it contains. For  
 12 example, in a presentation, shapes can have a certain look, whereas in an e-mail, all of the text can have  
 13 certain properties, and headings are styled.

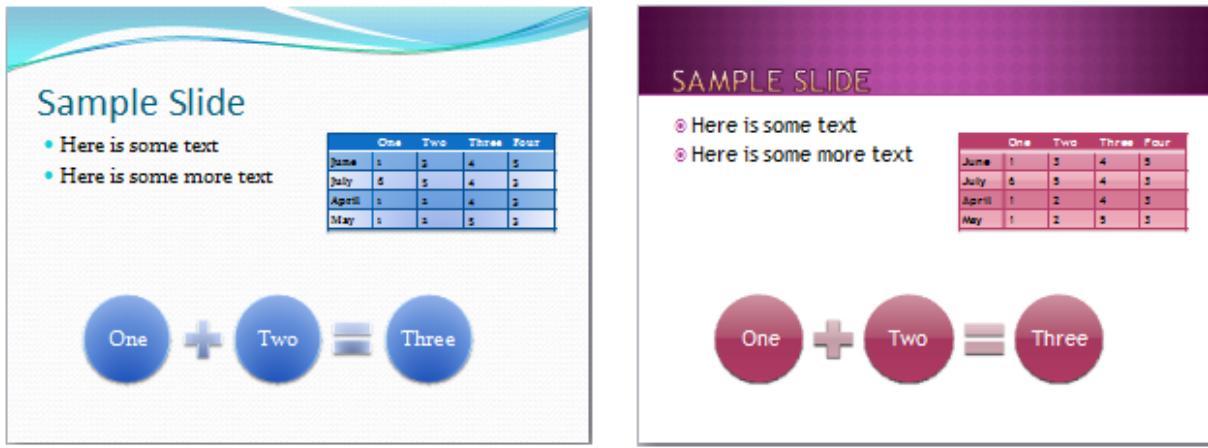
14 A second topic is the definition of a table style as used within DrawingML. A table style defines the look of a  
 15 table regardless of the data present in that table.

## 1    5.3.2      Shared Style Sheet

2    The shared-style sheet within DrawingML is responsible for containing different formatting options and style  
3    options that can be used within a given document.

### 4    5.3.2.1      Theme

5    The theme is the root-level complex type associated with a shared-style sheet. This complex type holds all of  
6    the different formatting options available to a theme, and defines the overall look and feel of a document  
7    when themed objects are used within the document. In figure 1 below, we can see an example of two  
8    different themes applied to the same slide in a presentation.



9

10   Figure 1: A theme applied to the same slide in a presentation. Not only does the font and colors change, but  
11   also the effects applied to the shapes and table.

12   A theme consists of four main parts, although the themeElements element is the piece that holds the main  
13   formatting defined within the theme. The other parts provide overrides, defaults, and additions to the  
14   information contained in themeElements. The complex type defining a theme, CT\_OfficeStyleSheet, is  
15   defined in the following manner:

```

16 <complexType name="CT_OfficeStyleSheet">
17   <sequence>
18     <element name="themeElements" type="CT_BaseStyles" minOccurs="1"
19       maxOccurs="1"/>
20     <element name="objectDefaults" type="CT_ObjectStyleDefaults"
21       minOccurs="0" maxOccurs="1"/>
22     <element name="extraClrSchemeLst" type="CT_ColorSchemeList"
23       minOccurs="0" maxOccurs="1"/>
24     <element name="custClrLst" type="CT_CustomColorList" minOccurs="0"
25       maxOccurs="1"/>
```

```

1      <element name="extLst" type="CT_OfficeArtExtensionList"
2          minOccurs="0" maxOccurs="1"/>
3      </sequence>
4      <attribute name="name" type="xsd:string" use="optional" default="" />
5      </complexType>

```

6 This complex type also holds a CT\_OfficeArtExtensionList, which is used for future extensibility of this  
7 complex type.

### 8 5.3.2.2 Theme Elements

9 The complex type CT\_BaseStyles defines the theme elements for a theme, and is the workhorse of the theme.  
10 The bulk of the shared theme information that is used by a given document is defined here. Within this  
11 complex type is defined a color scheme, a font scheme, and a style matrix (format scheme) that defines  
12 different formatting options for different pieces of a document. The complex type CT\_BaseStyles is defined in  
13 the following manner:

```

14     <complexType name="CT_BaseStyles">
15         <sequence>
16             <element name="clrScheme" type="CT_ColorScheme" minOccurs="1"
17                 maxOccurs="1"/>
18             <element name="fontScheme" type="CT_FontScheme" minOccurs="1"
19                 maxOccurs="1"/>
20             <element name="fmtScheme" type="CT_StyleMatrix" minOccurs="1"
21                 maxOccurs="1"/>
22             <element name="extLst" type="CT_OfficeArtExtensionList"
23                 minOccurs="0" maxOccurs="1"/>
24         </sequence>
25     </complexType>

```

### 26 5.3.2.3 Color Scheme

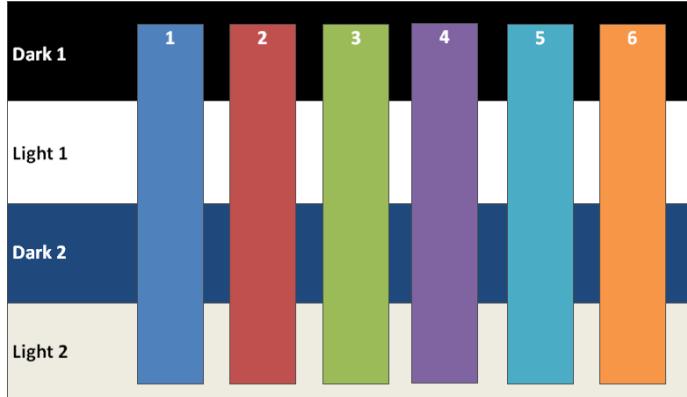
27 The complex type CT\_ColorScheme defines a set of colors for the theme. The set of colors consists of twelve  
28 color slots that can each hold a color of choice. The colors are organized in the following way:

- 29 • Dark 1 (dk1) – This represents a dark color, usually defined as a system text color
- 30 • Light 1 (lt1) – This represents a light color, usually defined as the system window color
- 31 • Dark 2 (dk2) – This represents a second dark color for use
- 32 • Light 2 (lt2) – This represents a second light color for use
- 33 • Accents 1 through 6 (accent1 through accent6) – These are six colors which can be used as accent  
34 colors in the theme
- 35 • Hyperlink (hlink) – The color of hyperlinks
- 36 • Followed Hyperlink (folHlink) – The color of a followed hyperlink

37 These colors define the theme colors that objects can utilize within a document. When an object uses a theme  
38 color, the color of the object can change when the theme is changed, but will always map to accent 1 if that

were the theme color used by the object. An example of theme colors defined and used can be seen in figure 2.

3



4 Figure 2: Sample colors defined and used for dark1/2, light1/2, and the six accent colors.

5 The complex type CT\_ColorScheme is defined in the following manner:

```
6
7   <complexType name="CT_ColorScheme">
8     <sequence>
9       <element name="dk1" type="CT_Color" minOccurs="1" maxOccurs="1"/>
10      <element name="lt1" type="CT_Color" minOccurs="1" maxOccurs="1"/>
11      <element name="dk2" type="CT_Color" minOccurs="1" maxOccurs="1"/>
12      <element name="lt2" type="CT_Color" minOccurs="1" maxOccurs="1"/>
13      <element name="accent1" type="CT_Color" minOccurs="1"
14        maxOccurs="1"/>
15      <element name="accent2" type="CT_Color" minOccurs="1"
16        maxOccurs="1"/>
17      <element name="accent3" type="CT_Color" minOccurs="1"
18        maxOccurs="1"/>
19      <element name="accent4" type="CT_Color" minOccurs="1"
20        maxOccurs="1"/>
21      <element name="accent5" type="CT_Color" minOccurs="1"
22        maxOccurs="1"/>
23      <element name="accent6" type="CT_Color" minOccurs="1"
24        maxOccurs="1"/>
25      <element name="hlink" type="CT_Color" minOccurs="1" maxOccurs="1"/>
26      <element name="folHlink" type="CT_Color" minOccurs="1"
27        maxOccurs="1"/>
```

```

1   <element name="extLst" type="CT_OfficeArtExtensionList"
2       minOccurs="0" maxOccurs="1"/>
3   </sequence>
4   <attribute name="name" type="xsd:string" use="required"/>
5   </complexType>
```

#### 6 5.3.2.4 Font Scheme

7 The complex type CT\_FontScheme defines a font pair. The pair consists of a major font and a minor font. An  
8 example of use would be the major font used in headings for a document and the minor font used for the  
9 paragraph parts of a document. The major and minor fonts are defined through a collection of font faces  
10 defined on a per-language basis. For example, one may define only a Latin-based font, or one can define many  
11 different fonts for different locals for a major or minor font. The font used in the document depends on the  
12 user's language.

13 The complex type CT\_FontScheme is defined in the following manner:

```

14  <complexType name="CT_FontScheme">
15      <sequence>
16          <element name="majorFont" type="CT_FontCollection" minOccurs="1"
17              maxOccurs="1"/>
18          <element name="minorFont" type="CT_FontCollection" minOccurs="1"
19              maxOccurs="1"/>
20          <element name="extLst" type="CT_OfficeArtExtensionList"
21              minOccurs="0" maxOccurs="1"/>
22      </sequence>
23      <attribute name="name" type="xsd:string" use="required"/>
24  </complexType>
```

#### 25 5.3.2.5 Major and Minor Font (Font Collection)

26 The complex type CT\_FontCollection defines a major and minor font which is used in the font scheme. A font  
27 collection consists of a font definition for Latin, East Asian, and complex script. On top of these three  
28 definitions, one may also define a font for use in a specific language or languages.

29 The complex type CT\_FontCollection is defined in the following manner:

```

30  <complexType name="CT_FontCollection">
31      <sequence>
32          <element name="latin" type="CT_TextFont" minOccurs="1"
33              maxOccurs="1"/>
34          <element name="ea" type="CT_TextFont" minOccurs="1" maxOccurs="1"/>
35          <element name="cs" type="CT_TextFont" minOccurs="1" maxOccurs="1"/>
36          <element name="font" type="CT_SupplementalFont" minOccurs="0"
37              maxOccurs="unbounded"/>
```

```

1   <element name="extLst" type="CT_OfficeArtExtensionList"
2       minOccurs="0" maxOccurs="1"/>
3   </sequence>
4 </complexType>
```

### 5.3.2.6 Supplemental Font

6 The complex type CT\_SupplementalFont defines an additional font that is used for language specific fonts in  
7 themes. For example, one can specify a font that gets used only within the Japanese language context.

8 The complex type CT\_SupplementalFont is defined in the following manner:

```

9   <complexType name="CT_SupplementalFont">
10      <attribute name="script" type="xsd:string" use="required"/>
11      <attribute name="typeface" type="ST_TextTypeface" use="required"/>
12   </complexType>
```

### 5.3.2.7 Format Scheme (Style Matrix)

14 The complex type CT\_StyleMatrix defines a set of formatting options, which can be referenced by documents  
15 that apply a certain style to a given part of an object. For example, in a given shape, say a rectangle, one can  
16 reference a themed line style, themed effect, and themed fill that would be theme specific and change when  
17 the theme is changed. All of these formatting options are defined within this style matrix. Background fills can  
18 also be contained within the style matrix. This is most useful to presentations (but not unique to  
19 presentations) which reference different background fills as slide backgrounds. Three sets of each type of  
20 formatting are defined, corresponding to subtle, moderate, and intense versions of each style. Combinations  
21 of styles are used to create, for example a shape style. An example of this would be a shape style utilizing a  
22 subtle fill, moderate line, and intense effect to define the overall look of a shape.

23 The complex type CT\_StyleMatrix is defined in the following manner:

```

24   <complexType name="CT_StyleMatrix">
25      <sequence>
26         <element name="fillStyleLst" type="CT_FillStyleList" minOccurs="1"
27             maxOccurs="1"/>
28         <element name="lnStyleLst" type="CT_LineStyleList" minOccurs="1"
29             maxOccurs="1"/>
30         <element name="effectStyleLst" type="CT_EffectStyleList"
31             minOccurs="1" maxOccurs="1"/>
32         <element name="bgFillStyleLst" type="CT_BackgroundFillStyleList"
33             minOccurs="1" maxOccurs="1"/>
34     </sequence>
35     <attribute name="name" type="xsd:string" use="optional" default="" />
36   </complexType>
```

1    5.3.2.8        [Fill Style List](#)

2    The complex type CT\_FillStyleList defines a set of three fill types. Currently, only three fill types are used,  
 3    corresponding to subtle, moderate, and intense fills, but the number of fills that can be defined is unbounded.  
 4    An example of three fills that could be present can be seen in figure 3. In this figure, we have a solid blue fill in  
 5    the subtle slot, a gradient fill in the moderate slot, and an image fill in the intense slot.



6  
 7    Figure 3: Three different fills increasing in relative intensity.

8    The complex type CT\_FillStyleList is defined in the following manner:

```
9    <complexType name="CT_FillStyleList">
10   <sequence>
11     <group ref="EG_FillProperties" minOccurs="3" maxOccurs="unbounded"/>
12   </sequence>
13 </complexType>
```

14   [5.3.2.9        Line Style List](#)

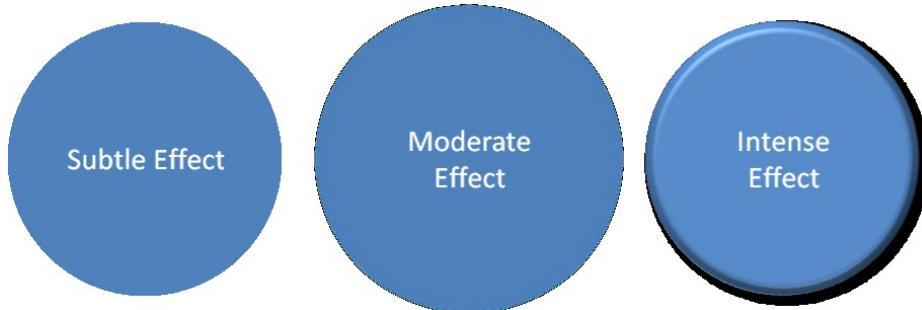
15   The complex type CTLineStyleList defines a set of three line styles. As with the fill style list, currently only  
 16   three styles are utilized corresponding to a subtle line, moderate line, and intense line.

17   The complex type CTLineStyleList is defined in the following manner:

```
18   <complexType name="CTLineStyleList">
19   <sequence>
20     <element name="ln" type="CTLineProperties" minOccurs="3"
21       maxOccurs="unbounded"/>
22   </sequence>
23 </complexType>
```

24   [5.3.2.10        Effect Style List](#)

25   The complex type CTEffectStyleList defines a set of three effect styles. As with the previously mentioned  
 26   style lists, three styles are currently utilized corresponding to subtle, moderate, and intense effect styles, but  
 27   the list remains unbounded. In figure 4 we see subtle, moderate, and intense effects applied to a given shape  
 28   with a blue fill. The subtle effect is, basically, no effect, whereas the moderate effect is a glow surrounding the  
 29   shape, and the intense effect is a 3-D bevel along with a shadow applied to the shape.



1

2 Figure 4: Subtle, moderate, and intense effects applied to a shape that has a blue fill.

3 The complex type CT\_EffectStyleList is defined in the following manner:

```

4 <complexType name="CT_EffectStyleList">
5   <sequence>
6     <element name="effectStyle" type="CT_EffectStyleItem" minOccurs="3"
7       maxOccurs="unbounded"/>
8   </sequence>
9 </complexType>
```

#### 10 5.3.2.11 Effect Style Item

11 The complex type CT\_EffectStyleItem holds the properties for a given effect style. Within this complex type,  
 12 one can define a list of effects (blur, shadow, reflection, etc.) along with any 3-D properties that are to be  
 13 applied to an object. A basic example of how effects can be applied to a shape can be seen in figure 4.

14 The complex type CT\_EffectStyleItem is defined in the following manner:

```

15 <complexType name="CT_EffectStyleItem">
16   <sequence>
17     <group ref="EG_EffectProperties" minOccurs="1" maxOccurs="1"/>
18     <element name="scene3d" type="CT_Scene3D" minOccurs="0"
19       maxOccurs="1"/>
20     <element name="sp3d" type="CT_Shape3D" minOccurs="0" maxOccurs="1"/>
21   </sequence>
22 </complexType>
```

#### 23 5.3.2.12 Background Fill Style List

24 The complex type CT\_BackgroundFillStyleList defines a set of three fill types similar to the fill style list.  
 25 Again, they define three fill types corresponding to subtle, moderate, and intense background fills but the list  
 26 itself is unbounded. The background-fills are meant, for example, to be applied to a slide background, or as  
 27 the background fill in a shape or table.

28 The complex type CT\_BackgroundFillStyleList is defined in the following manner:

```

1   <complexType name="CT_BackgroundFillStyleList">
2     <sequence>
3       <group ref="EG_FillProperties" minOccurs="3" maxOccurs="unbounded"/>
4     </sequence>
5   </complexType>

```

### 5.3.2.13 Table Styles

Table styles are responsible for the rapid formatting that can be applied to a table. This rapid formatting takes different things into account, such as if the first row or last row should be emphasized, or if there is some type of banding present on the table. All of these different types of formatting can be defined within a table style.

An example of different table styles in use on the same table can be seen in figure 1.

	Red	Blue	Yellow
1 <sup>st</sup> Qtr	21.5	18.3	4.5
2 <sup>nd</sup> Qtr	17.4	3.6	2.2
3 <sup>rd</sup> Qtr	9.1	19.8	7.9
4 <sup>th</sup> Qtr	12.2	13.4	12.1

	Red	Blue	Yellow
1 <sup>st</sup> Qtr	21.5	18.3	4.5
2 <sup>nd</sup> Qtr	17.4	3.6	2.2
3 <sup>rd</sup> Qtr	9.1	19.8	7.9
4 <sup>th</sup> Qtr	12.2	13.4	12.1

	Red	Blue	Yellow
1 <sup>st</sup> Qtr	21.5	18.3	4.5
2 <sup>nd</sup> Qtr	17.4	3.6	2.2
3 <sup>rd</sup> Qtr	9.1	19.8	7.9
4 <sup>th</sup> Qtr	12.2	13.4	12.1

11

Figure 5: Different table styles in use.

The application of a table style to a table formats the table in its entirety. There are numerous complex types that make up a table style. The pieces of a table style will be discussed first, before defining the table style itself.

### 5.3.2.14 Cell 3D

The complex type CT\_Cell3D defines all of the 3-D properties that an individual cell can hold. In the case of a table, these 3-D properties can be a bevel along with a material and a light rig for the cell. More explanation of these three pieces of a CT\_Cell3D can be found in the document on 3-D. These properties are applied on a per-cell basis, rather than to the table as a whole. A CT\_Cell3D is defined in the following manner:

```

21   <xsd:complexType name="CT_Cell3D">
22     <xsd:sequence>
23       <xsd:element name="bevel" type="CT_Bevel" minOccurs="1"
24         maxOccurs="1" />

```

```

1   <xsd:element name="lightRig" type="CT_LightRig" minOccurs="0"
2     maxOccurs="1" />
3   <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
4     maxOccurs="1" />
5 </xsd:sequence>
6   <xsd:attribute name="prstMaterial" type="ST_PresetMaterialType"
7     use="optional" default="plastic" />
8 </xsd:complexType>
```

9 This complex type also holds a CT\_OfficeArtExtension. This complex type is used for future extensibility and  
10 will be seen elsewhere throughout the tables area.

### 11 5.3.2.15 Themeable Styles

12 There are three groups and a complex type that account for style pieces that can be themed. These themed-  
13 aspects either pull from the style matrix, or they define an actual fill or effect for example. If they pull their  
14 style from the matrix, then an update to the document theme will also update the particular style dynamically.  
15 The three groups consist of the following groups:

```

16   <xsd:group name="EG_ThemeableFillStyle">
17     <xsd:choice>
18       <xsd:element name="fill" type="CT_FillProperties" minOccurs="1"
19         maxOccurs="1" />
20       <xsd:element name="fillRef" type="CT_StyleMatrixReference"
21         minOccurs="1" maxOccurs="1" />
22     </xsd:choice>
23   </xsd:group>
24   <xsd:group name="EG_ThemeableEffectStyle">
25     <xsd:choice>
26       <xsd:element name="effect" type="CT_EffectProperties" minOccurs="1"
27         maxOccurs="1" />
28       <xsd:element name="effectRef" type="CT_StyleMatrixReference"
29         minOccurs="1" maxOccurs="1" />
30     </xsd:choice>
31   </xsd:group>
32   <xsd:group name="EG_ThemeableFontStyles">
33     <xsd:choice>
34       <xsd:element name="font" type="CT_FontCollection" minOccurs="1"
35         maxOccurs="1" />
36       <xsd:element name="fontRef" type="CT_FontReference" minOccurs="1"
37         maxOccurs="1" />
38     </xsd:choice>
39   </xsd:group>
```

1 The three groups above all give a choice between using a themed style, or defining the style themselves. The  
 2 last type in this group is a complex type used to perform the same task as the above three, only it deals with  
 3 the lines in the table. The complex type CT\_ThemeableLineStyle is defined as:

```

4   <xsd:complexType name="CT_ThemeableLineStyle">
5     <xsd:choice>
6       <xsd:element name="ln" type="CT_LineProperties" minOccurs="1"
7         maxOccurs="1" />
8       <xsd:element name="lnRef" type="CT_StyleMatrixReference"
9         minOccurs="1" maxOccurs="1" />
10      </xsd:choice>
11    </xsd:complexType>
```

### 12 5.3.2.16 On/Off Property Definition

13 The simple type ST\_OnOffStyleType defines a type with values of on, off, or default. The default value means  
 14 to follow the parent settings. This comes into play for a themed property, which means follow what the theme  
 15 says. For an unthemed property, this means to follow the parent setting in the property inheritance chain.

### 16 5.3.2.17 Text Properties

17 The complex type CT\_TableStyleTextStyle defines the table text properties that can be styled. The text  
 18 properties contain a reference to a themeable font style along with bold and italic being enabled or disabled.  
 19 The CT\_TableStyleTextStyle is defined in the following manner:

```

20   <xsd:complexType name="CT_TableStyleTextStyle">
21     <xsd:sequence>
22       <xsd:group ref="EG_ThemeableFontStyles" minOccurs="0"
23         maxOccurs="1" />
24       <xsd:group ref="EG_ColorChoice" minOccurs="0" maxOccurs="1" />
25       <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
26         maxOccurs="1" />
27     </xsd:sequence>
28     <xsd:attribute name="b" type="ST_OnOffStyleType" use="optional"
29       default="def" />
30     <xsd:attribute name="i" type="ST_OnOffStyleType" use="optional"
31       default="def" />
32   </xsd:complexType>
```

### 33 5.3.2.18 Cell Border Properties

34 The complex type CT\_TableCellBorderStyle defines the properties of the borders that can be styled in a table.  
 35 The border styles can be applied to the following different types of borders in a table:

- 36 • left – left border
- 37 • right – right border
- 38 • top – top border

- 1     • bottom – bottom border
- 2     • insideH – inner horizontal borders
- 3     • insideV – inner vertical borders
- 4     • tl2br – diagonal border from top left corner to bottom right corner
- 5     • tr2bl – diagonal border from top right corner to bottom left corner

6     The complex type is defined in the following manner:

```

7     <xsd:complexType name="CT_TableCellBorderStyle">
8         <xsd:sequence>
9             <xsd:element name="left" type="CT_ThemeableLineStyle" minOccurs="0"
10                maxOccurs="1" />
11             <xsd:element name="right" type="CT_ThemeableLineStyle" minOccurs="0"
12                maxOccurs="1" />
13             <xsd:element name="top" type="CT_ThemeableLineStyle" minOccurs="0"
14                maxOccurs="1" />
15             <xsd:element name="bottom" type="CT_ThemeableLineStyle"
16                minOccurs="0" maxOccurs="1" />
17             <xsd:element name="insideH" type="CT_ThemeableLineStyle"
18                minOccurs="0" maxOccurs="1" />
19             <xsd:element name="insideV" type="CT_ThemeableLineStyle"
20                minOccurs="0" maxOccurs="1" />
21             <xsd:element name="tl2br" type="CT_ThemeableLineStyle" minOccurs="0"
22                maxOccurs="1" />
23             <xsd:element name="tr2bl" type="CT_ThemeableLineStyle" minOccurs="0"
24                maxOccurs="1" />
25             <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
26                maxOccurs="1" />
27         </xsd:sequence>
28     </xsd:complexType>
```

### 29     5.3.2.19      Cell Style Properties

30     The complex type CT\_TableCellStyle contains the definition for cell properties which can be styled.  
31     Within this complex type are held the border style, cell fill style, and the cell 3-D. The complex type is defined  
32     in the following manner:

```

33     <xsd:complexType name="CT_TableCellStyle">
34         <xsd:sequence>
35             <xsd:element name="tcBdr" type="CT_TableCellBorderStyle"
36                minOccurs="0" maxOccurs="1" />
```

```

1   <xsd:group ref="EG_ThemeableFillStyle" minOccurs="0"
2     maxOccurs="1" />
3   <xsd:element name="cell3D" type="CT_Cell3D" minOccurs="0"
4     maxOccurs="1" />
5   </xsd:sequence>
6 </xsd:complexType>
```

### 5.3.2.20 Table Background Style

The complex type CT\_TableBackgroundStyle defines the style elements associated with the background of the table. The table background style can contain a fill and effect. The complex type is defined in the following manner:

```

11  <xsd:complexType name="CT_TableBackgroundStyle">
12    <xsd:sequence>
13      <xsd:group ref="EG_ThemeableFillStyle" minOccurs="0"
14        maxOccurs="1" />
15      <xsd:group ref="EG_ThemeableEffectStyle" minOccurs="0"
16        maxOccurs="1" />
17    </xsd:sequence>
18 </xsd:complexType>
```

### 5.3.2.21 Table Part Style

The complex type CT\_TablePartStyle defines a structure for holding the style information for a single part of the table. The table is broken up in 13 different parts, which are explained in the next subclause of this document. A table part contains a text style and a cell style and is defined in the following manner:

```

23  <xsd:complexType name="CT_TablePartStyle">
24    <xsd:sequence>
25      <xsd:element name="tcTxStyle" type="CT_TableStyleTextStyle"
26        minOccurs="0" maxOccurs="1" />
27      <xsd:element name="tcStyle" type="CT_TableStyleCellStyle"
28        minOccurs="0" maxOccurs="1" />
29    </xsd:sequence>
30 </xsd:complexType>
```

### 5.3.2.22 Table Style

The complex type CT\_TableStyle defines the actual table style. Apart from the table background, 13 different parts that can be defined in a table style. These parts work together to define the styling for a table, given the 6 combinations of on/off states for the first row, first column, last row, last column, row banding, and column banding options. The different parts of a table that make up a table style are:

- tableBg – table background (this is not a CT\_TablePartStyle)
- wholeTable – formatting for the entire table

- 1     • band1Horizontal – applied when row banding is enabled, this is the first row style, which alternates  
2       with band2Horizontal
- 3     • band2Horizontal – applied when row banding is enabled, this is the second row style, which alternates  
4       with band1Horizontal
- 5     • band1Vertical – applied when column banding is enabled, this is the first column style, which  
6       alternates with band2Vertical
- 7     • band2Vertical – applied when column banding is enabled, this is the second column style, which  
8       alternates with band1Vertical
- 9     • lastCol – formatting applied to the last column when last column formatting is enabled
- 10    • firstCol – formatting applied to the first column when first column formatting is enabled
- 11    • lastRow – formatting applied to the last row when last row formatting is enabled
- 12    • firstRow – formatting applied to the first row when first row formatting is enabled
- 13    • seCell – formatting applied to the cell in the southeast corner of the table when last column and last  
14      row are enabled
- 15    • swCell – formatting applied to the cell in the southwest corner of the table when first column and last  
16      row are enabled
- 17    • neCell – formatting applied to the cell in the northeast corner of the table when the last column and  
18      first row are enabled
- 19    • nwCell – formatting applied to the cell in the northwest corner of the table when the first column and  
20      first row are enabled

21   The table style is defined in the following manner:

```

22 <xsd:complexType name="CT_TableStyle">
23   <xsd:sequence>
24     <xsd:element name="tblBg" type="CT_TableBackgroundStyle"
25       minOccurs="0" maxOccurs="1" />
26     <xsd:element name="wholeTbl" type="CT_TablePartStyle" minOccurs="0"
27       maxOccurs="1" />
28     <xsd:element name="band1H" type="CT_TablePartStyle" minOccurs="0"
29       maxOccurs="1" />
30     <xsd:element name="band2H" type="CT_TablePartStyle" minOccurs="0"
31       maxOccurs="1" />
32     <xsd:element name="band1V" type="CT_TablePartStyle" minOccurs="0"
33       maxOccurs="1" />
34     <xsd:element name="band2V" type="CT_TablePartStyle" minOccurs="0"
35       maxOccurs="1" />
36     <xsd:element name="lastCol" type="CT_TablePartStyle" minOccurs="0"
37       maxOccurs="1" />
38     <xsd:element name="firstCol" type="CT_TablePartStyle" minOccurs="0"
39       maxOccurs="1" />
40     <xsd:element name="lastRow" type="CT_TablePartStyle" minOccurs="0"
41       maxOccurs="1" />
```

```

1      <xsd:element name="seCell" type="CT_TablePartStyle" minOccurs="0"
2          maxOccurs="1" />
3      <xsd:element name="swCell" type="CT_TablePartStyle" minOccurs="0"
4          maxOccurs="1" />
5      <xsd:element name="firstRow" type="CT_TablePartStyle" minOccurs="0"
6          maxOccurs="1" />
7      <xsd:element name="neCell" type="CT_TablePartStyle" minOccurs="0"
8          maxOccurs="1" />
9      <xsd:element name="nwCell" type="CT_TablePartStyle" minOccurs="0"
10         maxOccurs="1" />
11     <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
12         maxOccurs="1" />
13   </xsd:sequence>
14   <xsd:attribute name="styleId" type="ST_Guid" use="required" />
15   <xsd:attribute name="styleName" type="xsd:string" use="required" />
16 </xsd:complexType>
```

17 Also contained within the table style are an ID and a name. The name shows up as the name for the table  
 18 style, and the ID is the unique id (GUID) that is associated with the table style.

### 19 5.3.2.23 Table Style List

20 The final complex type dealing with table styles is simply a list of table styles. Also contained in this list is the  
 21 default style which gets applied to the table when a default is to be used. The complex type  
 22 CT\_TableStyleList is defined in the following manner:

```

23   <xsd:complexType name="CT_TableStyleList">
24     <xsd:sequence>
25       <xsd:element name="tblStyle" type="CT_TableStyle" minOccurs="0"
26           maxOccurs="unbounded" />
27     </xsd:sequence>
28     <xsd:attribute name="def" type="ST_Guid" use="required" />
29   </xsd:complexType>
```

## 30 5.4 Text

### 31 5.4.1 Introduction

32 This subclause provides a high-level overview of the content described in the following schemas: dml-text.xsd,  
 33 dml-textParagraph.xsd, dml-textRun.xsd, dml-textCharacter.xsd and dml-textBullet.xsd.

34 The best way to understand these schemas as they relate to one another is to learn about the DrawingML Text  
 35 file format in the following order.

- 36 • Text Overview
- 37 • Body Level Properties

1       • Paragraph Level Properties

2       • Run and Character Level Properties

3      Companion schemas build on the ones discussed in this document. As these are encountered below, pointers  
4      to them are provided.

5      This subclause provides a structured breakdown of the text portion of the DrawingML file format. Other  
6      subclauses build on this foundation and explain more about topics such as text frame, text styles, text fields,  
7      and embedded fonts.

8      Note that DrawingML Text described within this document is distinct from WordprocessingML Text in that  
9      the file framework surrounding it has been optimized for use in a graphics-centric, presentation-like manner.  
10     As a contrast, WordprocessingML Text allows for text to be stored in a format that is optimized for layout of  
11     printed documents but not for art-based text as is found within the <a:> namespace.

12     Note that the use of the "p" namespace within this document references to the PresentationML-specific  
13     schemas while the use of the "a" namespace within this document references to the DrawingML-specific  
14     schemas.

15     

## 5.4.2 Overview

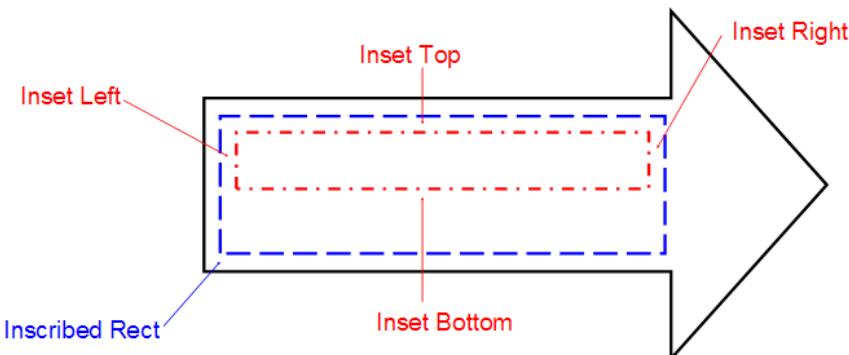
16     Consider an XML tree that has the following basic structure:

```
17       <p:sld>
18        <p:cSld>
19        <p:spTree>
20        <p:sp>
21        <p:txBody>
22           Your text here!
23           (not really, there are other elements needed within here)
24           </p:txBody>
25        </p:sp>
26        <p:sp>
27        </p:sp>
28        ...
29        </p:spTree>
30        </p:cSld>
31       </p:sld>
```

32     In the structure above, we are interested in the content contained within the matching p:txBody tags. The  
33     understanding of this tag in relation to the basic slide structure above encompasses the schema background  
34     needed to digest effectively the remainder of this description.

35     Note that shapes are the standard container within which all text resides. Usually, the shape does not have any  
36     visual properties attached to it and thus no visible shape is rendered; nonetheless, a shape is still present and  
37     does house any content text.

- 1 Each shape contains an inset rectangle that houses any text attached to that shape. The shape has margins or
- 2 *insets* that buffer this rectangle on all four sides (top, bottom, left, and right) just like margins on a page. When
- 3 thinking about text within a shape, it is useful to keep these inset properties in mind.
- 4 An illustration of this is provided below.



- 5
- 6 Let's look at the different element tags contained within p:txBody. Listed below are only those tags discussed
- 7 here. (Note that this sample framework is a skeleton and does not fully show all elements and attributes
- 8 needed.)

```

9   <p:txBody>
10    <a:bodyPr />           required, only listed once.
11    <a:lstStyle />         optional, if present only listed once.
12    <a:p>
13      <a:pPr />          required, no limit on the number of instances.
14      <a:r>                optional, if present only listed once.
15        <a:rPr />         required, no limit on the number of instances.
16        <a:t>Your text here!</a:t>
17                      Actual text for this run is contained here.
18        </a:r>
19        <a:endParaRPr />  optional, if present only listed once.
20    </a:p>
21  </p:txBody>
```

Element	Purpose	Description
a:bodyPr	Body Properties	Describes text anchor points, shape autofit, number of columns, text warping, and 3D scenes and lighting effects. See §5.4.3.
a:lstStyle	List Style	Used to define style properties for the paragraph and its nine list levels.

a:p	Single Paragraph	Houses a single paragraph and its corresponding paragraph-level properties. Contained within here are also all the text runs that comprise this paragraph. See §5.4.3.4.
a:pPr	Paragraph Properties	Describes the format and style with which the corresponding paragraph is presented. Some possible settings that can be utilized within this space include, but are not limited to, the following: spacing, margins, and alignment. See §5.4.3.4.
a:r	Single Run	Specifies the existence of a run of text within a paragraph. A run represents the most granular form of text that can be represented in the file format. See §5.4.3.8.
a:rPr	Run Properties	Allows the attachment of properties to the run of text specified by its parent a:r element. These properties include, but are not limited to, the following: underline, strikethrough, and text caps. See §5.4.3.8.
a:t	Actual text	Allows for the storage of the specific text that all these body, paragraph and run level properties are describing. This tag is the most important as it gives context to all the other elements and attributes that have come before it.
a:endParaRPr	Persistent Run Properties	Specifies the properties that are to persist should the user begin to type additional text after this paragraph. This property should only be set when the style that should follow this paragraph is different from the paragraph itself.

### 1    5.4.3      Body Level Properties

2    Schemas represented here: dml-text.xsd

3    In this subclause, we'll explore the sorts of properties that can be attached to the body as a whole. As shown in  
 4    the sample XML above, there are three essential property levels available. The body-level properties are to the  
 5    broadest of these. Note that some of these body-level properties are applied as attributes to the body  
 6    property tag while others are expressed as child elements. The specific method by which each property is  
 7    applied can be found in the schemas listed above.

```

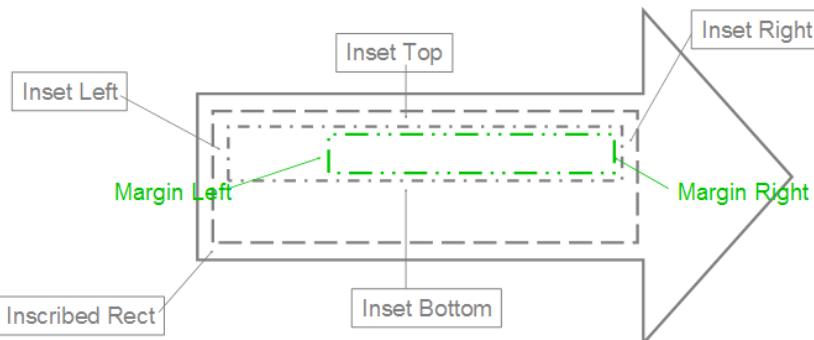
1   <p:txBody>
2     <a:bodyPr /> ← Main element covered in this subclause
3     <a:lstStyle />
4     <a:p>
5       <a:pPr />
6       <a:r>
7         <a:rPr />
8         <a:t>Your text here!</a:t>
9       </a:r>
10      </a:p>
11    </p:txBody>

```

#### 12 5.4.3.1 Setting Up the Text Area

13 Let's start with how a text area might be initially described. This area is the container within which all the child  
 14 text for this body resides. First, it is useful to understand the inset properties; specifically, the top, bottom, left  
 15 and right inset properties that are also known as internal margins for the text body. The anchor attribute  
 16 allows us to specify where the text area should be anchored within its bounding rectangle.

17 An illustration of this bounding rectangle is highlighted below by the inner green box. Notice here that the  
 18 bounding rectangle is anchored to the right.



19  
 20 Here's how the text will appear inside. Attribute AutoFit allows for three basic scenarios:

- 21 • No AutoFit: The text is allowed to flow outside the container.
- 22 • Normal AutoFit: The text is resized using defined constraints to fit inside the container area. (This is  
 23 used when the text is too large or long to fit in the text container.)
- 24 • Shape AutoFit: The actual text container is resized to contain all the text. (This is the only option that  
 25 can cause the container to have its dimensions changed.)

26 The term *flow* is used to describe the way in which text moves around inside this text area, and to describe  
 27 how each of the body properties affects the text within the text area.

28 One way that text can flow is from one line to the next. This can be done automatically by using the text-  
 29 wrapping attribute. Another way is to use columns. The XML framework allows for the specification of a

1 number of columns into which the text is to be automatically broken. This feature also allows for the specifying  
 2 of the spacing of columns and a right-to-left layout instead of the default left-to-right. Another way that text  
 3 can flow is vertical instead of horizontal. For this, there are many different types of vertical text that can be  
 4 described: from text that appears rotated to text where the characters are truly stacked. The text can even be  
 5 made to flow differently when an East Asian font is specified.

6 When looking at the flow it is useful to discuss the potential for *overflow*. That is, the text must flow outside  
 7 the text area because it is too large to fit inside. For this, there are two common types: vertical and horizontal.  
 8 The vertical overflow can be handled in three ways:

- 9     • **overflow**: This allows the text to flow outside the text area.



- 10    • **ellipsis**: This crops the text that overflows and adds "..." to denote that there is hidden text.



- 12    • **clip**: This crops the text just as ellipsis but does not insert "...", so the user has no indication that there  
 13    is hidden text.



- 15    • Horizontal overflow works exactly like vertical, but with only two options: **overflow** and  
 16    **clip**, which both operate as described above.

#### 17    5.4.3.2       Manipulating the Text

18 Let's look at the ways in which the text can be further enhanced at the text body level. Note that the  
 19 properties that follow apply only to the text body as a whole and thus cannot be applied to a specific  
 20 paragraph or run within the text body. These properties are as follows:

21    Text Warping: Text within the text area is made to distort itself according to a predefined shape. This  
 22    shape resides within the bounding box described earlier. This effect is known as *text warping* and has its



23    preset shapes defined further within dml-shapeGeometry.xsd

- 1     • 3D Text: Text can be described with respect to a 3D scene. Using this tag provides three basic options:
- 2         • The text resides within a 3D scene, but as planar text.
- 3         • The text is allowed to reside within the 3D scene and has 3D effects (such as bevel or extrusion) applied to it.
- 4         • The text resides on top of a 3D scene. The 3D scene properties are defined dml-shape3DStyles.xsd and dml-shape3DScene.xsd schemas.
- 5     • Rotated and Upright Text: A particular rotation can be specified that is applied to the text within the text area. Note that this is different from the rotation that is applied to the shape within which the text area resides. If this attribute is not specifically set then the rotation of the container shape is used.
- 6
- 7
- 8
- 9

#### 10    5.4.3.3       Backwards and Forwards Compatibility

11   The following areas are of interest when considering support for both past design and future innovation.

- 12     • From WordArt: This is specific to dealing with previous WordArt text. Now that text is described as simply a shape, there is no need for a WordArt-specific description. There is, however, the need to identify which pieces of text were from the old WordArt styles in case there is the need to write them back out in their old format.
- 13     • Future Extensions: The ability for future extensions has been provided to the body property tag via the ext tag. This can be used the widest way possible as it is a complex type and can thus describe the most complex future properties. Note that each of the schema subclauses below have their own ext tag.
- 14
- 15
- 16
- 17
- 18
- 19

#### 20    5.4.3.4       Paragraph-Level Properties

21   Schemas represented here: dml-textParagraph.xsd, dml-textBullet.xsd.

22   In this subclause, we will explore what sorts of properties can be attached to a paragraph as a whole. Paragraph-level properties allow for a more granular description of the text than the properties of the body tag described earlier. Keep in mind that the properties that can be applied at this level are not duplicates of the body or run levels, but unique only to the paragraph element. Once again, it should be noted that some of these paragraph-level properties are applied as attributes to the paragraph property tag while others are expressed as child elements. The specific method by which each property is applied can be found in the schemas listed above.

29    <p:txBody>

```

1      <a:bodyPr />
2      <a:lstStyle />
3      <a:p>
4          <a:pPr /> ← Main element covered in this subclause
5          <a:r>
6              <a:rPr />
7              <a:t>Your text here!</a:t>
8          </a:r>
9      </a:p>
10     </p:txBody>

```

#### 11 5.4.3.5 Spacing, Alignment, and Direction

12 The XML file format allows for the specifying of spacing both between lines in the form of line spacing, and also  
 13 outside the paragraph via margins and special before/after spacing. In addition to this, there is also the ability  
 14 to specify indent spacing for the beginning of the paragraph.

15 The standard alignment options include left-aligned, right-aligned, centered, justified, and distributed. Justified  
 16 alignment causes each line of text to be stretched out to a certain point. To ensure that short lines remain  
 17 readable, they are not stretched. Distributed alignment is quite similar, but stretches every line, regardless of  
 18 that line's length.

19 Text direction is specified as either left-to-right (the default) or right-to-left using the specific rtl tag.

#### 20 5.4.3.6 Tabs and Line Breaks

21 When the default tabs are not sufficient for the paragraph in question there is the option of including custom  
 22 tab stops in the XML file format. The information required for this is both a default tab size attribute and a full  
 23 tab stop list showing all tab stop positions that apply to this paragraph. Keep in mind that if tab stops are not  
 24 explicitly stated in the file format that the business logic of the application must use its own default positions if  
 25 tabs are needed.

26 Line break is a tag that informs the application as to whether it should break up a string of text onto multiple  
 27 lines based on Latin grammar rules or East Asian grammar rules. The East Asian option uses the Kinsoku  
 28 settings to determine whether a word is allowed to begin or end a line of text.

#### 29 5.4.3.7 Adding Bullets

30 Bullets are specified per paragraph, so bullets can be mixed and matched within a single text body to appear as  
 31 a coherent text group. The types of bullets available are:

- 32 • Character Bullets: Uses a font character to denote a bullet and can be set to appear in any size  
 33 (percentage of text), color (all available including theme colors), and font. *The properties are Bullet  
 34 Color, Bullet Size, Bullet Font Typeface, Bullet Character (represents the actual bullet)*

g Bullet 1

g Bullet 2

g Bullet 3

•

- Auto-Numbered Bullets: Uses the application logic to assign a series of numbers/characters to a specific bulleted item using just a bullet scheme and a starting number. (When a starting number is used, all bulleted paragraphs listed after the start number are automatically numbered based on this last known start number. The scope of this auto-numbering is only within its current text body, no start at number would ever carry over to a different text body.) The properties are Start At number, Bullet Scheme (letters, roman numerals, etc.), Bullet Color, Bullet Size, and Bullet Font Typeface.

1. Bullet 1

1. Bullet 2

2. Bullet 3

•

- Blip Bullets: Uses a picture to denote a bulleted item. The only additional property available with this type of applied bullet is the size (percentage of text). If the graphic is not in the applications standard set of graphics then the attached graphic is converted to a PNG format, placed in the document container and is given a relationship id that is used later to reference the image. The properties are Embed id (corresponds to a bullet graphic) and Bullet Size.

 Bullet 1

 Bullet 2

 Bullet 3

•

#### 5.4.3.8 Run- and Character-Level Properties

Schemas represented here: dml-textRun.xsd, dml-textCharacter.xsd

In this subclause, we'll explore the most granular text properties available in this XML framework, namely those described at the text run and character level. This level is usually the level in which text is broken up into differently formatted parts, because the most commonly used text properties almost all reside at this level. This allows for some very detailed formatting to be represented. Again, it should be noted that for consistency that some of these run and character level properties are applied as attributes to the run property tag while others are expressed as child elements. The specific method by which each property is applied can be found in the schemas listed above.

```

1   <p:txBody>
2     <a:bodyPr />
3     <a:lstStyle />
4     <a:p>
5       <a:pPr />
6       <a:r>
7         <a:rPr />    ← Main element covered in this subclause
8         <a:t>Your text here!</a:t>
9       </a:r>
10      </a:p>
11    </p:txBody>

```

#### 12 5.4.3.9 Visual Properties

13 When looking to format a run of text the first property that one might need to specify would be the font  
 14 typeface. The XML file format allows for the specification of not only Latin Fonts but also East Asian, Complex  
 15 Script, and Symbol fonts as well. These four font buckets give the application additional information that is  
 16 used to layout text in a manner fitting for the specific font. Along with the actual font being used, comes the  
 17 size of the font. To specify this simply use the sz attribute and along with a value that is 1/100th of the size in  
 18 points.

19 Other common formatting properties allowed in the XML framework are bold, italic, underline and  
 20 strikethrough. The use of both the bold and italic properties is simply via a Boolean value of 0 or 1. The usage  
 21 of the underline and strikethrough, however, allow a more specific selection to be made. There are 17 values  
 22 for underline, which range from a single line to wavy double lines. In addition to specifying the style of  
 23 underline that is to be used, the framework can also specify fill properties for the underline. These are solid  
 24 color, multi-color gradient, and texture fill. For strikethrough, there are two options: single and double strike  
 25 through.

26 When standard formatting isn't adequate, more complex effects can be defined for a specific run of text. The  
 27 basic breakdown for these is line properties, fill properties and effect properties. Encapsulated within each of  
 28 these areas is a wide range of customizable effects. A quick look at line properties, for example, reveals the  
 29 ability to specify a color, gradient, or pattern fill, along with a width and style applied. Along these lines fill  
 30 properties allows for transparent fill, solid fill, gradient fill, texture fill and even picture fill. While these  
 31 features alone give the XML file format plenty of robustness in describing text, other features are also  
 32 available. Because text is treated the same as a shape, a run of text can have virtually all shape effects applied  
 33 to it just as if it were a shape. These effects include shadow, glow, and reflection, and are placed in an effect  
 34 list under the run properties tag. An example of what these lines, fills and effects may look like is provided  
 35 below. More information on these effects can be found in either dml-shapeLineProperties.xsd or dml-  
 36 shapeEffects.xsd.

```

1   <a:rPr>
2     <a:ln>
3       <a:solidFill ... />    ← Line properties here
4     </a:ln>
5     <a:gradFill>
6       <a:gsLst ... />      ← Fill properties here
7     </a:gradFill>
8     <a:effectLst>
9       <a:reflection ... />  ← Effect properties here
10    </a:effectLst>
11  </a:rPr>

```

12 A few additional properties are worth noting:

- Minimum kerning size: This specifies the smallest font size at which kerning still occurs. When no tag is present for this the default value is 0, allowing kerning at any text size.
- Spacing between characters: The units here are the same as are used for font size. Along the lines of specifying horizontal spacing, vertical spacing can be specified via the baseline tag. This is typically used for subscript and superscript text and is specified in the same units as font size.
- Capitalization and Normalize: Capitalization sets the case of the character to either all small caps or all large caps. For this property there are only these two settings aside from the "none" setting at which point this property is ignored. Normalize height takes all shorter characters and adjusts their height up so that they are the same as taller characters. This property is set via a Boolean value.

#### 22 5.4.3.10 Properties for Interactivity

23 Hyperlinks: The XML file format allows for the inputting of hyperlinks that are activated by either click or mouse over. These two tags are `HyperlinkClick` and `HyperlinkMouseOver`, respectfully. They both allow for the specifying of a link to another resource very much like those found on a common website.

26 Spelling and Smart Tags: Although spelling is very much an application-specific part of text editing and is most likely to be done within the application itself there are a few ways that spelling settings and preferences can be persisted within the file format. One way is through the spelling error bit, which simply saves whether there is a known spelling mistake. The next is the spelling dirty bit. This gets set whenever the user has entered new text and the application has not had a chance to check for spelling errors on this piece of text. Lastly, in this realm we actually have a user preference of no proofing that is persisted for the next time a document is opened. This allows the user to specify a word that they do not want to have checked for spelling. Along with spell-checking comes the notion of smart tags which must be checked for just like spelling mistakes. For this there are two related tags. The first is the smart tag clean, which allows for a boolean value to be set determining if this portion of text has been checked for the presence of new smart tags. The next is the actual smart tag id. Once a piece of text has been determined to be a smart tag then a smart tag id is assigned which points to the actual smart tag information.

1    **5.4.3.11      International Language Support**

2    There exists the notion of the language id, which is simply a value that assists the application in laying out the  
 3    text. The tags that help with this are the language id tag and the alternate language id tag. Together, these  
 4    allow the file format to be robust and handle multiple languages for a single run of text. In addition, there is  
 5    also the kumimoji tag, which aids with the layout of East Asian text by specifying whether numbers appear  
 6    vertically with text (default) or horizontally. An illustration of a run of text with kumimoji applied is provided  
 7    below.

E  
C  
M  
A  
  
T  
C  
45

8

9    **5.5      Tables**

10    **5.5.1      Introduction**

11    This document provides a high-level overview of the content described in the schemas dml-table.xsd and dml-  
 12    tableStyle.xsd.

13    This aspect of DrawingML deals with the definition of a table and the associated styling information. The first  
 14    part describes the table styles aspect, while the second part describes the definition of a table within  
 15    DrawingML. The above-mentioned schemas fall into the following groupings:

<b>Table Styles</b>	<b>Table Definition</b>
dml-tableStyle.xsd	dml-table.xsd

16    **5.5.2      Table Styles**

17    Table styles are responsible for the rapid formatting that can be applied to a table. This rapid formatting takes  
 18    different things into account, such as if the first row or last row should be emphasized, or if there is some type  
 19    of banding present on the table. All of these different types of formatting can be defined within a table style.  
 20    An example of different table styles in use on the same table can be seen in figure 1.

The figure shows three tables side-by-side, each with four rows of data. The columns are labeled 'Red', 'Blue', and 'Yellow'. The data values are: Row 1 (Header): Red 21.5, Blue 18.3, Yellow 4.5; Row 2: Red 17.4, Blue 3.6, Yellow 2.2; Row 3: Red 9.1, Blue 19.8, Yellow 7.9; Row 4: Red 12.2, Blue 13.4, Yellow 12.1.

	Red	Blue	Yellow
1 <sup>st</sup> Qtr	21.5	18.3	4.5
2 <sup>nd</sup> Qtr	17.4	3.6	2.2
3 <sup>rd</sup> Qtr	9.1	19.8	7.9
4 <sup>th</sup> Qtr	12.2	13.4	12.1

	Red	Blue	Yellow
1 <sup>st</sup> Qtr	21.5	18.3	4.5
2 <sup>nd</sup> Qtr	17.4	3.6	2.2
3 <sup>rd</sup> Qtr	9.1	19.8	7.9
4 <sup>th</sup> Qtr	12.2	13.4	12.1

	Red	Blue	Yellow
1 <sup>st</sup> Qtr	21.5	18.3	4.5
2 <sup>nd</sup> Qtr	17.4	3.6	2.2
3 <sup>rd</sup> Qtr	9.1	19.8	7.9
4 <sup>th</sup> Qtr	12.2	13.4	12.1

1

2 Figure 6: Different table styles in use.

3 The application of a table style to a table formats the table in its entirety. A number of complex types make up  
4 a table style. The pieces of a table style are discussed first, before the table style itself is defined.5 

### 5.5.2.1 Cell 3-D

6 The complex type, CT\_Cell3D, defines all of the 3-D properties that an individual cell can hold. In the case of a  
7 table, these 3-D properties can be a bevel along with a material and a light rig for the cell. (More explanation  
8 of these three pieces of a CT\_Cell3D can be found in §5.6.) These properties are applied on a per-cell basis,  
9 rather than to the table as a whole. A CT\_Cell3D is defined in the following manner:

```

10   <xsd:complexType name="CT_Cell3D">
11     <xsd:sequence>
12       <xsd:element name="bevel" type="CT_Bevel" minOccurs="1"
13         maxOccurs="1" />
14       <xsd:element name="lightRig" type="CT_LightRig" minOccurs="0"
15         maxOccurs="1" />
16       <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
17         minOccurs="0" maxOccurs="1" />
18     </xsd:sequence>
19     <xsd:attribute name="prstMaterial" type="ST_PresetMaterialType"
20       use="optional" default="plastic" />
21   </xsd:complexType>
```

22 This complex type also holds a CT\_OfficeArtExtensionList. This complex type is used for future extensibility  
23 and will be seen elsewhere throughout the tables area.

### 5.5.2.2 Themeable Styles

Three groups and a complex type account for style pieces that can be themed. These themed aspects either pull from the style matrix, or they define an actual fill or effect for example. If they pull their style from the matrix, then an update to the document theme will also update the particular style dynamically. The three groups consist of the following groups:

```

6   <xsd:group name="EG_ThemeableFillStyle">
7     <xsd:choice>
8       <xsd:element name="fill" type="CT_FillProperties" minOccurs="1"
9         maxOccurs="1" />
10      <xsd:element name="fillRef" type="CT_StyleMatrixReference"
11        minOccurs="1" maxOccurs="1" />
12    </xsd:choice>
13  </xsd:group>
14  <xsd:group name="EG_ThemeableEffectStyle">
15    <xsd:choice>
16      <xsd:element name="effect" type="CT_EffectProperties"
17        minOccurs="1" maxOccurs="1" />
18      <xsd:element name="effectRef" type="CT_StyleMatrixReference"
19        minOccurs="1" maxOccurs="1" />
20    </xsd:choice>
21  </xsd:group>
22  <xsd:group name="EG_ThemeableFontStyles">
23    <xsd:choice>
24      <xsd:element name="font" type="CT_FontCollection" minOccurs="1"
25        maxOccurs="1" />
26      <xsd:element name="fontRef" type="CT_FontReference" minOccurs="1"
27        maxOccurs="1" />
28    </xsd:choice>
29  </xsd:group>
```

The three groups above all give a choice between using a themed style or defining the style themselves. The last type in this group is a complex type used to perform the same task as the above three, only it deals with the lines in the table. The complex type, CT\_ThemeableLineStyle, is defined as:

```

33  <xsd:complexType name="CT_ThemeableLineStyle">
34    <xsd:choice>
35      <xsd:element name="ln" type="CT_LineProperties" minOccurs="1"
36        maxOccurs="1" />
37      <xsd:element name="lnRef" type="CT_StyleMatrixReference"
38        minOccurs="1" maxOccurs="1" />
39    </xsd:choice>
40  </xsd:complexType>
```

1    5.5.2.3        On/Off Property Definition

2    The simple type, ST\_OnOffStyleType, defines a type with values of on, off, or default. A value of default  
 3    indicates that parent settings should be used. Thus, for a themed property, default indicates that the theme  
 4    properties should be followed. For an unthemed property, default means that the parent setting in the  
 5    property inheritance chain should be followed.

6    5.5.2.4        Text Properties

7    The complex type, CT\_TableStyleTextStyle, defines the table text properties that can be styled. The text  
 8    properties contains a reference to a themeable font style along with bold and italic being enabled or disabled.  
 9    The CT\_TableStyleTextStyle is defined in the following manner:

```
10   <xsd:complexType name="CT_TableStyleTextStyle">
11     <xsd:sequence>
12       <xsd:group ref="EG_ThemeableFontStyles" minOccurs="0"
13         maxOccurs="1" />
14       <xsd:group ref="EG_ColorChoice" minOccurs="0" maxOccurs="1" />
15         <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
16           minOccurs="0" maxOccurs="1" />
17     </xsd:sequence>
18     <xsd:attribute name="b" type="ST_OnOffStyleType" use="optional"
19       default="def" />
20     <xsd:attribute name="i" type="ST_OnOffStyleType" use="optional"
21       default="def" />
22   </xsd:complexType>
```

23    5.5.2.5        Cell Border Properties

24    The complex type, CT\_TableCellBorderStyle, defines the properties of the borders that can be styled in a table.  
 25    The border styles can be applied to the following different types of borders in a table:

- 26    • left – left border
- 27    • right – right border
- 28    • top – top border
- 29    • bottom – bottom border
- 30    • insideH – inner horizontal borders
- 31    • insideV – inner vertical borders
- 32    • tl2br – diagonal border from top left corner to bottom right corner
- 33    • tr2bl – diagonal border from top right corner to bottom left corner

34    The complex type is defined in the following manner:

```

1   <xsd:complexType name="CT_TableCellBorderStyle">
2     <xsd:sequence>
3       <xsd:element name="left" type="CT_ThemeableLineStyle"
4         minOccurs="0" maxOccurs="1" />
5       <xsd:element name="right" type="CT_ThemeableLineStyle"
6         minOccurs="0" maxOccurs="1" />
7       <xsd:element name="top" type="CT_ThemeableLineStyle"
8         minOccurs="0" maxOccurs="1" />
9       <xsd:element name="bottom" type="CT_ThemeableLineStyle"
10      minOccurs="0" maxOccurs="1" />
11      <xsd:element name="insideH" type="CT_ThemeableLineStyle"
12        minOccurs="0" maxOccurs="1" />
13      <xsd:element name="insideV" type="CT_ThemeableLineStyle"
14        minOccurs="0" maxOccurs="1" />
15      <xsd:element name="tl2br" type="CT_ThemeableLineStyle"
16        minOccurs="0" maxOccurs="1" />
17      <xsd:element name="tr2bl" type="CT_ThemeableLineStyle"
18        minOccurs="0" maxOccurs="1" />
19      <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
20        minOccurs="0" maxOccurs="1" />
21    </xsd:sequence>
22  </xsd:complexType>

```

### 5.5.2.6 Cell Style Properties

The complex type, CT\_TableStyleCellStyle, contains the definition for cell properties that can be styled. Within this complex type are held the border style, cell fill style, and the cell 3-D. The complex type is defined in the following manner:

```

27   <xsd:complexType name="CT_TableStyleCellStyle">
28     <xsd:sequence>
29       <xsd:element name="tcBdr" type="CT_TableCellBorderStyle"
30         minOccurs="0" maxOccurs="1" />
31       <xsd:group ref="EG_ThemeableFillStyle" minOccurs="0"
32         maxOccurs="1" />
33       <xsd:element name="cell3D" type="CT_Cell3D" minOccurs="0"
34         maxOccurs="1" />
35     </xsd:sequence>
36   </xsd:complexType>

```

### 5.5.2.7 Table Background Style

The complex type, CT\_TableBackgroundStyle, defines the style elements associated with the background of the table. The table background style can contain a fill and effect. The complex type is defined in the following manner:

```

1   <xsd:complexType name="CT_TableBackgroundStyle">
2     <xsd:sequence>
3       <xsd:group ref="EG_ThemeableFillStyle" minOccurs="0"
4         maxOccurs="1" />
5       <xsd:group ref="EG_ThemeableEffectStyle" minOccurs="0"
6         maxOccurs="1" />
7     </xsd:sequence>
8   </xsd:complexType>
```

### 9    5.5.2.8        Table Part Style

10   The complex type, CT\_TablePartStyle, defines a structure for holding the style information for a single part of  
 11   the table. The table is broken up in 13 different parts which will be explained in the next subclause of this  
 12   document. A table part contains a text style and a cell style and is defined in the following manner:

```

13   <xsd:complexType name="CT_TablePartStyle">
14     <xsd:sequence>
15       <xsd:element name="tcTxStyle" type="CT_TableStyleTextStyle"
16         minOccurs="0" maxOccurs="1" />
17       <xsd:element name="tcStyle" type="CT_TableStyleCellStyle"
18         minOccurs="0" maxOccurs="1" />
19     </xsd:sequence>
20   </xsd:complexType>
```

### 21   5.5.2.9        Table Style

22   The complex type, CT\_TableStyle, defines the actual table style. There are thirteen different parts (outside of  
 23   the table background) that can be defined in a table style. These parts work together to define the styling for a  
 24   table, given the six combinations of on/off states for the first row, first column, last row, last column, row  
 25   banding, and column banding options. The different parts of a table that make up a table style are:

- 26     • tableBg – table background (this is not a CT\_TablePartStyle)
- 27     • wholeTable – formatting for the entire table
- 28     • band1Horizontal – applied when row banding is enabled, this is the first row style, which alternates  
 29       with band2Horizontal
- 30     • band2Horizontal – applied when row banding is enabled, this is the second row style, which alternates  
 31       with band1Horizontal
- 32     • band1Vertical – applied when column banding is enabled, this is the first column style, which  
 33       alternates with band2Vertical
- 34     • band2Vertical – applied when column banding is enabled, this is the second column style, which  
 35       alternates with band1Vertical
- 36     • lastCol – formatting applied to the last column when last column formatting is enabled
- 37     • firstCol – formatting applied to the first column when first column formatting is enabled
- 38     • lastRow – formatting applied to the last row when last row formatting is enabled
- 39     • firstRow – formatting applied to the first row when first row formatting is enabled

- 1     • seCell – formatting applied to the cell in the southeast corner of the table when last column and last
- 2       row are enabled
- 3     • swCell – formatting applied to the cell in the southwest corner of the table when first column and last
- 4       row are enabled
- 5     • neCell – formatting applied to the cell in the northeast corner of the table when the last column and
- 6       first row are enabled
- 7     • nwCell – formatting applied to the cell in the northwest corner of the table when the first column and
- 8       first row are enabled

9     The table style is defined in the following manner:

```

10      <xsd:complexType name="CT_TableStyle">
11          <xsd:sequence>
12              <xsd:element name="tblBg" type="CT_TableBackgroundStyle"
13                  minOccurs="0" maxOccurs="1" />
14              <xsd:element name="wholeTbl" type="CT_TablePartStyle"
15                  minOccurs="0" maxOccurs="1" />
16              <xsd:element name="band1H" type="CT_TablePartStyle" minOccurs="0"
17                  maxOccurs="1" />
18              <xsd:element name="band2H" type="CT_TablePartStyle" minOccurs="0"
19                  maxOccurs="1" />
20              <xsd:element name="band1V" type="CT_TablePartStyle" minOccurs="0"
21                  maxOccurs="1" />
22              <xsd:element name="band2V" type="CT_TablePartStyle" minOccurs="0"
23                  maxOccurs="1" />
24              <xsd:element name="lastCol" type="CT_TablePartStyle"
25                  minOccurs="0" maxOccurs="1" />
26              <xsd:element name="firstCol" type="CT_TablePartStyle"
27                  minOccurs="0" maxOccurs="1" />
28              <xsd:element name="lastRow" type="CT_TablePartStyle"
29                  minOccurs="0" maxOccurs="1" />
30              <xsd:element name="seCell" type="CT_TablePartStyle" minOccurs="0"
31                  maxOccurs="1" />
32              <xsd:element name="swCell" type="CT_TablePartStyle" minOccurs="0"
33                  maxOccurs="1" />
34              <xsd:element name="firstRow" type="CT_TablePartStyle"
35                  minOccurs="0" maxOccurs="1" />
36              <xsd:element name="neCell" type="CT_TablePartStyle" minOccurs="0"
37                  maxOccurs="1" />
38              <xsd:element name="nwCell" type="CT_TablePartStyle" minOccurs="0"
39                  maxOccurs="1" />
```

```

1   <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
2       minOccurs="0" maxOccurs="1" />
3   </xsd:sequence>
4   <xsd:attribute name="styleId" type="ST_Guid" use="required" />
5   <xsd:attribute name="styleName" type="xsd:string" use="required" />
6   </xsd:complexType>
```

7 Also contained within the table style is an ID and a name. The name shows up as the name for the table style  
8 and the ID is the unique id (GUID) that is associated with the table style.

#### 9 5.5.2.10 Table Style List

10 The final complex type dealing with table styles is simply a list of table styles. Also contained in this list is the  
11 default style which is applied to the table when the a default is to be used. The complex type,  
12 CT\_TableStyleList, is defined in the following manner:

```

13  <xsd:complexType name="CT_TableStyleList">
14      <xsd:sequence>
15          <xsd:element name="tblStyle" type="CT_TableStyle" minOccurs="0"
16              maxOccurs="unbounded" />
17      </xsd:sequence>
18      <xsd:attribute name="def" type="ST_Guid" use="required" />
19  </xsd:complexType>
```

#### 20 5.5.3 Table Definition

21 In this subclause, the focus is on the actual definition of a table and the data contained within the table. There  
22 are not as many complex types in this subclause as with the table style subclause, but they are organized in the  
23 same way as the table style section.

#### 24 5.5.3.1 Cell Properties

25 The complex type, CT\_TableCellProperties, holds all the information that deals with the properties of a given  
26 cell. The cell properties contain a section for the different line properties (ln\*), the cell fill properties, the 3-D  
27 properties, cell margin information (mar\*), anchoring information (anchor and anchorCtr), a vertical text type,  
28 and finally an attribute which defines the behavior of horizontal text overflow (horzOverflow). As with many  
29 other types defined in this document, CT\_TableCellProperties contains an element reserved for future  
30 extensibility. The complex type is defined in the following manner:

```

31  <xsd:complexType name="CT_TableCellProperties">
32      <xsd:sequence>
33          <xsd:element name="lnL" type="CT_LineProperties" minOccurs="0"
34              maxOccurs="1" />
35          <xsd:element name="lnR" type="CT_LineProperties" minOccurs="0"
36              maxOccurs="1" />
37          <xsd:element name="lnT" type="CT_LineProperties" minOccurs="0"
38              maxOccurs="1" />
```

```

1  <xsd:element name="lnB" type="CT_LineProperties" minOccurs="0"
2    maxOccurs="1" />
3  <xsd:element name="lnTlToBr" type="CT_LineProperties"
4    minOccurs="0" maxOccurs="1" />
5  <xsd:element name="lnBlToTr" type="CT_LineProperties"
6    minOccurs="0" maxOccurs="1" />
7  <xsd:element name="cell3D" type="CT_Cell3D" minOccurs="0"
8    maxOccurs="1" />
9  <xsd:group ref="EG_FillProperties" minOccurs="0" maxOccurs="1" />
10 <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
11   minOccurs="0" maxOccurs="1" />
12 </xsd:sequence>
13 <xsd:attribute name="marL" type="ST_Coordinate32" use="optional"
14   default="91440" />
15 <xsd:attribute name="marR" type="ST_Coordinate32" use="optional"
16   default="91440" />
17 <xsd:attribute name="marT" type="ST_Coordinate32" use="optional"
18   default="45720" />
19 <xsd:attribute name="marB" type="ST_Coordinate32" use="optional"
20   default="45720" />
21 <xsd:attribute name="vert" type="ST_TextVerticalType" use="optional"
22   default="horz" />
23 <xsd:attribute name="anchor" type="ST_TextAnchoringType"
24   use="optional" default="t" />
25 <xsd:attribute name="anchorCtr" type="xsd:boolean" use="optional"
26   default="false" />
27 <xsd:attribute name="horzOverflow" type="ST_TextHorzOverflowType"
28   use="optional" default="clip" />
29 </xsd:complexType>
```

### 5.5.3.2 Column

The complex type, CT\_TableCol, defines a table column element. The table column element simply holds the width for a given column in a table along with an element reserved for future extensibility. The complex type is defined as:

```

34 <xsd:complexType name="CT_TableCol">
35   <xsd:sequence>
36     <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
37       minOccurs="0" maxOccurs="1" />
38   </xsd:sequence>
39   <xsd:attribute name="w" type="ST_Coordinate" use="required" />
40 </xsd:complexType>
```

1    5.5.3.3        Table Grid

2    The complex type, CT\_TableGrid, defines a list of table column elements, or rather CT\_TableCol complex types.  
 3    The CT\_TableGrid should contain a CT\_TableCol for each column in the table and it is defined in the following  
 4    manner:

```
5      <xsd:complexType name="CT_TableGrid">
6          <xsd:sequence>
7              <xsd:element name="gridCol" type="CT_TableCol" minOccurs="0"
8                  maxOccurs="unbounded" />
9          </xsd:sequence>
10     </xsd:complexType>
```

11    5.5.3.4        Cell

12    The complex type, CT\_TableCell, defines a cell in a table. Within this complex type lies a text body which holds  
 13    the data of the cell along with any formatting applied to the text. This complex type also holds a table cell  
 14    property complex type which has already been defined. The rowSpan and gridSpan attributes are available  
 15    along with hMerge and vMerge attributes. The hMerge and vMerge attributes define if the current cell is  
 16    supposed to be merged with the previous cell horizontally or vertically. This is how the table is parsed and  
 17    created. The complex type, CT\_TableCell, is defined as:

```
18     <xsd:complexType name="CT_TableCell">
19         <xsd:sequence>
20             <xsd:element name="txBody" type="CT_TextBody" minOccurs="0"
21                 maxOccurs="1" />
22             <xsd:element name="tcPr" type="CT_TableCellProperties"
23                 minOccurs="0" maxOccurs="1" />
24             <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
25                 minOccurs="0" maxOccurs="1" />
26         </xsd:sequence>
27         <xsd:attribute name="rowSpan" type="xsd:int" use="optional"
28             default="1" />
29         <xsd:attribute name="gridSpan" type="xsd:int" use="optional"
30             default="1" />
31         <xsd:attribute name="hMerge" type="xsd:boolean" use="optional"
32             default="false" />
33         <xsd:attribute name="vMerge" type="xsd:boolean" use="optional"
34             default="false" />
35     </xsd:complexType>
```

36    5.5.3.5        Row

37    The complex type, CT\_TableRow, defines a table row. This complex type is somewhat more complex than the  
 38    similar table column complex type in that it holds a sequence of CT\_TableCell structures along with a height for  
 39    the row. The complex type is defined in the following way:

```

1   <xsd:complexType name="CT_TableRow">
2     <xsd:sequence>
3       <xsd:element name="tc" type="CT_TableCell" minOccurs="0"
4         maxOccurs="unbounded" />
5       <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
6         minOccurs="0" maxOccurs="1" />
7     </xsd:sequence>
8     <xsd:attribute name="h" type="ST_Coordinate" use="required" />
9   </xsd:complexType>
```

### 5.5.3.6 Table Properties

The complex type, CT\_TableProperties, defines the properties for a table as a whole. Within this complex type is a definition for a table style that is currently applied to the table, or the GUID for the built in table style that is applied to the table. Also in this complex type are right-to-left settings, the effects applied to the table (shadow, reflection, etc), background fill information, and the states for the different on/off table style options. The complex type is defined as:

```

16  <xsd:complexType name="CT_TableProperties" >
17    <xsd:sequence>
18      <xsd:group ref="EG_FillProperties" minOccurs="0" maxOccurs="1" />
19      <xsd:group ref="EG_EffectProperties" minOccurs="0"
20        maxOccurs="1" />
21      <xsd:choice oxsd:cname="TableStyleOrLink" minOccurs="0"
22        maxOccurs="1" >
23        <xsd:element name="tableStyle" type="CT_TableStyle" />
24        <xsd:element name="tableStyleId" type="ST_Guid" />
25      </xsd:choice>
26      <xsd:element name="extLst" type="CT_OfficeArtExtensionList"
27        minOccurs="0" maxOccurs="1" />
28    </xsd:sequence>
29    <xsd:attribute name="rtl" type="xsd:boolean" use="optional"
30      default="false" />
31    <xsd:attribute name="firstRow" type="xsd:boolean" use="optional"
32      default="false" />
33    <xsd:attribute name="firstCol" type="xsd:boolean" use="optional"
34      default="false" />
35    <xsd:attribute name="lastRow" type="xsd:boolean" use="optional"
36      default="false" />
37    <xsd:attribute name="lastCol" type="xsd:boolean" use="optional"
38      default="false" />
39    <xsd:attribute name="bandRow" type="xsd:boolean" use="optional"
40      default="false" />
```

```

1      <xsd:attribute name="bandCol" oxsdatatype="xsd:boolean" use="optional"
2          default="false" />
3      </xsd:complexType>
```

#### 4 5.5.3.7 Table

5 The final complex type, CT\_Table, is the root element for a table. This complex type holds all the information  
6 that is needed to create a table within DrawingML. Within CT\_Table are the table properties, a table grid, and  
7 a table row. A CT\_Table is defined in the following manner:

```

8      <xsd:complexType name="CT_Table">
9          <xsd:sequence>
10             <xsd:element name="tblPr" type="CT_TableProperties" minOccurs="0"
11                 maxOccurs="1" />
12             <xsd:element name="tblGrid" type="CT_TableGrid" minOccurs="1"
13                 maxOccurs="1" />
14             <xsd:element name="tr" type="CT_TableRow" minOccurs="0"
15                 maxOccurs="unbounded" />
16         </xsd:sequence>
17     </xsd:complexType>
```

### 18 5.6 3D Aspects

#### 19 5.6.1 Introduction

20 This subclause provides a high-level overview of the content described in the following schemas: dml-  
21 shape3DStyles.xsd, dml-shape3DScene.xsd, dml-shape3DScenePlane.xsd, dml-shape3DLighting.xsd, and dml-  
22 shape3DCamera.xsd.

23 This aspect of DrawingML deals mainly with the 3-D aspects, and can be broken down into two topics: 3-  
24 D properties associated with an object, and the styling information associated with an object. The above-  
25 mentioned schemas fall into the grouping in the following way:

3-D	Styles
dml-shape3DScene.xsd dml-shape3DScenePlane.xsd dml-shape3DLighting.xsd dml-shape3DCamera.xsd	dml-shape3DStyles.xsd

#### 26 5.6.2 3-D

27 Here we'll explain the 3-D definitions contained in DrawingML. The goal here is to define a 3-D scene so that  
28 lighting calculations can be made on the geometry within the scene.

1    5.6.2.1        3-D Scene

2    Every 3-D scene consists of a camera, a light, and a backdrop, that define the associated properties of the  
 3    scene. The complex type, CT\_Scene3D, defines the scene as follows:

```

4      <xsd:complexType name="CT_Scene3D" oxsd:cname="Scene3D">
5          <xsd:sequence>
6              <xsd:element name="camera" type="CT_Camera" minOccurs="1"
7                  maxOccurs="1" />
8              <xsd:element name="lightRig" type="CT_LightRig" minOccurs="1"
9                  maxOccurs="1"/>
10             <xsd:element name="backdrop" type="CT_Backdrop" minOccurs="0"
11                 maxOccurs="1" />
12             <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
13                 maxOccurs="1" />
14         </xsd:sequence>
15     </xsd:complexType>
```

16   As was stated above, the complex type, CT\_Scene3D, contains a camera, a set of lights (the light rig), and a  
 17   backdrop. Those familiar with 3-D rendering techniques understand the usage of a camera and set of lights, or  
 18   light rig. The backdrop, however, is a special structure (which will be defined below) that allows for a special  
 19   plane to render certain effects that need to be rendered together in a single plane. The final element of a  
 20   CT\_Scene3D is the ext element. This is a DrawingML structure used for future extensibility. This element will  
 21   be seen in other complex types dealing with the 3-D scene as well.

22   5.6.2.2        Camera

23   The complex type, CT\_Camera, defines a camera within the 3-D scene. A camera is based on a preset, with an  
 24   optional rotation, field-of-view, and zoom, which all act as overrides for the preset values. A camera is defined  
 25   in the following way:

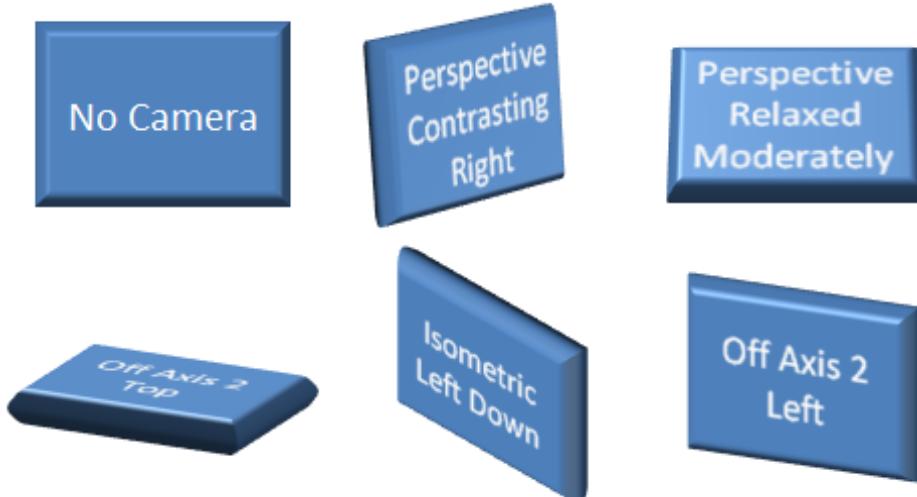
```

26      <xsd:complexType name="CT_Camera">
27          <xsd:sequence>
28              <xsd:element name="rot" type="CT_SphereCoords" minOccurs="0"
29                  maxOccurs="1" oxsd:dataStructure="optional" />
30          </xsd:sequence>
31          <xsd:attribute name="prst" type="ST_PresetCameraType" use="required"
32              />
33          <xsd:attribute name="fov" type="ST_FOVAngle" use="optional" />
34          <xsd:attribute name="zoom" type="ST_PositivePercentage"
35              use="optional" default="100000" />
36      </xsd:complexType>
```

37   The only complex type contained in the camera, CT\_SphereCoords, is a type defined elsewhere within the  
 38   DrawingML. There are three simple types associated with a camera:

- 1     • ST\_FOVAngle (field of view angle), which is a positive angle between 0 and 180 in 60,000th of a  
 2       degree.  
 3     • ST\_PositivePercentage (zoom), which is defined as a percentage in 1,000th of a percent.  
 4     • ST\_PresentCameraType (preset camera)

5     Figure 1 below shows some different presets applied to a shape.



6

7     Figure 7: Different default cameras applied to a shape

8     The available options for ST\_PresetCameraType are as follows:

- 9     legacyObliqueTopLeft  
 10    legacyObliqueTop  
 11    legacyObliqueLeft  
 12    legacyObliqueFront  
 13    legacyObliqueRight  
  
 14    legacyObliqueBottomLeft  
 15    legacyObliqueBottom  
 16    legacyObliqueBottomRight  
 17    legacyPerspectiveTopLeft  
 18    legacyPerspectiveTop  
  
 19    legacyPerspectiveTopRight  
 20    legacyPerspectiveLeft  
 21    legacyPerspectiveFront  
 22    legacyPerspectiveRight  
 23    legacyPerspectiveBottomLeft  
  
 24    legacyPerspectiveBottom  
 25    legacyPerspectiveBottomRight

```
1 orthographicFront
2 isometricTopUp
3 isometricTopDown

4 isometricBottomDown
5 isometricLeftUp
6 isometricLeftDown
7 isometricRightUp
8 isometricRightDown

9 isometricOffAxis1Left
10 isometricOffAxis1Right
11 isometricOffAxis1Top
12 isometricOffAxis2Left
13 isometricOffAxis2Right

14 isometricOffAxis2Top
15 isometricOffAxis3Left
16 isometricOffAxis3Right
17 isometricOffAxis3Bottom
18 isometricOffAxis4Left

19 isometricOffAxis4Right
20 isometricOffAxis4Bottom
21 obliqueTopLeft
22 obliqueTopRight
23 obliqueLeft

24 obliqueRight
25 obliqueBottomLeft
26 obliqueBottom
27 obliqueBottomRight
28 perspectiveFront

29 perspectiveLeft
30 perspectiveRight
31 perspectiveAbove
32 perspectiveBelow
33 perspectiveAboveLeftFacing

34 perspectiveAboveRightFacing
35 perspectiveContrastingRightFacing
36 perspectiveContrastingLeftFacing
37 perspectiveHeroicLeftFacing
38 perspectiveHeroicRightFacing
```

```

1 perspectiveHeroicExtremeLeftFacing
2 perspectiveHeroicExtremeRightFacing
3 perspectiveRelaxed
4 perspectiveRelaxedModerately

```

### 5 5.6.2.3 Light

6 The complex type, CT\_LightRig, defines the lighting of the scene. A light rig consists of a preset direction,  
7 preset rig type, and a rotation that serves as an override for the direction. The complex type is defined as:

```

8 <xsd:complexType name="CT_LightRig">
9   <xsd:sequence>
10    <xsd:element name="rot" type="CT_SphereCoords" minOccurs="0"
11      maxOccurs="1" />
12  </xsd:sequence>
13  <xsd:attribute name="rig" type="ST_LightRigType" use="required" />
14  <xsd:attribute name="dir" type="ST_LightRigDirection" use="required" />
15 </xsd:complexType>
16

```

17 Just as with the camera, the complex type, CT\_SphereCoords, is defined elsewhere in the DrawingML. This  
18 element, however, serves as an override for the default light right direction. Figure 2 below shows some of the  
19 different preset lights applied to a shape.



20

21 Figure 8: Some preset lights applied to a shape.

22 The types of available light rigs are:

```

23 legacyFlat1
24 legacyFlat2
25 legacyFlat3

```

```

1 legacyFlat4
2 legacyNormal1

3 legacyNormal2
4 legacyNormal3
5 legacyNormal4
6 legacyHarsh1
7 legacyHarsh2

8 legacyHarsh3
9 legacyHarsh4
10 threePoint
11 balanced
12 soft

13 harsh
14 flood
15 contrasting
16 morning
17 sunrise

18 sunset
19 chilly
20 freezing
21 flat
22 twoPoint
23 glow
24 brightRoom

25 The types of available present directions are:

26 tl – top left
27 t – top
28 tr – top right
29 l – left

30 r – right
31 bl – bottom left
32 b – bottom
33 br – bottom right

34 5.6.2.4 Backdrop
35 The complex type, CT_Backdrop, defines a unique place in the 3-D scene. The backdrop is a flat 2-D plane that
36 can hold effects, such as shadows, oriented in 3-D space. The points and vectors contained within the
37 backdrop are relative to world space. The complex type is defined as:

```

```

1   <xsd:complexType name="CT_Backdrop">
2     <xsd:sequence>
3       <xsd:element name="anchor" type="CT_Point3D" minOccurs="1"
4         maxOccurs="1" />
5       <xsd:element name="norm" type="CT_Vector3D" minOccurs="1"
6         maxOccurs="1" />
7       <xsd:element name="up" type="CT_Vector3D" minOccurs="1"
8         maxOccurs="1"/>
9       <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
10        maxOccurs="1" />
11     </xsd:sequence>
12   </xsd:complexType>
```

13 All of the complex types defined within this backdrop are defined elsewhere in DrawingML. As with other  
14 complex types, the backdrop also contains an element reserved for future extensibility.

### 15 **5.6.3 Styles**

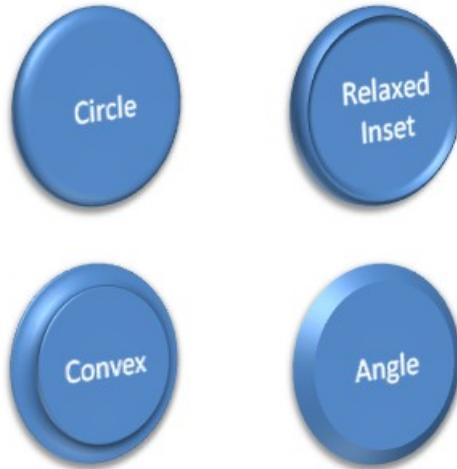
16 The 3-D styles section encompasses the properties for 3-D shapes. These properties are those that get applied  
17 to the 3-D shape, such as a bevel or a contour, and they define the look of the shape in 3-D. A number of  
18 simple types used within the complex types of this group are defined below.

#### 19 **5.6.3.1 Simple Types**

20 The simple types defined here outline the different presets available to the user. These presets are applied to  
21 the shapes through the complex type definitions that are outlined later.

##### 22 **5.6.3.1.1 Bevel Type**

23 The simple type, ST\_BevelPresetType, defines a preset bevel for a shape and some examples can be seen in  
24 figure 3.



1   Figure 9: Different bevel types applied to a shape

2   The different types of bevels available are:

3   relaxedInset

4   circle

5   slope

6   cross

7   angle

8   softRound

9   convex

10   coolSlant

11   divot

12   riblet

13   hardedge

14   artDeco

#### 15   5.6.3.1.2              Preset Material Type

16   The simple type, ST\_PresetMaterialType, defines a material for the shape. The material properties describe  
17   the surface appearance of the shape, and are used in lighting calculations to define exactly how the light  
18   interacts with the shape. Some example material types can be seen in figure 4.



19

20   Figure 10: Different material types on a shape.

21   The different preset material types are:

```

1 legacyMatte
2 legacyPlastic
3 legacyMetal
4 legacyWireframe
5 matte

6 plastic
7 metal
8 warmMatte
9 translucentPowder
10 powder

11 dkEdge
12 softEdge
13 clear
14 flat
15 softMetal

```

### 5.6.3.2 Complex Types

The complex types in this area define the actual 3-D properties that get applied to a shape. These properties work together in order to define the geometry of a shape along with the scene related properties that define the look of the geometry of the shape.

#### 5.6.3.2.1 Bevel

The complex type, CT\_Bevel, defines a bevel for a shape. The bevel consists of a width and a height value, along with a preset bevel. The complex type is defined in the following manner:

```

23 <xsd:complexType name="CT_Bevel">
24   <xsd:attribute name="w" type="ST_PositiveCoordinate" use="optional"
25     default="76200" />
26   <xsd:attribute name="h" type="ST_PositiveCoordinate" use="optional"
27     default="76200" />
28   <xsd:attribute name="prst" type="ST_BevelPresetType" use="optional"
29     default="circle" />
30 </xsd:complexType>

```

#### 5.6.3.2.2 Shape 3-D

The complex type, CT\_Shape3D, defines all of the 3-D properties associated with an individual shape. A shape can have two bevels, one on the top and one on the bottom. An extrusion color also defined, which, when applied, applies a color to the surface of the extrusion. There is also an extrusion width, which defines the width of the extrusion. A contour color and width can be defined for the shape. A z-axis anchor is defined within the complex type and is the anchor relative to the shape's top face. The shape 3-D complex type also holds a present material. Finally the shape 3-D contains another element just as in previous complex types, which is used for future extensibility. The CT\_Shape3D complex type is defined in the following manner:

```

1   <xsd:complexType name="CT_Shape3D">
2     <xsd:sequence>
3       <xsd:element name="bevelT" type="CT_Bevel" minOccurs="0"
4         maxOccurs="1" />
5       <xsd:element name="bevelB" type="CT_Bevel" minOccurs="0"
6         maxOccurs="1" />
7       <xsd:element name="extrusionClr" type="CT_Color" minOccurs="0"
8         maxOccurs="1" />
9       <xsd:element name="contourClr" type="CT_Color" minOccurs="0"
10        maxOccurs="1" />
11      <xsd:element name="ext" type="CT_OfficeArtExtension" minOccurs="0"
12        maxOccurs="1" />
13    </xsd:sequence>
14    <xsd:attribute name="z" type="ST_Coordinate" use="optional"
15      default="0" />
16    <xsd:attribute name="extrusionH" type="ST_PositiveCoordinate"
17      use="optional" default="0" />
18    <xsd:attribute name="contourW" type="ST_PositiveCoordinate"
19      use="optional" default="0" />
20    <xsd:attribute name="prstMaterial" type="ST_PresetMaterialType"
21      use="optional" default="warmMatte" />
22  </xsd:complexType>

```

### 23 5.6.3.2.3 Flat Text

24 The complex type, CT\_FlatText, defines a text object in a 3-D scene that should be rendered as a normal, flat,  
25 text overlay outside of the 3-D scene. The complex type is defined in the following manner:

```

26   <xsd:complexType name="CT_FlatText">
27     <xsd:attribute name="z" type="ST_Coordinate" use="optional"
28       default="0" />
29   </xsd:complexType>

```

### 30 5.6.3.2.4 Group, Text 3-D

31 The final structure to be defined is a group, EG\_Text3D, which describes how text should be applied in the 3-  
32 D scene. If the text object is a member of the 3-D scene, then there are three different ways it can be  
33 displayed:

- 34 • If no EG\_Text3D choice is provided, the text will be rendered in a scene coherent manner and will be  
35 rendered in perspective inside of the 3D scene as a planar shape inside the 3-D.
- 36 • If CT\_Shape3D is provided then the text will be scene coherent and fully 3-D.
- 37 • If CT\_FlatText is provided then the text will be drawn as normal 2-D text rendered on top of the 3-  
38 D scene.

39 An EG\_Text3D is defined in the following manner:

```

1   <xsd:group name="EG_Text3D">
2     <xsd:choice oxsd:cname="Text3DChoice"
3       oxsd:cnameMember="text3DChoice">
4       <xsd:element name="sp3dtype="CT_Shape3D" minOccurs="1"
5         maxOccurs="1"/>
6       <xsd:element name="flatTx" type="CT_FlatText" minOccurs="1"
7         maxOccurs="1" />
8     </xsd:choice>
9   </xsd:group>
```

## 10 5.7 Coordinate Systems and Transformations

### 11 5.7.1 Introduction

12 This document provides an overview of the transformation elements for shapes and groups, represented by  
13 <a:xfrm> in dml-baseTypes.xsd. These schemas are for the representation of scaling and rotation on individual  
14 shapes and groups.

15 §5.7.2, §5.7.3, and §5.7.4 provide a qualitative overview of the transformation pipeline. §5.7.6 provides  
16 mathematical details.

### 17 5.7.2 Coordinate System

18 All DrawingML shapes are located on a 2-D Cartesian coordinate space with the origin (0,0) in the upper left-  
19 hand corner of the canvas. The x-axis coordinates grow positively as one moves from left to right, and the y-  
20 axis coordinates grow positively as one moves from top to bottom.

21 Coordinates are measured in EMUs (914400 EMUs per inch), and can be positive or negative.

### 22 5.7.3 Shape Transformations

23 In this subclause, we describe the transformation pipeline for a shape. To summarize, the *shape*  
24 *transformation* for a shape is defined as the following sequence of operations:

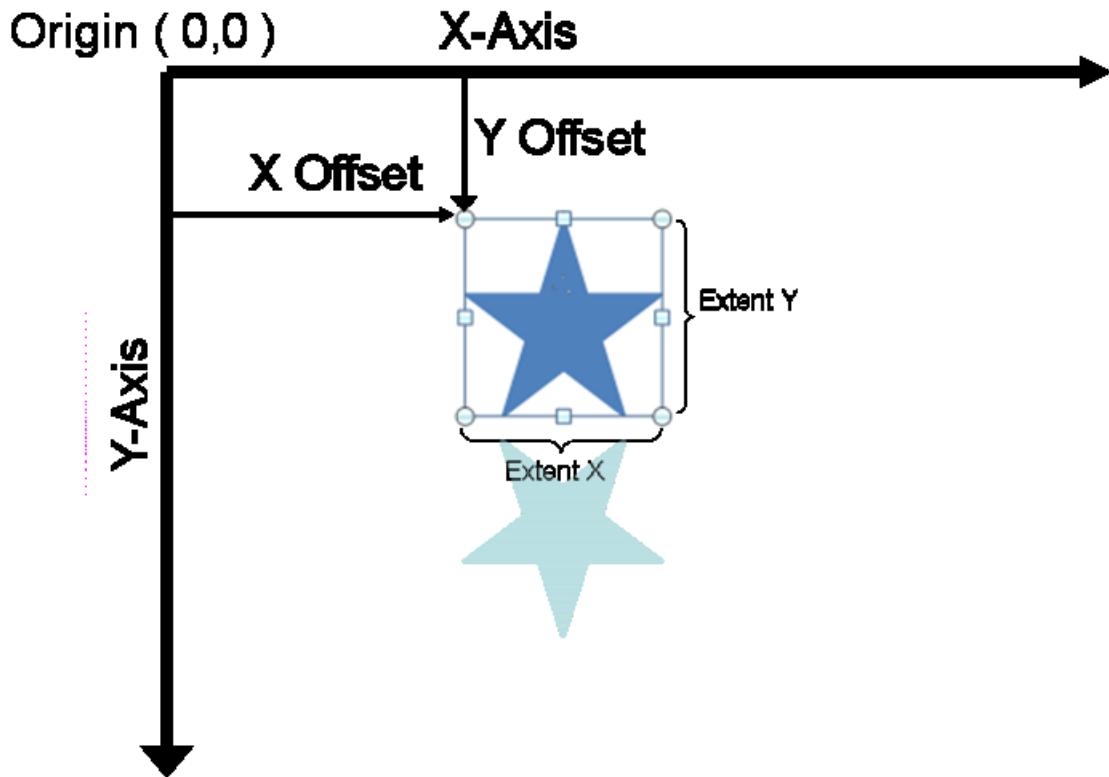
- 25 1. The translation and scaling required to transform its original bounding box to a rectangle specified by  
26 the offset and extents.
- 27 2. A flip across the center of the bounding box according to flipH and flipV.
- 28 3. A rotation about the center of the bounding box according to the rot attribute.

29 To render a shape that is not inside a group (§5.7.4), the renderer simply applies the shape transformation to  
30 the original shape.

#### 31 5.7.3.1 Scaling and Translating a Shape

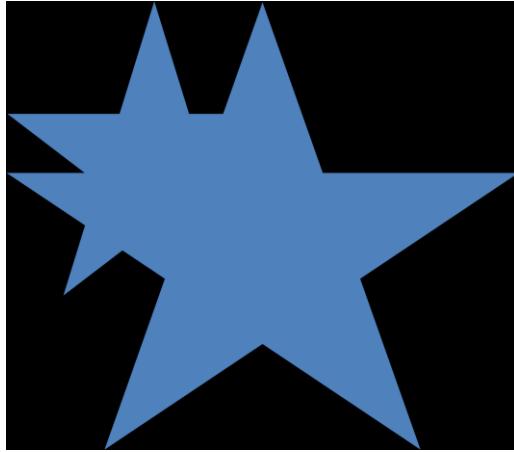
32 The shape is scaled horizontally, scaled vertically, and translated in both dimensions, to fill a given bounding  
33 box. The bounding box is represented by specifying an offset in x and y (attributes x and y of a:off) and extents  
34 in x and y (attributes cx and cy of a:ext, both of which must be greater than or equal to zero). The upper left

- 1 corner of the bounding box is located at the offset, and the lower right corner of the bounding box is located at the offset plus extent.
- 2
- 3 If the starting shape has zero width (e.g., it is a vertical line), then the cx attribute of a:ext is ignored and the horizontal scaling is skipped. Similarly, if the starting shape has zero height, then the cy attribute of a:ext is ignored and the vertical scaling is skipped.
- 4
- 5



- 6
- 7 The following XML fragment represents the offset and extents for the star shape above:
- 8     

```
<a:xfrm>
9     <a:off x="1866680" y="990600"/>
10    <a:ext cx="1371600" cy="1371600"/>
11  </a:xfrm>
```
- 12 Notice that as demonstrated with the example above, any effects attached to the shape are disregarded when scaling and translating the shape to fill the given bounding box.
- 13
- 14 This example illustrates that no additional parameters are needed to represent the scaling of a shape. The bounding-box parameters are sufficient to represent scaling. The following XML Fragments represent the offset and extents for a star shape, before and after scaling. In this particular example, the bounding boxes have been chosen to have the same upper-left corner, i.e., the same offset.
- 15
- 16
- 17



1

2 Before scaling (small star):

```

3   <a:xfrm>
4     <a:off x="1066800" y="990600"/>
5     <a:ext cx="1371600" cy="1371600"/>
6   </a:xfrm>
```

7 After scaling (large star):

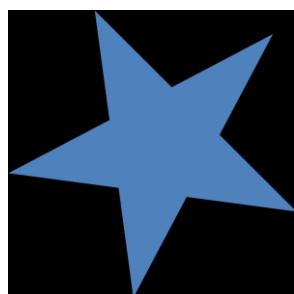
```

8   <a:xfrm>
9     <a:off x="1066800" y="990600"/>
10    <a:ext cx="2438400" cy="2133600"/>
11  </a:xfrm>
```

### 5.7.3.2 Rotating a Shape

13 Rotation is represented with the rot attribute. The shape is rotated clockwise about the bounding-box center,  
14 by the amount specified in the rot attribute. Each unit of rotation is 1/1,000 of an arc minute (1/60,000 of a  
15 degree).

16 This example represents the small star from above, with a subsequent 45-degree rotation clockwise. Since the  
17 y axis points down, a clockwise rotation is positive.



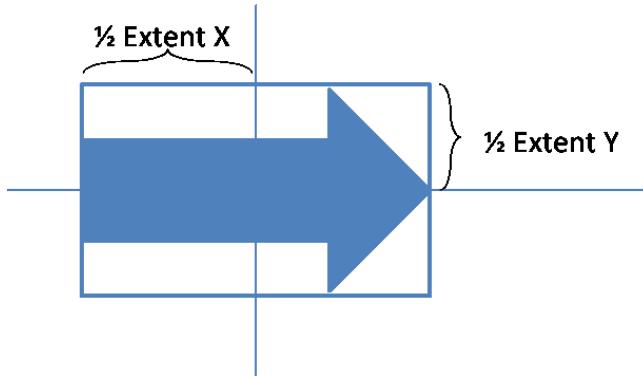
18

```

1   <a:xfrm rot="2700000">
2     <a:off x="1066800" y="990600"/>
3     <a:ext cx="1371600" cy="1371600"/>
4   </a:xfrm>
```

### 5.7.3.3 Flipping a Shape

6 A flip is a reflection across a vertical or horizontal line that intersects the center of the bounding box. The  
7 optional flipH and flipV attributes control horizontal and vertical flipping, respectively. Each is absent or equal  
8 to 0 if no flipping is to be performed, and equal to 1 if flipping is to be performed.



9  
10 The following XML fragment illustrates a shape that has been flipped both horizontally and vertically.

```

11   <a:xfrm flipH="1" flipV="1">
12     <a:off x="3964937" y="2652643"/>
13     <a:ext cx="168838" cy="1219199"/>
14   </a:xfrm>
```

### 5.7.4 Group Transformations

16 A group is composed of zero to many shapes. Because a group is a shape, this composition relationship can  
17 nest recursively. (A group with zero shapes is degenerate; it produces no user-visible output. A group with one  
18 shape is also degenerate; it has no representational power beyond that of the one shape.)

19 The definition of a group transformation is identical to that of a shape transformation, except that in place of  
20 the pre-transform bounding box of a shape, we use the union of all of its children prior to their individual  
21 rotations. To summarize, a *group transformation* is the following sequence of operations:

- 22 1. The translation and scaling required to transform the union of the children's bounding boxes to a  
23 rectangle defined by the group's offset and extent attributes.
- 24 2. A flip about that bounding box according to the flipH and flipV attributes.
- 25 3. A rotation about the center of that bounding box according to the rot attribute.

26 To render a simple shape that is inside a group hierarchy, the renderer does not simply apply the shape  
27 transformation and all parent group transformations to the original shape. Instead (see §5.7.5), it applies the  
28 transformation equal to the following sequence of operations:

1. Horizontal scaling and flipping by a factor equal to the product of the horizontal scalings and flips in its own transformation and those of its parents.
2. Vertical scaling and flipping by a factor equal to the product of the vertical scalings and flips in its own transformation and those of its parents.
3. Rotation by an amount equal to the sum of the rotations in its own transformation and those of its parents.
4. Translation such that its center coincides with the point obtained by applying the shape transformation and all parent group transformations to the shape's original center.

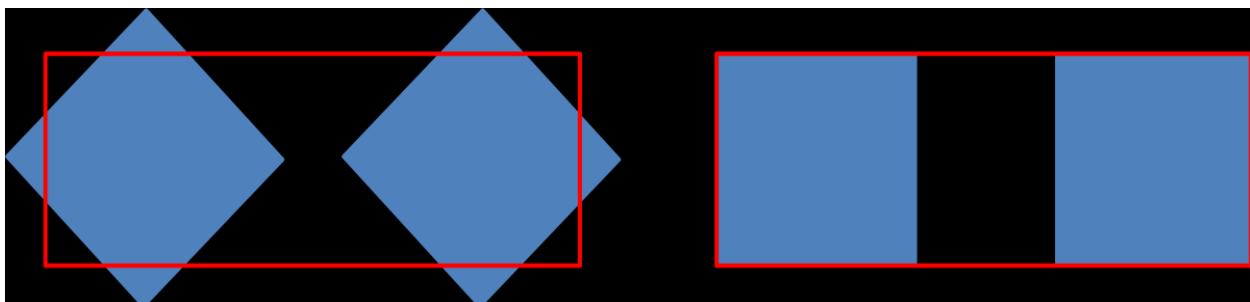
9 Because of the similarity with the transformation pipeline for a shape, the forthcoming subsections primarily  
10 cover illustrative examples.

#### 11 5.7.4.1 Scaling and Translating a Group

12 The group is scaled horizontally, scaled vertically, and translated in both dimensions. The parameters are  
13 chosen to transform the child bounding box (specified by a:chOff and a:chExt) to the group bounding box  
14 (specified by a:off and a:ext). The *child bounding box* is defined as the bounding box around the group's  
15 children as they would have been had their rot attributes been absent.

16 It is possible for the child bounding box to have a zero value for cx or cy in a:chExt, e.g., because the starting  
17 shape is a horizontal or vertical line, or because the starting shape was scaled to have zero width or height.  
18 Such a case is handled in the same way as previously described for simple shapes.

19 This example demonstrates the definition of the child bounding box. The two shapes on the left, rotated  
20 squares, are grouped. The two shapes on the right, non-rotated squares, are also grouped.



21  
22 The red bands are not part of the drawing; each represents the child bounding box of a group. In the XML  
23 fragments, the child bounding boxes have identical y values, illustrating that they are computed based on the  
24 bounding boxes of the squares prior to their rotation.

25 For the left-hand group:

```

1   <a:xfrm>
2     <a:off x="762000" y="1828800" />
3     <a:ext cx="3327400" cy="1219200" />
4     <a:chOff x="762000" y="1828800" />
5     <a:chExt cx="3327400" cy="1219200" />
6   </a:xfrm>
```

7 For the right-hand group:

```

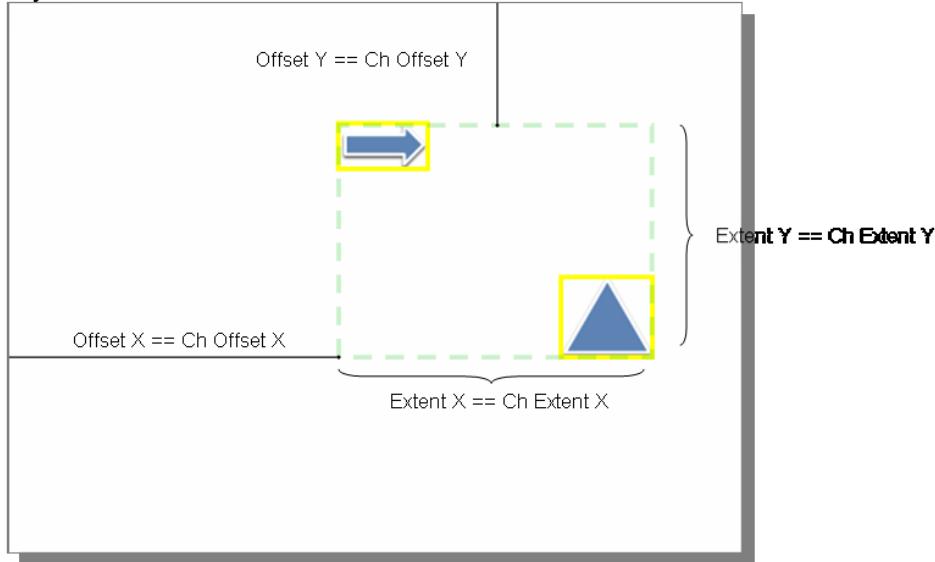
8   <a:xfrm>
9     <a:off x="4978400" y="1828800" />
10    <a:ext cx="3327400" cy="1219200" />
11    <a:chOff x="4978400" y="1828800" />
12    <a:chExt cx="3327400" cy="1219200" />
13  </a:xfrm>
```

14 The remainder of the examples in this subsection illustrate translation and scaling of a group.

15 In this situation, two shapes are grouped: an arrow and a triangle. No further translation, scaling, rotation, or  
16 flipping is applied.

17 To represent this situation, the child bounding box is the bounding box around both of these shapes; and  
18 because no further transformation is applied, the group bounding box is equal to the child bounding box.

(0,0)



19

20 The following is an XML snippet representing the transform variables of the group.

```

1   <p:grpSpPr>
2     <a:xfrm>
3       <a:off x="2209800" y="2514600"/>
4       <a:ext cx="4038600" cy="2286000"/>
5       <a:chOff x="2209800" y="2514600"/>
6       <a:chExt cx="4038600" cy="2286000"/>
7     </a:xfrm>
8   </p:grpSpPr>

```

9 This example illustrates that no additional parameters are needed to represent the translation of a group. A  
10 group is moved to the right. The following diagram shows the starting state, prior to the translation. Note that  
11 offset==ChOffset and extent==ChExtent.

12

Before	Group Shape		Child Shapes		
		X		X	
	Offset	914	1524	→ Offset	914
	Extent	3803	2840	→ Extent	978
	ChOffset	914	1524	▲ Offset	3657
	ChExtent	3803	2840	▲ Extent	1060

13

14 Increasing the x component of the offset moves the group to the right.

After	Group Shape		Child Shapes		
		X		X	Y
	Offset	4197	1524	→ Offset	914
	Extent	3803	2840	→ Extent	978
	ChOffset	914	1524	▲ Offset	3657
	ChExtent	3803	2840	▲ Extent	1060

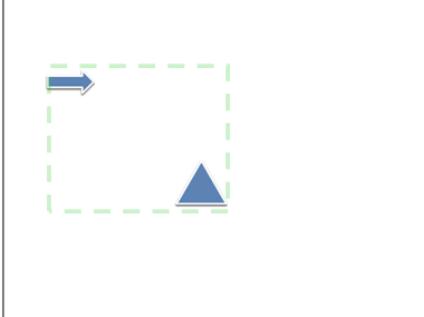
15

16 Similarly, scaling can be performed by adjusting the group bounding box.

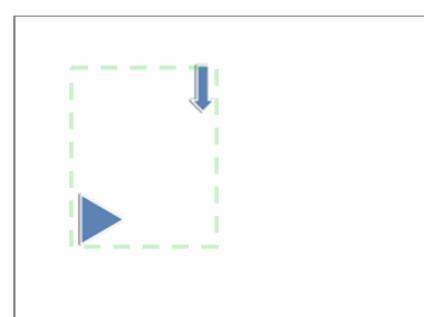
#### 5.7.4.2 Rotating a Group

18 Group rotation is identical to shape rotation. The group is rotated clockwise about the bounding-box center, by  
19 the amount specified in the rot attribute.

- 1 In this example, group is rotated 90 degrees clockwise. The following diagram shows the starting state, prior to  
 2 the rotation.

Before		<b>Group Shape</b>	<b>Child Shapes</b>																													
		<table border="1"> <thead> <tr> <th></th><th>X</th><th>Y</th></tr> </thead> <tbody> <tr> <td>Offset</td><td>914</td><td>1524</td></tr> <tr> <td>Extent</td><td>3803</td><td>2840</td></tr> <tr> <td>ChOffset</td><td>914</td><td>1524</td></tr> <tr> <td>ChExtent</td><td>3803</td><td>2840</td></tr> </tbody> </table>		X	Y	Offset	914	1524	Extent	3803	2840	ChOffset	914	1524	ChExtent	3803	2840	<table border="1"> <thead> <tr> <th></th><th>X</th><th>Y</th></tr> </thead> <tbody> <tr> <td>→ Offset</td><td>914</td><td>1524</td></tr> <tr> <td>→ Extent</td><td>978</td><td>484</td></tr> <tr> <td>▲ Offset</td><td>3657</td><td>3449</td></tr> <tr> <td>▲ Extent</td><td>1060</td><td>914</td></tr> </tbody> </table>		X	Y	→ Offset	914	1524	→ Extent	978	484	▲ Offset	3657	3449	▲ Extent	1060
	X	Y																														
Offset	914	1524																														
Extent	3803	2840																														
ChOffset	914	1524																														
ChExtent	3803	2840																														
	X	Y																														
→ Offset	914	1524																														
→ Extent	978	484																														
▲ Offset	3657	3449																														
▲ Extent	1060	914																														

- 3
- 4 Setting the rotation attribute to 5,400,000 rotates the group clockwise 90 degrees.

After		<b>Group Shape</b>	<b>Child Shapes</b>																													
		<table border="1"> <thead> <tr> <th></th><th>X</th><th>Y</th></tr> </thead> <tbody> <tr> <td>Offset</td><td>914</td><td>1524</td></tr> <tr> <td>Extent</td><td>3803</td><td>2840</td></tr> <tr> <td>ChOffset</td><td>914</td><td>1524</td></tr> <tr> <td>ChExtent</td><td>3803</td><td>2840</td></tr> </tbody> </table>		X	Y	Offset	914	1524	Extent	3803	2840	ChOffset	914	1524	ChExtent	3803	2840	<table border="1"> <thead> <tr> <th></th><th>X</th><th>Y</th></tr> </thead> <tbody> <tr> <td>→ Offset</td><td>914</td><td>1524</td></tr> <tr> <td>→ Extent</td><td>978</td><td>484</td></tr> <tr> <td>▲ Offset</td><td>3657</td><td>3449</td></tr> <tr> <td>▲ Extent</td><td>1060</td><td>914</td></tr> </tbody> </table>		X	Y	→ Offset	914	1524	→ Extent	978	484	▲ Offset	3657	3449	▲ Extent	1060
	X	Y																														
Offset	914	1524																														
Extent	3803	2840																														
ChOffset	914	1524																														
ChExtent	3803	2840																														
	X	Y																														
→ Offset	914	1524																														
→ Extent	978	484																														
▲ Offset	3657	3449																														
▲ Extent	1060	914																														
		<b>Rotation = 5,400,000</b>																														

- 5
- 6 **5.7.5 Nesting Transformations**
- 7 The following example illustrates the rendering procedure described at the end of the introduction to §5.7.4,  
 8 which differs from a conventional transformation pipeline in any case where a scaled group contains a rotated  
 9 child.
- 10 [Example: In this example, the diagram on the left side is a group that comprises a rotated red square centered  
 11 inside a non-rotated blue square. The diagram on the right side is the same group, scaled horizontally. The red  
 12 square scales along an axis parallel to its own edges instead of an axis parallel to the edges of the blue square.]



## 1 5.7.6 Transformation Matrices

2 The preceding sections fully define the transformation pipeline and its parameters. This section assists  
 3 developers in implementing the pipeline by describing it mathematically. It is generalized to describe either  
 4 the shape transformation pipeline or the group transformation pipeline.

### 5 5.7.6.1 Symbol Definitions

6 Let the following symbols represent parameters described in the preceding sections.

$(B_x, B_y)$  For a shape: upper left corner of untransformed shape.  
 For a group: upper left corner of child bounding box (a:chOff).

$(D_x, D_y)$  For a shape: (width,height) of untransformed shape.  
 For a group: (width,height) of child bounding box (a:chExt).

$(B'_x, B'_y)$  Upper left corner of bounding box (a:off), prior to rotation and flip.

$(D'_x, D'_y)$  (width,height) of bounding box (a:ext), prior to rotation and flip.

$\theta$  Clockwise rotation (from the attribute rot, which uses thousandths of an arc minute).

$F_x$  -1 if flipH is true; +1 otherwise.

$F_y$  -1 if flipV is true; +1 otherwise.

7 We use homogeneous coordinates, in which a point p is represented in the form

$$8 \quad p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} .$$

9 We use the convention in which transformations are applied by left-hand multiplication. Thus, to obtain the  
 10 point  $p'$  by applying transformation T to point p we write:

$$11 \quad p' = T p$$

### 12 5.7.6.2 Transformation Pipeline

13 The entire transformation pipeline that defines either a shape transformation or a group transformation is  
 14 represented by the matrix T, defined at the end of this subsection.

15 Scaling and translation are produced by the following matrix:

$$16 \quad T_{st} = \begin{bmatrix} \frac{D'_x}{D_x} & 0 & B'_x - \left( \frac{D'_x}{D_x} \right) B_x \\ 0 & \frac{D'_y}{D_y} & B'_y - \left( \frac{D'_y}{D_y} \right) B_y \\ 0 & 0 & 1 \end{bmatrix} .$$

1 The following matrix translates the bounding box to the origin in preparation for rotation and flipping:

$$2 \quad U = \begin{bmatrix} 1 & 0 & -\left(B'_x + \frac{D'_x}{2}\right) \\ 0 & 1 & -\left(B'_y + \frac{D'_y}{2}\right) \\ 0 & 0 & 1 \end{bmatrix} .$$

3 Rotation and flipping are produced by the following matrix:

$$4 \quad T_{rf} = U^{-1} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_x & 0 & 0 \\ 0 & F_y & 0 \\ 0 & 0 & 1 \end{bmatrix} U .$$

5 The entire transformation pipeline for one step in the group hierarchy is represented by the matrix

$$6 \quad T = T_{rf} T_{st}$$

## 7 **5.8 Shape Properties and Effects**

### 8 **5.8.1 Introduction**

9 This subclause provides a high-level overview of the Fill Properties, Line Properties, and Effect Properties  
10 described in the following schemas: dml-shapeLineProperties.xsd, dml-shapeEffects.xsd, dml-baseTypes.xsd.

11 Color Models are also covered, as Fills, Lines, and Effects all reference color model schemas to represent color.

### 12 **5.8.2 Color Models**

13 There are several methods of expressing color: scrgbClr, srgbClr, hslClr, sysClr, schemeClr, and prstClr.  
14 Although srgbClr is the most commonly used model, the rationale for having various equivalent color models  
15 stems from a desire to have different ways of naturally expressing a color choice.

#### 16 **5.8.2.1 scrgbClr**

17 scrgbClr is a legacy form of expressing Red, Green, Blue color. Values are expressed in Percentages. r, g, and b  
18 are all required and correspond to red, green, and blue, respectively.

19 `<a:scrgbClr r="10000" g="20000" b="30000">`

#### 20 **5.8.2.2 srgbClr**

21 srgbClr is similar to scrgbClr with the exception that instead of expressing the values as percentages, they are  
22 specified using two hex digits per color, in the order RGB.

23 `<a:srgbClr val="FFFF00">`

#### 24 **5.8.2.3 hslClr**

25 hslClr represents a color using the Hue, Saturation, and Luminescence color model. Values are expressed in  
26 Percentages. h, s, and l are all required, and correspond to hue, saturation, and luminescence respectively. A  
27 perceptual gamma of 2.2 is assumed.

```
1 <a:hslClr h="10000" s="20000" l="30000">
```

#### 2 5.8.2.4 sysClr

3 sysClr represents a system color, and introduces a level of indirection. For example, specifying:

```
4 <a:sysClr val="windowText">
```

5 binds the color to be the color chosen in the system for "Window Text". The possible values are:

```
6   scrollBar  
7   background  
8   activeCaption  
9   inactiveCaption  
10  menu  
11  window  
12  windowFrame  
13  menuText  
14  windowText  
15  captionText  
16  activeBorder  
17  inactiveBorder  
18  appWorkspace  
19  highlight  
20  highlightText  
21  btnFace  
22  btnShadow  
23  grayText  
24  btnText  
25  inactiveCaptionText  
26  btnHighlight  
27  3dDkShadow  
28  3dLight  
29  infoText  
30  infoBk  
31  hotLight  
32  gradientActiveCaption  
33  gradientInactiveCaption  
34  menuHighlight  
35  menuBar
```

#### 36 5.8.2.5 schemeClr

37 schemeClr represents a color from a theme. The color changes if theme bindings change. For example,  
38 specifying:

1       <a:schemeClr val="lt1">  
 2     binds the color to be Light 1 color of the current theme. The possible values are:

bg1	semantic background color
bg1	semantic background color
tx1	semantic text color
bg2	semantic additional background color
tx2	semantic additional text color
accent1	extra scheme color 1
accent2	extra scheme color 2
accent3	extra scheme color 3
accent4	extra scheme color 4
accent5	extra scheme color 5
accent6	extra scheme color 6
hlink	hyperlink color
folHlink	followed hyperlink color
dk1	main dark color 1
lt1	Main light color 1
phClr	a color used in theme definitions which means "use the color of the style"
dk2	main dark color 2
lt2	main light color 2

3       5.8.2.6           prstClr

4     prstClr represents a preset color. This is a legacy definition of colors which is no longer currently used. A preset  
 5     color is a choice from among several presets provided in older versions of Office.

6       <a:prstClr val="black">  
 7     The selected color is "black". Valid values for this setting are:

```
1    aliceBlue
2    antiqueWhite
3    aqua
4    aquamarine
5    azure
6    beige
7    bisque
8    black
9    blanchedAlmond
10   blue
11   blueViolet
12   brown
13   burlyWood
14   cadetBlue
15   chartreuse
16   chocolate
17   coral
18   cornflowerBlue
19   cornsilk
20   crimson
21   cyan
22   dkBlue
23   dkCyan
24   dkGoldenrod
25   dkGray
26   dkGreen
27   dkKhaki
28   dkMagenta
29   dkOliveGreen
30   dkOrange
31   dkOrchid
32   dkRed
33   dkSalmon
34   dkSeaGreen
35   dkSlateBlue
36   dkSlateGray
37   dkTurquoise
38   dkViolet
39   deepPink
40   deepSkyBlue
```

```
1 dimGray
2 dodgerBlue
3 firebrick
4 floralWhite
5 forestGreen
6 fuchsia
7 gainsboro
8 ghostWhite
9 gold
10 goldenrod
11 gray
12 green
13 greenYellow
14 honeydew
15 hotPink
16 indianRed
17 indigo
18 ivory
19 khaki
20 lavender
21 lavenderBlush
22 lawnGreen
23 lemonChiffon
24 ltBlue
25 ltCoral
26 ltCyan
27 ltGoldenrodYellow
28 ltGray
29 ltGreen
30 ltPink
31 ltSalmon
32 ltSeaGreen
33 ltSkyBlue
34 ltSlateGray
35 ltSteelBlue
36 ltYellow
37 lime
38 limeGreen
39 linen
40 magenta
```

```
1 maroon
2 medAquamarine
3 medBlue
4 medOrchid
5 medPurple
6 medSeaGreen
7 medSlateBlue
8 medSpringGreen
9 medTurquoise
10 medVioletRed
11 midnightBlue
12 mintCream
13 mistyRose
14 moccasin
15 navajoWhite
16 navy
17 oldLace
18 olive
19 oliveDrab
20 orange
21 orangeRed
22 orchid
23 paleGoldenrod
24 paleGreen
25 paleTurquoise
26 paleVioletRed
27 papayaWhip
28 peachPuff
29 peru
30 pink
31 plum
32 powderBlue
33 purple
34 red
35 rosyBrown
36 royalBlue
37 saddleBrown
38 salmon
39 sandyBrown
40 seaGreen
```

```

1  seaShell
2  sienna
3  silver
4  skyBlue
5  slateBlue
6  slateGray
7  snow
8  springGreen
9  steelBlue
10 tan
11 teal
12 thistle
13 tomato
14 transparent
15 turquoise
16 violet
17 wheat
18 white
19 whiteSmoke
20 yellow
21 yellowGreen

```

### 5.8.3 Color Transforms

A color transform is a modification to related properties of an underlying color. For example, transparency is a property that is related to color. Color transforms are specified as child tags off any color model's tag.

```

<a:solidFill>
  <a:srgbClr val="00B050">
    <a:alpha val="51000"/>
  </a:srgbClr>
</a:solidFill>

```

The following are the allowed color transforms and descriptions of the transformations they apply:

- tint: Yields a lighter version of its input color. A 10% tint is 10% of the input color combined with 90% white.
- shade: Yields a darker version of its input color. A 10% shade is 10% of the input color combined with 90% black.
- comp: Yields the complement of its input color. For example, the complement of red is green.
- inv: Yields the inverse of its input color. For example, the inverse of red (1,0,0) is cyan (0,1,1).
- gray: Yields a grayscale of its input color, taking into relative intensities of the red, green, and blue primaries.
- alpha: Yields its input color with the specified opacity, but with its color unchanged.

- 1     • alphaOff: Yields a more or less opaque version of its input color. An alpha offset never increases the  
2       alpha beyond 100% or decreases below 0%; i.e., the result of the transform pins the alpha to the range  
3       of [0%,100%]. A 10% alpha offset increases a 50% opacity to 60%. A -10% alpha offset decreases a  
4       50% opacity to 40%.
- 5     • alphaMod: Yields a more or less opaque version of its input color. An alpha modulate never increases  
6       the alpha beyond 100%. A 200% alpha modulate makes a input color twice as opaque as before. A  
7       50% alpha modulate makes a input color half as opaque as before.
- 8     • hue: Yields the input color with the specified hue, but with its saturation and luminance unchanged.
- 9     • hueOff: Yields the input color with its hue shifted, but with its saturation and luminance unchanged.
- 10    • hueMod: Yields the input color with its hue modulated by the given percentage.
- 11    • sat: Yields the input color with the specified saturation, but with its hue and luminance unchanged.  
12       Typically saturation values fall in the range [0%, 100%].
- 13    • satOff: Yields the input color with its saturation shifted, but with its hue and luminance unchanged.
- 14    • satMod: Yields the input color with its saturation modulated by the given percentage. A 50%  
15       saturation modulate will reduce the saturation by half. A 200% saturation modulate will double the  
16       saturation.
- 17    • lum: Yields the input color with the specified luminance, but with its hue and saturation unchanged.  
18       Typically, luminance values fall in the range [0%,100%].
- 19    • lumOff: Yields the input color with its luminance shifted, but with its hue and saturation unchanged.
- 20    • lumMod: Yields the input color with its luminance modulated by the given percentage. A  
21       50% luminance modulate will reduce the luminance by half. A 200% luminance modulate will double the  
22       luminance.
- 23    • red: Yields the input color with the specified red component, but with its green and blue components  
24       unchanged.
- 25    • redOff: Yields the input color with its red component shifted, but with its green and blue components  
26       unchanged.
- 27    • redMod: Yields the input color with its red component modulated by the given percentage. A 50% red  
28       modulate will reduce the red component by half. A 200% red modulate will double the red  
29       component.
- 30    • green: Yields the input color with the specified green component, but with its red and blue  
31       components unchanged.
- 32    • greenOff: Yields the input color with its green component shifted, but with its red and blue  
33       components unchanged.
- 34    • greenMod: Yields the input color with its green component modulated by the given percentage. A  
35       50% green modulate will reduce the green component by half. A 200% green modulate will double the  
36       green component.
- 37    • blue: Yields the input color with the specified blue component, but with its red and green components  
38       unchanged.
- 39    • blueOff: Yields the input color with its blue component shifted, but with its red and green components  
40       unchanged.

- blueMod: Yields the input color with its blue component modulated by the given percentage. A 50% blue modulate will reduce the blue component by half. A 200% blue modulate will double the blue component.
- gamma: Yields the sRGB gamma shift of its input color.
- invGamma: Yields the inverse sRGB gamma shift of its input color.

## 5.8.4 Fills

There are six types of fills:

- No Fill
- Solid Fill
- Gradient Fill
- Blip Fill
- Pattern Fill
- Group Fill

The se types describe the general structure of all fills; however, not all fills are permitted in all locations. For example, Blip Fills and Group Fills are not permitted on lines.

### 5.8.4.1 Solid Fills

```

<p:sp>
  <p:nvSpPr>...
  <p:spPr>
    <a:xfrm>
      <a:off x="5410200" y="2438400"/>
      <a:ext cx="2895600" cy="304800"/>
    </a:xfrm>
    <a:prstGeom prst="rect">
      <a:avLst/>
    </a:prstGeom>
    <a:solidFill>
      <a:srgbClr val="FFFF00"/>
    </a:solidFill>
  </p:spPr>
  <p:style>...
  <p:txBody>...
</p:sp>

```

**Solid Fill**

A solid fill specifies a single color, using any valid color model

## 1 5.8.4.2 Gradient Fills

```

<a:gradFill>
  <a:gsLst>
    <a:gs pos="69000">
      <a:schemeClr val="accent1"/>
    </a:gs>
    <a:gs pos="0"> Gradient Stop List
      <a:scrgbClr r="0" g="0" b="0"/>
    </a:gs>
  </a:gsLst>
  <a:lin ang="2700000" scaled="1"/>
</a:gradFill>

```

Gradient Fill

2

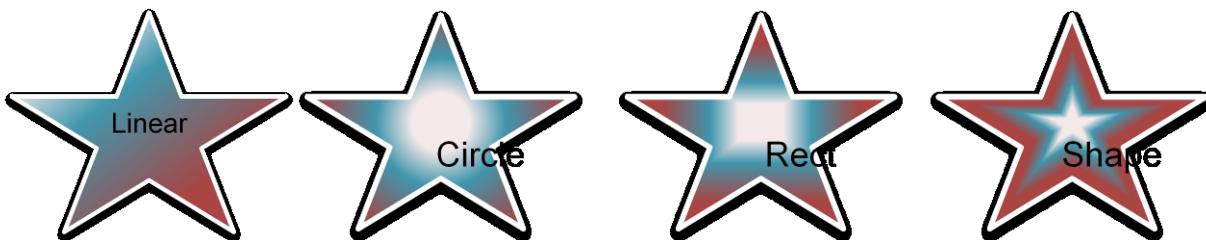
3 Gradient Fills consist of three elements: a list of gradient stops, a shading specification, as well as some  
4 attributes.

5 Two attributes are available on gradient fills.

- 6 • flip specifies how to flip a tile region when using it to fill a larger fill region.
- 7 • rotWithShape specifies whether the fill rotates along with a shape when the shape is rotated.

8 A gradient stop list is a list of locations and colors that make up the gradient fill. Positions are specified as  
9 percentages.

10 The shading specification specifies the two possible kinds of gradient fills: linear, or path based. A linear fill  
11 follows a straight-line direction as specified by the angle of the line. A path-based fill follows the contours of a  
12 well-defined path (such as a shape, circle, or rectangle).



13

1    5.8.4.3              Blip Fills

```

<p:blipFill>
  <a:blip r:embed="rId4" r:link="" />
  <a:srcRect l="11000" t="14000"
    r="20000" b="28667" />
  <a:stretch>
    <a:fillRect />
  </a:stretch>
</p:blipFill>
```



2

3    BLIPs refer to Binary Large Image or Pictures. Blip Fills are made up of several components: a Blip Reference, a  
4    Source Rectangle, and a Fill Mode.

5    The Blip reference, a:blip, is the main reference to the blip content itself. A reference ID serves as the main link  
6    with an attribute allowed to specify compression level of the blip (one of: email, screen, print, hqprint, none).  
7    A Blip Effect may optionally be specified to indicate a modification of the raw blip content. Valid Blip Effects  
8    are:

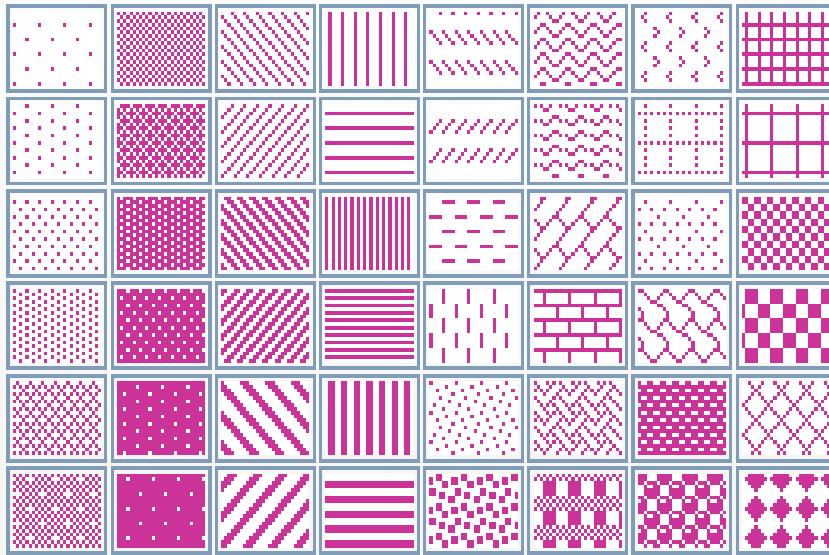
```

9      alphaBiLevel
10     alphaCeiling
11     alphaFloor
12     alphaInv
13     alphaMod
14     alphaModFix
15     alphaRepl
16     biLevel
17     blur
18     clrChange
19     clrRepl
20     duotone
21     fillOverlay
22     grayscl
23     hsl
24     lum
25     tint
```

26    The blip effects mirror the color transformations (see the descriptions in the color transformations subclause  
27    for descriptions of Blip Effects).

- 1 A Source Rectangle, a:srcRect, is used to implement image cropping, and indicates the rectangular window of content which is of interest.
- 2
- 3 Finally, two fill modes are possible, tiling, and stretching. This indicates the behavior to be performed when the user resizes an image to an area larger than the source rectangle. Tiling 'tiles' an image so that image content is simply duplicated while stretching scales the source rectangle content to fill the bounds of the fillRect (bounding box of blip filled shape).
- 4
- 5
- 6

#### 7 5.8.4.4 Pattern Fills



- 8
- 9 Pattern Fills are legacy Office 11 fills which consist of a Foreground Color, a Background Color and a preset pattern value. Possible pattern values are:
- 10

```

11    pct5
12    pct10
13    pct20
14    pct25
15    pct30
16    pct40
17    pct50
18    pct60
19    pct70
20    pct75
21    pct80
22    pct90
23    horz
24    vert
25    lHorz

```

```

1    ltVert
2    dkHorz
3    dkVert
4    narHorz
5    narVert
6    dashHorz
7    dashVert
8    cross
9    dnDiag
10   upDiag
11   ltDnDiag
12   ltUpDiag
13   dkDnDiag
14   dkUpDiag
15   wdDnDiag
16   wdUpDiag
17   dashDnDiag
18   dashUpDiag
19   diagCross
20   smCheck
21   lgCheck
22   smGrid
23   lgGrid
24   dotGrid
25   smConfetti
26   lgConfetti
27   horzBrick
28   diagBrick
29   solidDmnd
30   openDmnd
31   dotDmnd
32   plaid
33   sphere
34   weave
35   divot
36   shingle
37   wave
38   trellis
39   zigZag

```

#### 40 5.8.4.5 Group Fills

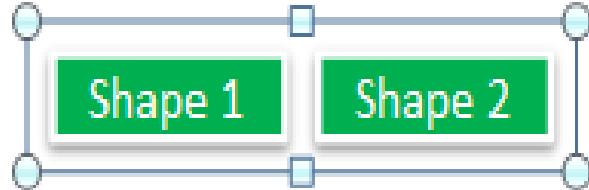
41 When objects are grouped together, a group fill is a convenient structure for indicating that the fill properties  
 42 of any individual element inherits from the fill properties of parent group.

```

<p:grpSp>
  <p:nvGrpSpPr>...
  <p:grpSpPr>
    <a:xfrm>...
    <a:solidFill>
      <a:srgbClr val="00B050"/>
    </a:solidFill>
  </p:grpSpPr>
  <p:sp>
    <p:nvSpPr>...
    <p:spPr>
      <a:xfrm>...
      <a:prstGeom prst="rect">...
      <a:grpFill/>
    </p:spPr>
    <p:style>...
    <p:txBody>...
  </p:sp>
  <p:sp>
    <p:nvSpPr>...
    <p:spPr>
      <a:xfrm>...
      <a:prstGeom prst="rect">...
      <a:grpFill/>
    </p:spPr>
    <p:style>...
    <p:txBody>...
  </p:sp>
</p:grpSp>

```

1



## 2 5.8.5 Line Properties

3 While it is obvious that line properties, *a:ln* are used to represent properties for lines, what may be less  
 4 obvious is where this structure can appear. Lines aren't just for lines-- just about any object can have a line  
 5 property-- usually referring to the outlines that are possible on shapes, pictures, or text. Lines used in this  
 6 context also yield additional characteristics we wish to persist-- like what happens when line segments meet  
 7 (i.e., line joins). So when understanding this section on line properties, it's important to visualize two possible  
 8 cases-- a single line segment and the properties of that segment, and the case of multiple line segments (e.g.,  
 9 an outline of an autoshape). By considering both cases, the meaning of most properties becomes intuitively  
 10 clear.

11 Line properties consist of several sections: line fill properties, line dash properties, line join properties,  
 12 head/tail properties, as well as a few attributes.

### 13 5.8.5.1 Line Fill Properties

14 Line fill properties are a proper subset of general fill properties. One of the following can be used: *noFill*,  
 15 *solidFill*, *gradFill*, or *pattFill*. Blip fills and group fills are not permitted for line fill properties.

### 16 5.8.5.2 Line Dash Properties

17 Line Dash properties may be either one of the presets, *a:prstDash*, or a custom dashing scheme, *a:custDash*.  
 18 For the presets, the following options are available:

- 1     • solid: Solid (continuous) pen.
- 2     • dot: Dot style. [- - - - - - - - - - ]
- 3     • dash: Short dash style. [--- --- --- --- --- --- ]
- 4     • lgDash: Long dash style. [----- ----- ----- ----- ]
- 5     • dashDot" Short dash followed by dot. [--- - --- - --- - --- - ]
- 6     • lgDashDot: Long dash followed by dot. [----- - - - - - - - - ]
- 7     • lgDashDotDot: Long dash followed by two dots. [----- - - - - - - - - - ]
- 8     • sysDash: System short dash style (PS\_DASH). [--- --- --- --- --- --- ]
- 9     • sysDot: System dot style (PS\_DOT). [- - - - - - - - ]
- 10    • sysDashDot: System short dash and one dot (PS\_DASHDOT) [--- --- --- --- - - - - ]
- 11    • sysDashDotDot: System short dash and two dots (PS\_DASHDOTDOT) [--- --- --- --- - - - - - ]
- 12    ]

13    5.8.5.2.1           Custom Dashes

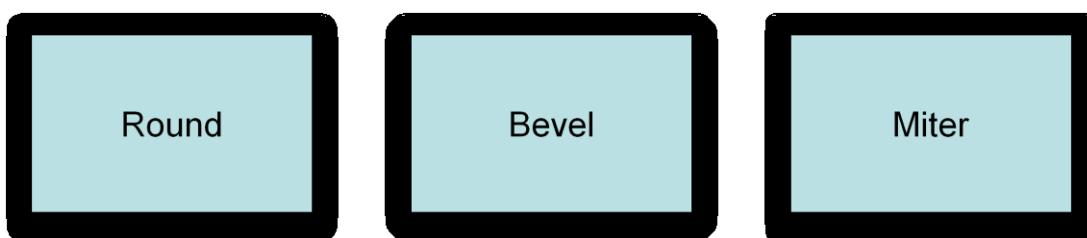
14    Custom dashes allow full flexibility in expressing any dashing scheme. Custom dashes are also known as dash  
 15    stop lists, a:ds, due to the way the custom dashes are expressed. An element of the list specifies two  
 16    attributes: d for the length of the dash relative to line width, and sp for length of the space relative to line  
 17    width. Any number of elements may be combined into a dash stop list for full generality in expressing dashing  
 18    schemes.

19    5.8.5.3           Line Join Properties

```

20 <a:ln w="38100" cap="sq" cmpd="thickThin" algn="ctr">
  <a:solidFill>
    <a:schemeClr val="lt1"/>
  </a:solidFill>
  <a:prstDash val="sysDot"/>
  <a:bevel/>
</a:ln>
```

21    Line join properties are for expressing the visual appearance of what happens when line segments meet. They  
 22    can be round, beveled, or mitered. Notice the corners of the following rectangles, which illustrate the effect  
 23    line join properties have.



24

25    The only attribute of line join properties is lim. This attribute limits the amount by which lines can be extended  
 26    to form a join. Normally, this is a relatively infrequent occurrence, but in the case of nearly parallel lines, this  
 27    attribute comes into play.

1    5.8.5.4        Head/Tail End Properties

2    Head/Tail End properties specify whether there are any special attachments to the head or the tail of a line. All  
 3    parameters are specified in attributes: a type, a w (width of line end to width of line), and a len (length of the  
 4    end relative to the line width). By default, no head/tail end properties are applied. The type can be one of:  
 5    none, triangle, stealth, diamond, oval, arrow. w and len can be one of sm, med and lg corresponding to small,  
 6    medium, and large respectively.

```

<a:ln w="25400" cap="rnd"
      cmpd="sng" algn="ctr">
  <a:solidFill>
    <a:srgbClr val="4F81BD"/>
  </a:solidFill>
  <a:prstDash val="solid"/>
  <a:tailEnd type="arrow" w="lg" len="lg"/>
</a:ln>
```

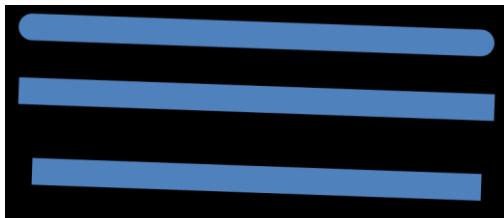


7

8    5.8.5.5        Line Attributes

9    Line Properties, a:ln, takes several attributes: w specifies the line width. cap specifies whether the line ends  
 10   are round (value rnd), square (sq), or flat (flat).

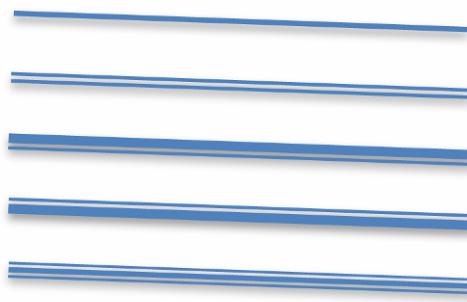
11



12   cmpd specifies a compound line type. Its permitted values are shown below:

## “cmpd” compound line type

- sng (simple)
- dbl (double)
- thickThin
- ThinThick
- tri



13

14   5.8.6        Effects

15   Effects are most naturally applied to shapes, but like fill properties and line properties, they can apply to  
 16   shapes, pictures, and text. Effects are represented two ways: via an Effect List, a:effectLst, or an effects  
 17   container, a:effectDag.

## 1 5.8.6.1 Effects Lists

```

<p:sp>
  <p:nvSpPr>...
  <p:spPr>
    <a:xfrm>...
    <a:prstGeom prst="star5">...
    <a:effectLst>
      <a:glow rad="190500">
        <a:schemeClr val="accent5">
          <a:alpha val="80000"/>
        </a:schemeClr>
      </a:glow>
      <a:outerShdw blurRad="50800" dist="50800"
        dir="2700000" algn="tl"
        rotWithShape="0">
        <a:srgbClr val="000000">
          <a:alpha val="43137"/>
        </a:srgbClr>
      </a:outerShdw>
      <a:reflection stA="75000" endA="10000"
        dist="101600" dir="5400000"
        sy="-100000" algn="bl"
        rotWithShape="0"/>
      <a:softEdge rad="31750"/>
    </a:effectLst>
    <a:scene3d>...
    <a:sp3d>...
  </p:spPr>
  <p:style>...
  <p:txBody>...
</p:sp>

```

2 An effect list is made up of one or more primitive effects that can be applied one after another. The primitives  
3 are:

4  
5     Blur  
6     fillOverlay  
7     glow  
8     innerShdw  
9     outerShdw  
10    prstShdw  
11    reflection  
12    softEdge

## 13 5.8.6.2 Blur

14 Blur blurs all color channels, including alpha. Two attributes, rad (radius of blur) and grow (boolean), apply  
15 here. grow specifies if the bounds should grow as a result of the blurring.

## 1    5.8.6.3        Inner Shadow

```

<a:effectLst>
  <a:innerShdw blurRad="317500"
    dist="293171"
    dir="13500000">
    <a:srgbClr val="000000">
      <a:alpha val="43000"/>
    </a:srgbClr>
  </a:innerShdw>
</a:effectLst>

```



2

3    Inner Shadows contain a color choice, as well as three attributes:

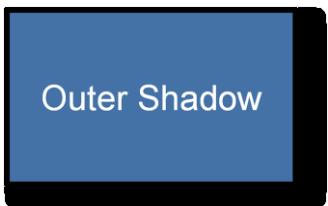
- 4            • blurRad: blur radius  
 5            • dist: how far to offset the shadow  
 6            • dir: direction to offset the shadow

## 7    5.8.6.4        Outer Shadow

```

<a:effectLst>
  <a:outerShdw blurRad="50800" dist="50800"
    dir="2700000"
    sx="106000" sy="106000"
    algn="tl" rotWithShape="0">
    <a:srgbClr val="000000">
      <a:alpha val="43137"/>
    </a:srgbClr>
  </a:outerShdw>
</a:effectLst>

```



8

9    Outer shadows contain a color choice as well as several attributes:

- 11          • blurRad: blur radius  
 12          • dist: how far to offset the shadow  
 13          • dir: direction to offset the shadow  
 14          • sx, sy: horizontal/vertical scale factors  
 15          • kx, ky: horizontal/vertical skew angles

- 1     • align: shadow alignment. Alignment happens first and effectively sets the origin for scale, skew, and  
 2       offset  
 3     • rotWithShape: (boolean) Rotate shadow with shape

4     5.8.6.5           Preset Shadows

5 Preset shadows consist of a color choice, and a preset shadow:

```
6     shdw1
7     shdw2
8     shdw3
9     shdw4
10    shdw5
11    shdw6
12    shdw7
13    shdw8
14    shdw9
15    shdw10
16    shdw11
17    shdw12
18    shdw13
19    shdw14
20    shdw15
21    shdw16
22    shdw17
23    shdw18
24    shdw19
25    shdw20
```

26 The attributes for Preset Shadows are:

- 27     • dist: how far to offset the shadow  
 28     • dir: direction to offset the shadow

29     5.8.6.6           Reflection Effects



```
<a:effectLst>
  <a:reflection blurRad="12700" stA="50000" endPos="75000"
    dist="12700" dir="5400000" sy="-100000"
    align="bl" rotWithShape="0"/>
</a:effectLst>
```

31 Reflections are represented entirely through attributes:

- 1     • blurRad: Blur Radius
- 2     • stA: (Start Alpha) starting reflection opacity
- 3     • stPos: start position along gradient ramp of start alpha value
- 4     • endA: (End Alpha) ending reflection opacity
- 5     • endPos: end position along gradient, ramp of end alpha value
- 6     • dist: how far to offset reflection
- 7     • dir: Direction to offset reflection
- 8     • fadeDir: direction of alpha gradient, ramp relative to shape itself
- 9     • sx, sy: horizontal/vertical scale factors
- 10    • kx, ky: horizontal/vertical skew angles
- 11    • align: reflection alignment
- 12    • rotWithShape: (boolean)
- 13    • rotate: reflection with shape

14    5.8.6.7       Soft Edge Effects



Soft Edge Effect

```

<a:effectLst>
  <a:softEdge rad="127000"/>
</a:effectLst>
```

16    Soft Edge blurs the edges of the applied object subject to the specified blur radius rad.

17    5.8.6.8       Glow Effects

```

<a:effectLst>
  <a:glow rad="101600">
    <a:schemeClr val="accent2">
      <a:alpha val="60000"/>
    </a:schemeClr>
  </a:glow>
</a:effectLst>
```

19    A glow effect is very similar to a soft edge effect, but differs in that it permits a color specification in addition  
20    to rad. Basically, a glow is a soft edge effect, except with the color specified used instead of the object's color.

## 21    5.9      Shape Definitions and Attributes

### 22    5.9.1      Introduction

23    This document provides a high-level overview of the content described in the dml-shapeGeometry.xsd schema.

24    This aspect of DrawingML deals mainly with the shapes and their attributes, and is broken down into two  
25    topics:

- 26    • Working with preset shapes

- 1       • Defining custom shapes and their properties

2     **5.9.2       The Coordinate Systems**

3     To specify a shape there are a few high level systems that must first be understood, namely the coordinate  
4     systems that will be used. These are the document, shape and path coordinate systems, described in the  
5     following sub clauses.

6     **5.9.2.1       The Document Coordinate System**

7     To first specify a shape within a document the document coordinate system must be understood. This system  
8     has both an x and y component and starts with a value of (0,0) in the upper left corner of the document. As the  
9     x-coordinate increases, the point will move to the right. As the y-coordinate increases, the point will move  
10    downwards. The units of measurement within the document coordinate system are EMUs (91440 EMUs/U.S.  
11    inch, 36000 EMUs/cm). In addition to specifying a position for the shape, you must also specify the width and  
12    height of the shape, which is called the extent of the shape. This value is again measured in EMUs. To specify  
13    these two values, the following transform would be used.

```
14 <p:sp>
15   <p:spPr>
16     <a:xfrm>
17       <a:off x="3200400" y="1600200"/>
18       <a:ext cx="1200000" cy="1000000"/>
19     </a:xfrm>
20   </p:spPr>
21 </p:sp>
```

22    Here we can see that this new shape will be placed at x = 3200400 and y = 1600200 within the document  
23    coordinate system. In addition, we see that this shape will have a width of 1200000 EMUs and a height of  
24    1000000 EMUs.

25    The width and height set the bounding box within which the entire shape will be contained.

26     **5.9.2.2       The Shape Coordinate System**

27    Now that we have a width and height specified, we can now move into the explanation of the shape  
28    coordinate system. The shape coordinate system has both an x and y component and starts with a value of  
29    (0,0) in the upper left corner of the shape. The width and height of this coordinate system are specified by the  
30    extent of the shape, which was recently specified above, and the units are once again EMUs. This coordinate  
31    system is used to define the locations of many of the shape attributes.

32     **5.9.2.3       The Path Coordinate System**

33    The final coordinate system is the path coordinate system which also has both an x and y component and  
34    starts with a value of (0,0) in the upper left corner of the shape. Now it must be known that this coordinate  
35    system is a unique one in that its units are relative to the specified width and height of the coordinate space.  
36    The path coordinate system has exactly the same EMU dimensions as the shape coordinate system but

1 different units. While the shape coordinate system uses EMUs, the path coordinate system uses (1/width) as  
 2 the x units and (1/height) as the y units. That is if the path was specified to have a width of 2 and a height of 1,  
 3 then the path coordinate (1,1) would be equivalent to (600000,1000000) in the shape coordinate system. The  
 4 path coordinate system will be better understood later, once the path element is described.

5 Note that all dimensions and coordinates must be specified using whole numbers.

### 6 **5.9.3 Specifying a Preset Shape**

7 Within the Shape Definitions and Attributes section of DrawingML there are many pre-defined shapes that can  
 8 be used, 187 to be exact. Of course, if the user does not wish to use a preset shape there is always the option  
 9 of specifying a custom shape that will be described further in §5.9.4.

#### 10 **5.9.3.1 Defining a Preset Shape**

11 It is quite easy to specify a preset shape as that is the whole notion around presets. They are meant to solve  
 12 the most common cases of shape definition.

13 To specify a heart shape for instance the following DrawingML code can be used.

```
14 <p:sp>
15   <p:spPr>
16     <a:xfrm>
17       <a:off x="1981200" y="533400"/>
18       <a:ext cx="1143000" cy="1066800"/>
19     </a:xfrm>
20     <a:prstGeom prst="heart">
21       </a:prstGeom>
22   </p:spPr>
23 </p:sp>
```



24

25 This heart is rendered by the generating application using the custom shape code for this shape, which is fully  
 26 documented within ST\_ShapeTypes located in the reference documentation. Thus, we see that the user need  
 27 on specify the preset name to place a shape within their document.

#### 28 **5.9.3.2 Adjusting a Preset Shape**

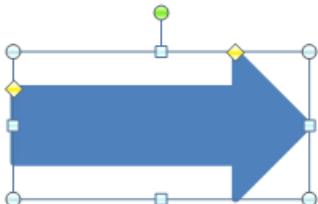
29 While specifying a preset shape is convenient and looks good most of the time. There may also be the need for  
 30 the user to adjust this preset to more closely suit the needs of their document. For this we introduce the  
 31 notion of adjust values. The preset shape is built using lines, curves and calculations, just as a custom shape

1 would be defined. To allow for the adjusting of these preset shapes we have based certain properties of  
 2 shapes on adjust values rather than concrete dimensions. This means that they can be modified which will in  
 3 turn modify the geometry of the shape.

4 A simple arrow would be specified using the following DrawingML code.

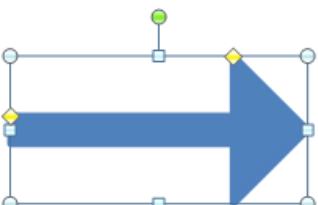
```

5   <p:sp>
6     <p:spPr>
7       <a:xfrm>
8         <a:off x="3276600" y="990600"/>
9         <a:ext cx="978408" cy="484632"/>
10        </a:xfrm>
11        <a:prstGeom prst="rightArrow">
12          <a:avLst>
13            <a:gd name="adj1" fmla="val 50000"/>
14            <a:gd name="adj2" fmla="val 50000"/>
15          </a:avLst>
16        </a:prstGeom>
17      </p:spPr>
18    </p:sp>
```



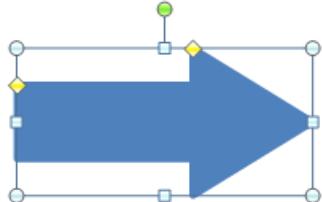
19  
 20 This will specify the basic arrow shown above which might be sufficient for the document needs of the user  
 21 but it also may not. If this standard arrow is not sufficient then the two adjust values for this shape may be  
 22 adjusted. For instance, if the body of the arrow is too large then the value for adj1 can be decreased. The  
 23 following DrawingML code would specify such a case.

24 <a:gd name="adj1" fmla="val 18553"/>



25  
 26 Similarly, if the arrow head itself was too short then the value of adj2 can be increased. The following  
 27 DrawingML code would specify such a case.

1       <a:gd name="adj2" fmla="val 81447"/>



2

3     Thus, it can be seen that while each preset is indeed a preset with a pre-defined geometry, it can be modified.  
4     Through the use of adjust values, the user is able to custom fit a preset shape to their document needs without  
5     having to specify an entirely custom shape.

6     Note that the values used here for adjust values have no real units as they are simply input parameters into  
7     the equations that make up the shape geometry. More on these equations and their parameters will be  
8     discussed in §5.9.4.2.

## 9     5.9.4     Specifying a Custom Shape

10    In addition to the specifying of a preset shape there is also the possibility of specifying a custom shape. This is  
11    accomplished by defining a geometry from a set of construction methods and applying various shape  
12    properties to this geometry. This complements preset shapes, giving the user the opportunity to specify a  
13    complete shape with any custom properties that are deemed necessary.

### 14    5.9.4.1     Defining the Geometry

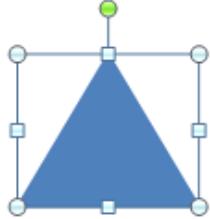
15    Just like a preset shape, a custom shape has a position and a shape bounding box that is specified by the offset  
16    and extent transform values. The shape coordinate system is defined by these values as was described in  
17    section 1.2 above. The path coordinate system is also partially defined by these in that it has its width and  
18    height set by these values. The units of the path system however are determined by the specified width and  
19    height of the path.

20    A custom shape with a single path can be specified using the following DrawingML code.

```
21    <p:sp>
22      <p:spPr>
23        <a:xfrm>
24          <a:off x="3200400" y="1600200"/>
25          <a:ext cx="1200000" cy="1000000"/>
26        </a:xfrm>
27        <a:custGeom>
28          <a:pathLst>
29            <a:path w="2" h="2">
30              <a:moveTo>
31                <a:pt x="0" y="2"/>
32              </a:moveTo>
```

```

1      <a:lnTo>
2          <a:pt x="2" y="2"/>
3      </a:lnTo>
4      <a:lnTo>
5          <a:pt x="1" y="0"/>
6      </a:lnTo>
7      <a:close/>
8      </a:path>
9  </a:pathLst>
10 </a:custGeom>
11 </p:spPr>
12 </p:sp>
```



13

14 As can be seen in the above code, the path has a width and height of 2. This means that the path coordinate  
 15 space will have units of  $(1/2 * \text{shape width})$  for x-coordinate and  $(1/2 * \text{shape height})$  for the y-coordinate.  
 16 Thus we see that a coordinate of (2,2) in the path coordinate system will be the same as (1200000,1000000)  
 17 within the shape coordinate system.

18 To define the shape path above we can see that there are a few different parts to defining this custom shape.  
 19 The first is to define the first path in what is called the path list. It should be noted that the path list can have  
 20 multiple paths in it, some filled, some not, some outlined, some not. To define the path we must specify the  
 21 width, height and thus units for this path via the following DrawingML.

```
22 <a:path w="2" h="2">
```

23 This sets up the path coordinate system for this path as was previously described. Next we need to move the  
 24 drawing cursor to the point in this path coordinate system that we wish to start drawing our shape from. The  
 25 following DrawingML does just that.

```
26 <a:moveTo>
27     <a:pt x="0" y="2"/>
28 </a:moveTo>
```

29 This will move the drawing cursor to the bottom left position (0,2) which is equivalent to (0,1000000) in the  
 30 shape coordinate system. Following this we can now start by drawing the first line in the shape via the  
 31 following line.

```

1   <a:lnTo>
2     <a:pt x="2" y="2"/>
3   </a:lnTo>
```

4 This will draw a line from the current drawing cursor position of (0,2) to (2,2) which is the bottom right corner  
 5 of the path coordinate system. This is equivalent to drawing a line from (0,1000000) to (1200000,1000000) in  
 6 the shape coordinate system. Now that we have the bottom edge of the triangle drawn we can continue to the  
 7 final edge in the shape via the following.

```

8   <a:lnTo>
9     <a:pt x="1" y="0"/>
10    </a:lnTo>
```

11 This will draw the final line that will be drawn from the current drawing cursor position of (2,2) to (1,0) which is  
 12 in the top middle of the path coordinate system. This is equivalent to drawing a line from (1200000,1000000)  
 13 to (600000,1000000) in the shape coordinate system. Now that most of the triangle has been drawn. It should  
 14 be noted that since the <close/> element is specified at the end of the path that a line will be drawn from the  
 15 last point in the path back to the first point in the path. This explains a bit of the existence of the following final  
 16 element.

17 <a:close/>

18 This finalizes the edges of the shape path being specified. Since the fill of this path is set to normal, this path  
 19 will have a fill no matter if this close tag is specified or not. However, the fact that it is specified determines  
 20 that there will be a final edge drawn between the final drawing cursor point and the path starting point. Now  
 21 that the path has been fully specified, this shape can be filled and thus be considered finished.

## 22 5.9.4.2 Adjusting the Geometry

23 Now that we have shown how a custom shape can be specified we can look at how it might be adjusted. This  
 24 adjusting is different from the typical resizing that can happen by using the shape transform elements. Using  
 25 these shape adjusting methods, a shape can be made to have many different resize/adjustment  
 26 characteristics.

## 27 5.9.4.3 Geometry Guides

28 A guide within a shape is essentially an equation with a set number of inputs and a single output. A guide is  
 29 used to calculate construction values for a shape and thus can be manipulated to govern the shape's overall  
 30 geometry.

31 An example of this can be seen in the following DrawingML.

```

32   <gdLst>
33     <gd name="y1" fmla="*/ h adj1 100"/>
34   </gdLst>
```

1 This guide will calculate it's output based on 3 input parameters and assign this output to a guide named y1.  
 2 The formula that will be used in the calculation here is the multiply divide formula. The result for this guide will  
 3 be calculated in the following manner:  $y1 = ((h * adj1) / 100)$ . After the result here is calculated, the guide y1  
 4 can be used later within the <gdlst> or <path> to calculate further values. That is it can be used as an input for  
 5 calculating another guide value. These guides then allow for a path to be based off of series of equations  
 6 rather than static path coordinate values. To use a guide in the defining of a path we would simply specify the  
 7 following within the path list.

```
8   <a:lnTo>
9     <a:pt x="2" y="y1"/>
10    </a:lnTo>
```

11 This would draw a line to the point (2,y1) where y1 is the calculated result of the guide equation shown above.  
 12 The drawing of this line will then change based on the input parameters of h and adj1 which are previously  
 13 calculated guides as well.

14 Note that while h is a previously calculated guide. It is not calculated for each shape, rather it is a built-in guide  
 15 that the generating application makes available to the shape.

#### 16 5.9.4.3.1 Adjust Handles

17 To allow for the adjusting UI of a shape we introduce the notion of an adjust handle. This adjust handle will be  
 18 linked to adjust values that will then be used as input to the guide equations defined previously. The numerical  
 19 chain described here will thus directly change the geometry of the related shape. There are two types of adjust  
 20 handles that can be specified. An XY adjust handle acts in the horizontal/vertical direction and has two related  
 21 guides, both a horizontal and a vertical respectively. A polar adjust handle acts in a polar manner and has two  
 22 related guides as well. One guide for the radial width and the other for the radial angle. An adjust handle is  
 23 specified to have an x and y coordinate as well as these adjust handles. This adjust handle can then be moved  
 24 around in a generating application's UI to adjust a pair of guides which will in turn adjust the shape being  
 25 rendered.

26 An adjust handle can be specified by the following DrawingML.

```
27   <ahXY gdRefX="adj1" minX="-2147483647" maxX="2147483647" gdRefY="adj2"
28     minY="-2147483647" maxY="2147483647">
29     <pos x="x1" y="y1"/>
30   </ahXY>
```

31 Above is an XY adjust handle that has two guide references, a min and max allowed position for both the x and  
 32 y coordinates as well as a position within the shape coordinate system where this adjust handle should be  
 33 placed.

1    5.9.4.4              Additional Properties

2    In addition to specifying the geometry for a shape and all the associated adjustments for it there are also a few  
 3    other properties that are of special significance. These properties do not act on the geometry of the shape but  
 4    instead enhance a shape so that it may be used for a more specialized task.

5    5.9.4.4.1              Connection Sites

6    As one may have experienced when trying to draw a diagram with shapes and connections between those  
 7    shapes, it is quite difficult to move a part of your diagram without entirely redrawing the connections between  
 8    shapes. For this, there is the notion of connection sites that allow for the specification of specific points within  
 9    a shape to attach connection shapes to. This allows a user to build a diagram from a set of shapes and connect  
 10   them together using connection shapes. A connection site is specified within the connection list and consists of  
 11   an x-coordinate, y-coordinate and an attachment angle.

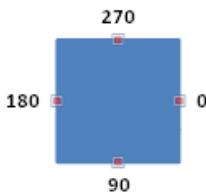
12   The following DrawingML code defines two connection sites, one at each edge of this triangle.

```
13      <a:cxnLst>
14          <a:cxn ang="10800000">
15              <a:pos x="0" y="679622"/>
16          </a:cxn>
17          <a:cxn ang="0">
18              <a:pos x="1705233" y="679622"/>
19          </a:cxn>
20      </a:cxnLst>
```



21

22   The attachment angle works by specifying an angle in 60,000ths of a degree that a connector should attach to.  
 23   The diagram below shows an actual connection point and the attachment angles that correspond to the sides  
 24   of this point. This information along with the geometry of the shape is used by the generating applications  
 25   connector routing algorithm to correctly route connectors around connected shapes.



26

27   5.9.4.4.2              Text Rectangle

28   Within each shape is a text box that allows for the attaching of text to any given shape. The text rectangle  
 29   defines where text will reside within the shape. Depending on Auto-fit options that are selected for the body

1 of text attached to this shape the text may intentionally flow outside this text rectangle. It must also be  
 2 pointed out that this text rectangle will also be the bounding box that is used to compute the geometry of a  
 3 <prstTxWarp>. The EMU dimensions of this text rectangle will be used to compute this geometry just like the  
 4 transform extent element is used to compute the actual shape.

5 The following DrawingML specifies a text rectangle within a shape.

6       <a:rect l="0" t="0" r="1200000" b="1000000"/>

7 The text rectangle shown above will have a left edge of 0 x-coordinate, top edge of 0 y-coordinate, right edge  
 8 of 1200000 x-coordinate and a bottom edge of 1000000 y-coordinate. This will effectively specify a space that  
 9 is 1200000 EMUs in width and 1000000 EMUs in height.

10 Note that the edges of this text rectangle can be set so as to allow text to be placed outside the actual  
 11 geometry of the shape.

## 12 **5.10 Pictures**

### 13 **5.10.1 Introduction**

14 This subclause provides a high-level overview of the content described in the dml-picture.xsd schema.

15 The DrawingML Picture file format is broken down into the following subjects:

- 16       • Specifying a basic picture
- 17       • Attaching properties to this picture
- 18       • Transforming this picture

19 The best way to understand the above subjects will be to cover them in the ordering above.

### 20 **5.10.2 Specifying a Basic Picture**

21 A picture can be inserted into a presentation slide by use of the picture element, pic, which is similar to the  
 22 shape element but contains some key differences that enable more complete storage of picture information.  
 23 This basic picture element should contain a blipfill and some basic non-visual picture properties.



```
1
2      <p:pic>
3          <p:nvPicPr>
4              <p:cNvPr id="4" name="St_Patrick's_Day.jpg"/>
5              <p:cNvPicPr>
6                  <a:picLocks noChangeAspect="1"/>
7              </p:cNvPicPr>
8          <p:nvPr/>
9      </p:nvPicPr>
10     <p:blipFill>
11         <a:blip r:embed="rId2"/>
12         <a:stretch>
13             <a:fillRect/>
14         </a:stretch>
15     </p:blipFill>
16     <p:spPr>
17         <a:xfrm>
18             <a:off x="1346200" y="914400"/>
19             <a:ext cx="3657600" cy="2743200"/>
20         </a:xfrm>
21         <a:prstGeom prst="rect">
22             <a:avLst/>
23         </a:prstGeom>
24         <a:noFill/>
25         <a:ln>
26             <a:noFill/>
27         </a:ln>
28     </p:spPr>
29 </p:pic>
```

### 1    5.10.3    Attaching Properties to this Picture

2    Now that the base picture has been specified, we can move on to more complicated properties, such as  
 3    recolor options and picture descriptions. In the picture below, notice that the picture that was once green has  
 4    been re-colored in a purple hue. This can be done by utilizing the duotone element, which allows for the  
 5    setting of two base colors to use for re-coloring the entire picture. The first is used to act upon the darker  
 6    regions of the picture and the second is used to act upon the lighter regions. This we can see below that black  
 7    (#000000) is indeed used below for the darker regions while accent4 (purple in this case) is used for the lighter  
 8    areas.



9

```

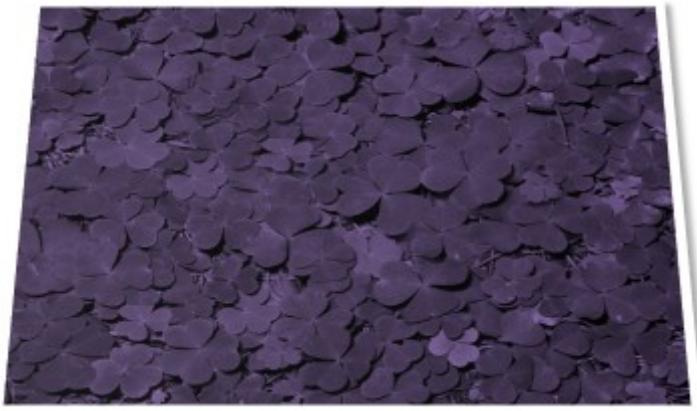
10   <p:pic>
11     <p:nvPicPr>
12       <p:cNvPr id="4" name="St_Patrick's_Day.jpg"
13         descr="This is a Saint Patrick's day picture"/>
14       <p:cNvPicPr>
15         <a:picLocks noChangeAspect="1"/>
16       </p:cNvPicPr>
17       <p:nvPr/>
18     </p:nvPicPr>
19     <p:blipFill>
20       <a:blip r:embed="rId2">
21         <a:duotone>
22           <a:srgbClr val="000000"/>
23           <a:schemeClr val="accent4"/>
24         </a:duotone>
25       </a:blip>
26       <a:stretch>
27         <a:fillRect/>
28       </a:stretch>
29     </p:blipFill>
```

```

1   <p:spPr>
2     <a:xfrm>
3       <a:off x="1346200" y="914400"/>
4       <a:ext cx="3657600" cy="2743200"/>
5     </a:xfrm>
6     <a:prstGeom prst="rect">
7       <a:avLst/>
8     </a:prstGeom>
9     <a:noFill/>
10    <a:ln>
11      <a:noFill/>
12    </a:ln>
13  </p:spPr>
14 </p:pic>
```

#### 5.10.4 Transforming this Picture

Now that both basic properties and additional picture properties have been specified, we can begin incorporating shape properties. Below is the same picture as described above, with 3D camera perspective applied along with a simple shadow and a white outline. These shape properties are the same that can be applied to a shape element. One picture-specific difference can be seen here with the border around the picture. Instead of the border growing both inward and outward, it only grows outward.



```

21
22 <p:pic>
23   <p:nvPicPr>
24     <p:cNvPr id="4" name="St_Patrick's_Day.jpg"
25       descr="This is a Saint Patrick's day picture"/>
26     <p:cNvPicPr>
27       <a:picLocks noChangeAspect="1"/>
28     </p:cNvPicPr>
29     <p:nvPr/>
30   </p:nvPicPr>
```

```

1   <p:blipFill>
2     <a:blip r:embed="rId2">
3       <a:duotone>
4         <a:srgbClr val="000000"/>
5         <a:schemeClr val="accent4"/>
6       </a:duotone>
7     </a:blip>
8     <a:stretch>
9       <a:fillRect/>
10    </a:stretch>
11  </p:blipFill>
12  <p:spPr>
13    <a:xfrm>
14      <a:off x="1346200" y="914400"/>
15      <a:ext cx="3657600" cy="2743200"/>
16    </a:xfrm>
17    <a:prstGeom prst="rect">
18      <a:avLst/>
19    </a:prstGeom>
20    <a:noFill/>
21    <a:ln w="57150">
22      <a:solidFill>
23        <a:schemeClr val="bg1"/>
24      </a:solidFill>
25    </a:ln>
26    <a:effectLst>
27      <a:outerShdw blurRad="50800" dist="50800" dir="2700000" algn="t1"
28        rotWithShape="0">
29        <a:srgbClr val="7D7D7D">
30          <a:alpha val="65000"/>
31        </a:srgbClr>
32      </a:outerShdw>
33    </a:effectLst>
34    <a:scene3d>
35      <a:camera prst="perspectiveRelaxedModerately"/>
36      <a:lightRig rig="threePt" dir="t">
37        <a:rot lat="0" lon="0" rev="18900000"/>
38      </a:lightRig>
39    </a:scene3d>
40  </p:spPr>
41 </p:pic>

```

42 **End of informative text.**

## 1    5.11 WordprocessingML Drawing

2    Within a WordprocessingML document, it is possible to include graphical DrawingML objects:

- 3        • Charts
- 4        • Diagrams
- 5        • Locked Canvases
- 6        • Pictures

7    When these objects are present in a word processing document, it is necessary to include information that  
8    specifies how the objects are to be positioned relative to the paginated document.

9    The WordprocessingML Drawing namespace acts in this capacity, specifying all information necessary to  
10   anchor and display DrawingML objects within a word processing document.

11   Consider a DrawingML picture that is to displayed in the center of the printed page on which it appears,  
12   modifying the flow of text as necessary. This object would be specified as follows:

```
13      <w:r>
14          <w:drawing>
15              <wp:anchor relativeHeight="10" allowOverlap="true">
16                  <wp:positionH relativeFrom="margin">
17                      <wp:align>center</wp:align>
18                  </wp:positionH>
19                  <wp:positionV relativeFrom="margin">
20                      <wp:align>center</wp:align>
21                  </wp:positionV>
22                  <wp:extent cx="2441542" cy="1828800"/>
23                  <wp:wrapSquare wrapText="bothSides"/>
24                  <a:graphic>
25                      ...
26                  </a:graphic>
27              </wp:anchor>
28          </w:drawing>
29      </w:r>
```

30   The anchor element specifies that this object is not positioned in-line with text, and its child elements specify  
31   that the object is centered on the page horizontally and vertically, and that text can wrap around it in a square.

### 32    5.11.1 Object Anchoring

33   When the WordprocessingML Drawing namespace is used to anchor a DrawingML object within a document,  
34   that object can be anchored in one of two ways:

- 35        • In line with text - The object is displayed within the regular text stream (modifying line height and so  
36              on to accommodate it).

- Floating – The object is positioned absolutely or relatively within the document and text flow is modified as needed around it.

### 3 5.11.2 Text Wrapping

4 Aside from positioning data, WordprocessingML Drawing also needs to specify how text flows around the  
5 object. There are five different types of text wrapping which can be applied to floating objects present in  
6 WordprocessingML documents:

- 7     • In Front/Behind Text - In this type of text wrapping, the drawing object is positioned on the document  
8       and text is not displaced around it.

On the Insert tab, the galleries include items that are designed to coordinate with the overall look of your document. You can use these galleries to insert tables, headers, footers, lists, cover pages, and other document building blocks. When you insert charts, or diagrams, they also coordinate with your current document look.

You can easily change the format of selected text from the Quick Access toolbar, the other controls on the Home tab, or using a format that's part of the theme or using a style that's part of the current theme.

To change the overall look of your document, click the Themes gallery or the Quick Styles gallery on the ribbon, and then click the More button to open the Quick Style Set dialog box. You can also click the Reset button on the Page Layout tab to reset all elements on the Page Layout tab. To change the looks available in the Quick Styles gallery, click the Change Current Quick Style Set command. Both the Themes gallery and the Quick Styles gallery provide a Reset command so that you can always restore the look of your document to the original contained in your current template.

- Square Wrapping - In this type of text wrapping, the drawing object is positioned on the document and a rectangle is stored within the file format to determine the wrapping extents.

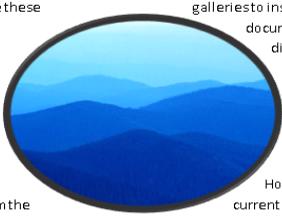
On the Insert tab, the galleries include items that are designed to coordinate with the overall look of your document. You can use these galleries to insert tables, headers, footers, lists, cover pages, and other document building blocks. When you create pictures, charts, or diagrams, they also coordinate with your current document look.

You can easily change the document text by selected text from the Home tab. You can also the other controls on the offer a choice of using the or using a format that you

To change the overall look of your document, choose new Theme elements or change the looks available in the Quick Style gallery, use

- Tight Wrapping - In this type of text wrapping, a wrapping polygon is created and stored in the WordprocessingML document, and this polygon determines how text wraps around the left and right sides of the drawing object.

On the Insert tab, the galleries include items that are designed to coordinate with the overall look of your document. You can use these lists, cover pages, and other create pictures, charts, or with your current



galleries to insert tables, headers, footers, document building blocks. When you diagrams, they also coordinate document look.

You can easily change the the document text by selected text from the Home tab. You can also the other controls on the choice of using the look from the you specify directly.

formatting of selected text in choosing a look for the Quick Styles gallery on the Home tab. Most controls offer a current theme or using a format that

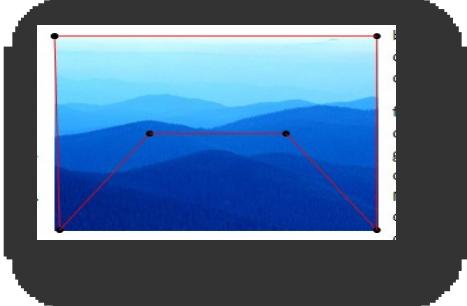
1 To change the overall look of your document, choose new Theme elements on the Page Layout

2

- 3 • Through Wrapping - In this type of text wrapping, a wrapping polygon is created just like with tight wrapping, but any indents in the wrap polygon can be filled with text in this case.

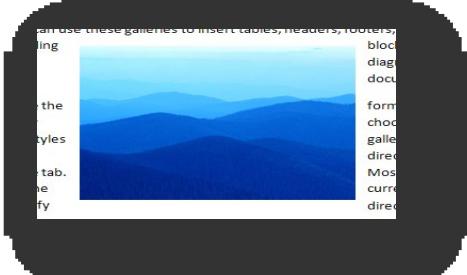
4

5 If the wrapping polygon looks like the following:



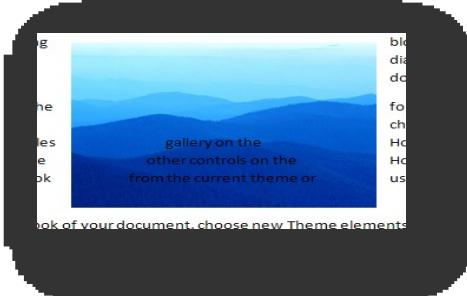
6

7 Tight wrapping would look like this:



8

9 While through wrapping would look like this:



10

11 In the latter case, notice that text fills in the 'indentation' within the wrapping polygon.

12

- 13 • Top and Bottom Wrapping - In this type of text wrapping, text cannot wrap around either side of the object, and shall only restart below the bottom edge of the document.

1

On the Insert tab, the galleries include items that are designed to coordinate with the overall look of



2

your document. You can use these galleries to insert tables, headers, footers, lists, cover pages, and other document building blocks. When you create pictures, charts, or diagrams, they also coordinate with your current document look.

3

## 5.12 SpreadsheetML Drawing

### 5.12.1 Introduction

This subclause provides a high-level overview of the content described in the dml-spreadsheetDrawing.xsd schema. The elements in this schema specify how drawing elements are to be described within a spreadsheet. For example, suppose we want to specify a rectangle drawing shape within a worksheet to look like the following:

	A	B	C	D	E	F
1						
2	○		○			
3						
4						
5						
6	■			■		
7						
8						
9						
10	○		○			
11						
12						
13						
14						
15						

### 5.12.2 Overview

The elements that specify the drawing objects within a worksheet are all located within its respective drawing XML file. This file is located under the "drawings" folder inside the spreadsheet file. For example, if the drawing element is located on Worksheet 1, then the specifications for the said element would be located in the file \xl\drawings\drawing1.xml.

### 5.12.3 Worksheet Drawings

Within the drawing\*.xml file is contained a single worksheet drawing wsDr element, which is the parent element for all the drawing elements. Its child specifies the anchoring properties of the drawing elements. It is

1 within this element that the main specifications for the drawing elements are located. For example in the  
 2 above screenshot with a simple shape located on the worksheet, the XML for this would look like:

```

3   <xdr:wsDr>
4     <xdr:absoluteAnchor>
5       <xdr:pos x="2162175" y="1743075"/>
6       <xdr:ext cx="1238250" cy="1314450"/>
7       ...
8       </xdr:sp>
9     </xdr:absoluteAnchor>
10    </xdr:wsDr>
```

11 In this SpreadsheetDrawingML code there is a single drawing specified almost exactly as it would within the  
 12 regular DrawingML framework. However the SpreadsheetDrawingML wrapper that is used allows for the  
 13 specifying of spreadsheet specific properties in addition to the normal drawing properties.

#### 14 5.12.3.1 Anchoring Types

15 To define a drawing within a spreadsheet an anchoring type must be chosen. There are three different  
 16 anchoring types allowed for use within a spreadsheet: Absolute Anchoring, One Cell Anchoring, and Two Cell  
 17 Anchoring. Each of these types is described in the following subclauses.

##### 18 5.12.3.1.1 Absolute Anchoring

19 Absolute Anchoring describes the placement of the drawing within the spreadsheet based upon absolute  
 20 coordinates. This positioning information includes both position coordinates and extent coordinates. The  
 21 absoluteAnchor element is what specifies this anchoring behavior and a sample usage is shown below.

```

22   <xdr:absoluteAnchor>
23     <xdr:pos x="2162175" y="1552575"/>
24     <xdr:ext cx="1238250" cy="1123950"/>
25     ...
26     </xdr:sp>
27   </xdr:absoluteAnchor>
```

28 In this example, there is a single shape specified using absolute anchoring as its anchoring method.

##### 29 5.12.3.1.2 One Cell Anchoring

30 One Cell Anchoring describes the placement of the drawing within the spreadsheet based upon offsets as well  
 31 as a specified column and row. The offset is always in reference to the specified anchor cell and acts to offset  
 32 the shape object from being exactly on top of the anchor cell. The offset information determines the actual  
 33 placement of the drawing within the spreadsheet while the row and column are used to specify to which cell  
 34 the drawing should be anchored. Thus, if the anchor cell changes positions then the drawing can be moved as  
 35 well. The oneCellAnchor element is what specifies this anchoring behavior and a sample usage is shown  
 36 below.

```

1   <xdr:oneCellAnchor>
2     <xdr:from>
3       <xdr:col>3</xdr:col>
4       <xdr:colOff>333375</xdr:colOff>
5       <xdr:row>8</xdr:row>
6       <xdr:rowOff>28575</xdr:rowOff>
7     </xdr:from>
8     <xdr:ext cx="1238250" cy="1123950"/>
9     ...
10    </xdr:sp>
11  </xdr:oneCellAnchor>
```

12 In this example, there is a single shape specified using one cell anchoring as its anchoring method.

### 13 5.12.3.1.3 Two Cell Anchoring

14 Two Cell Anchoring describes the placement of the drawing within the spreadsheet based upon offsets as well  
 15 as a specified columns and rows. The offset is always in reference to the specified anchor cell and acts to offset  
 16 the shape object from being exactly on top of the anchor cell. The offset information determines the actual  
 17 placement of the drawing within the spreadsheet while the rows and columns are used to specify the cells to  
 18 which the drawing should be anchored and upon which the resizes is based. For instance, if the anchor cell  
 19 changes positions then the drawing can be moved. Likewise, if the anchor cells behind the shape grow, then  
 20 the shape can grow as well. The twoCellAnchor element is what specifies this anchoring behavior and a  
 21 sample usage is shown below.

```

22   <xdr:twoCellAnchor>
23     <xdr:from>
24       <xdr:col>3</xdr:col>
25       <xdr:colOff>447675</xdr:colOff>
26       <xdr:row>8</xdr:row>
27       <xdr:rowOff>28575</xdr:rowOff>
28     </xdr:from>
29     <xdr:to>
30       <xdr:col>5</xdr:col>
31       <xdr:colOff>466725</xdr:colOff>
32       <xdr:row>14</xdr:row>
33       <xdr:rowOff>9525</xdr:rowOff>
34     </xdr:to>
35     ...
36     </xdr:sp>
37   </xdr:twoCellAnchor>
```

38 In this example, there is a single shape specified using two cell anchoring as its anchoring method.

## 1    5.13    Charts

### 2    5.13.1    Overview

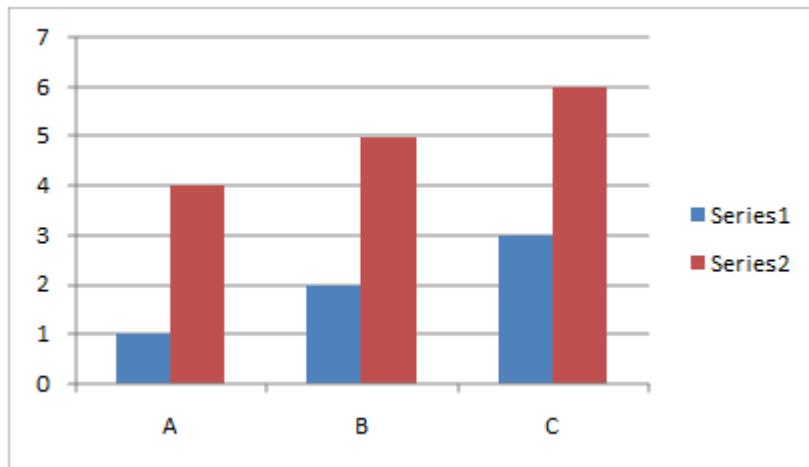
3    Charts provide a great way to visualize information by displaying a graphical representation of the data. The  
4    chart XML files can be reused and shared among different applications, such as a spreadsheet, presentation,  
5    and word processing.

6    Charts come in many different types and flavors, and this document provides a basic overview of both the  
7    different types of charts as well as the XML that is used to generate them.

8    Applications may allow many different runtime behaviors for charts, such as rules for displaying them. This  
9    clause and its corresponding reference material define only the XML that is needed to store and generate the  
10   charts, and do not dictate any runtime behaviors.

#### 11    5.13.1.1    Basic Chart Types

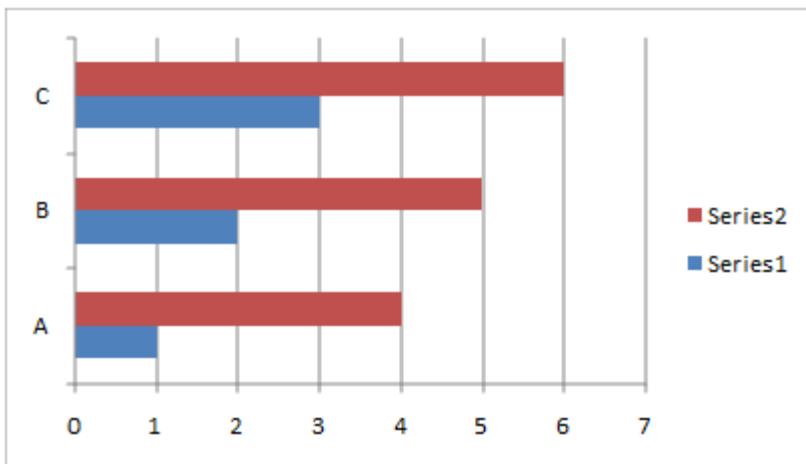
12    There are 10 basic chart types. Below are examples of each basic type:



13

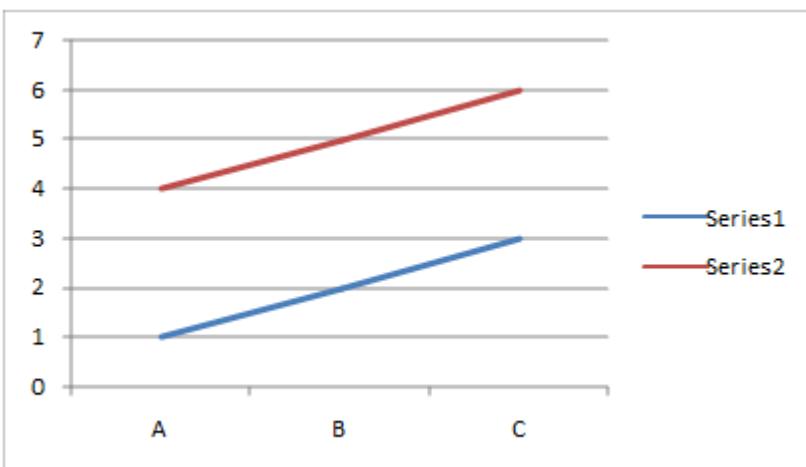
14    Column Chart (shown above)

15



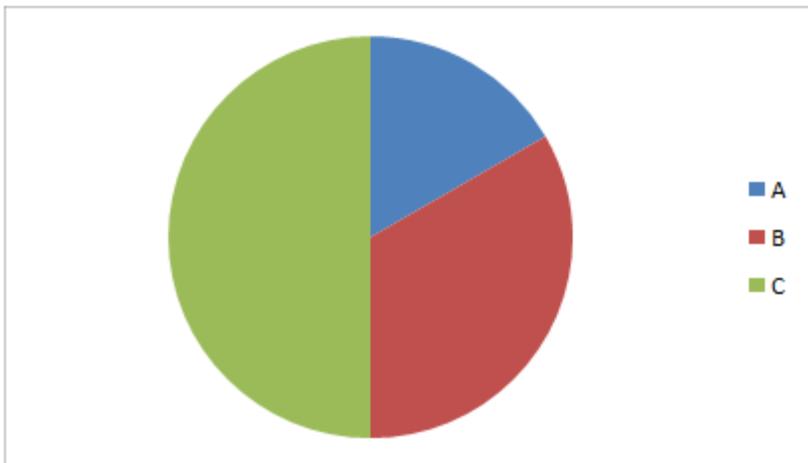
1  
2 Bar Chart (shown above)

3



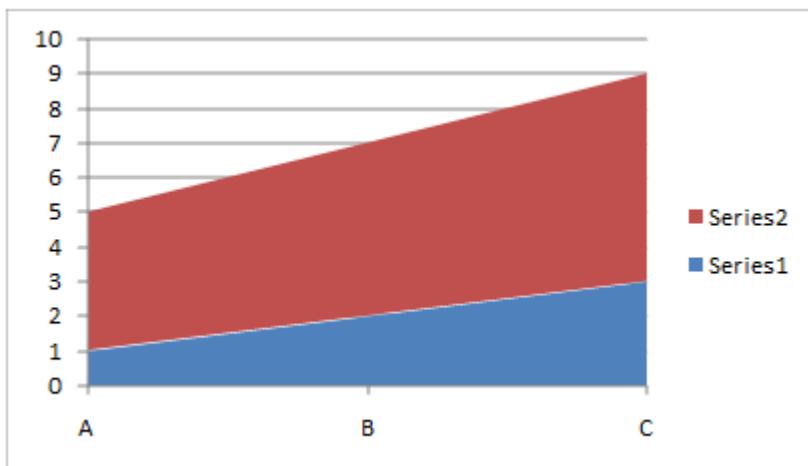
4  
5 Line Chart (shown above)

6



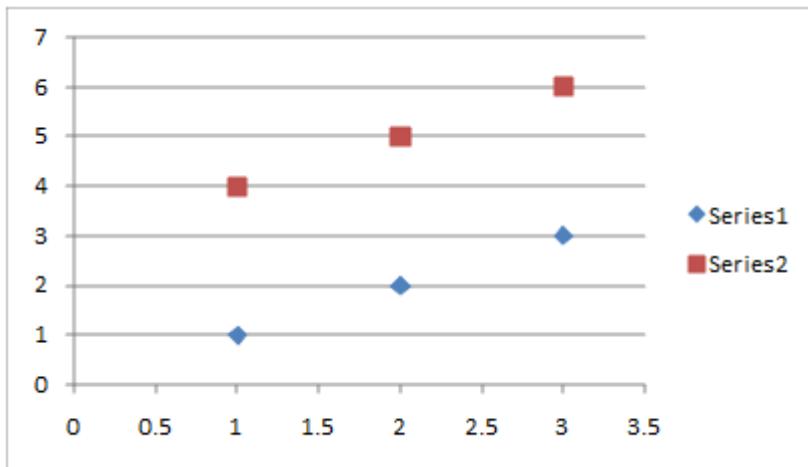
1  
2 Pie Chart (shown above)

3



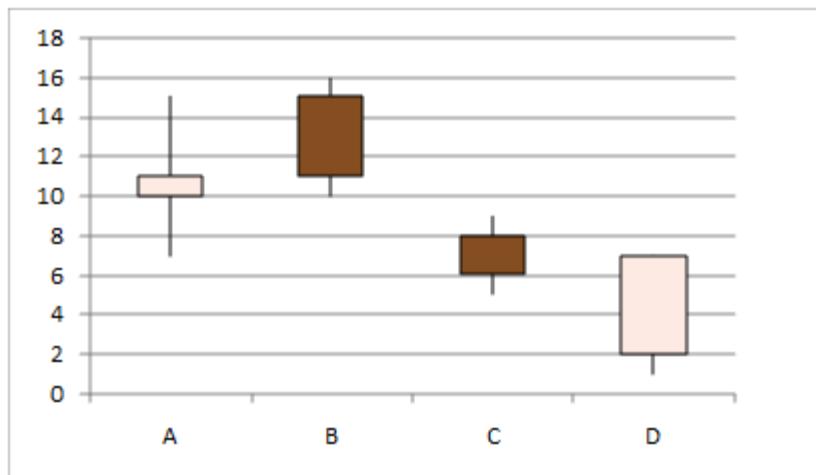
4  
5 Area Chart (shown above)

6



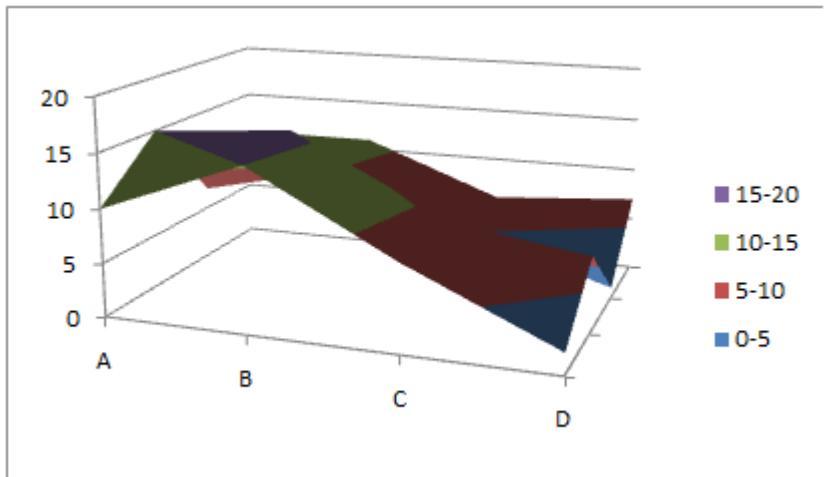
1  
2 Scatter Chart (shown above)

3



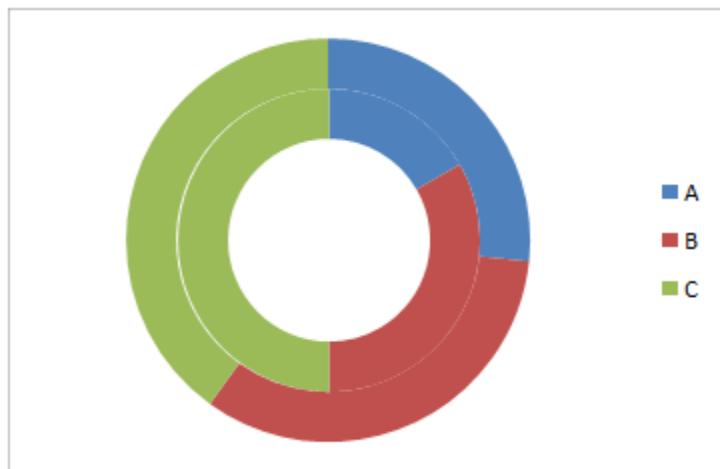
4  
5 Stock Chart (shown above)

6



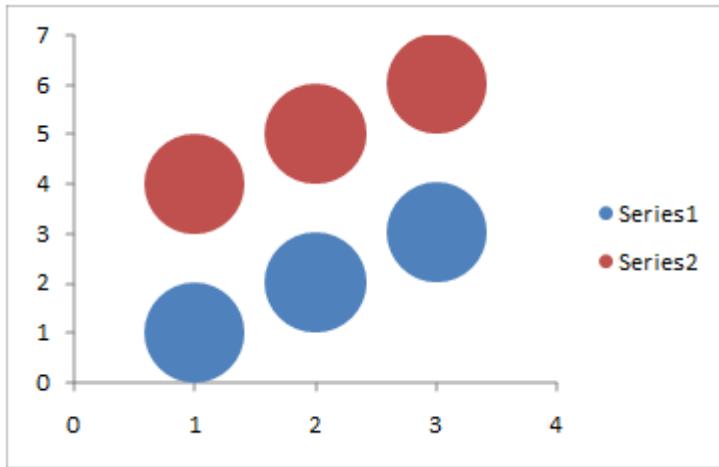
1  
2 Surface Chart (shown above)

3



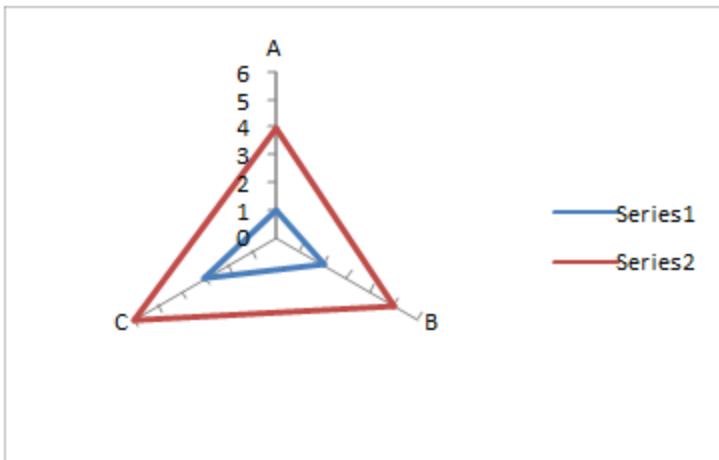
4  
5 Doughnut Chart (shown above)

6



1  
2 Bubble Chart (shown above)

3

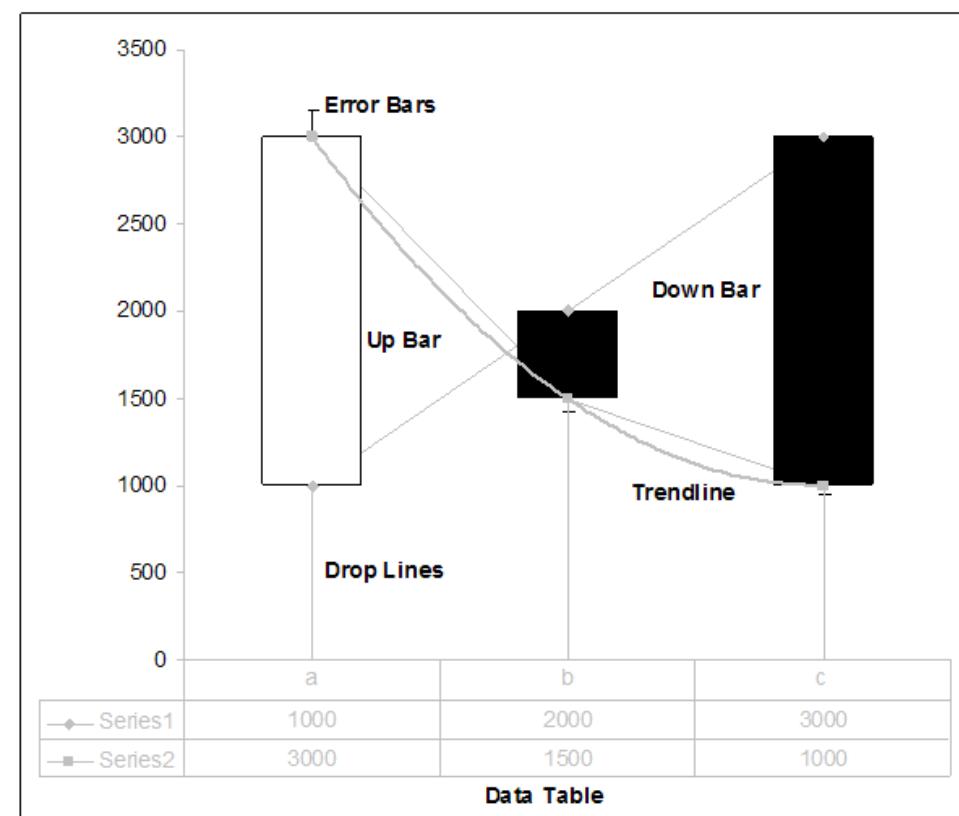
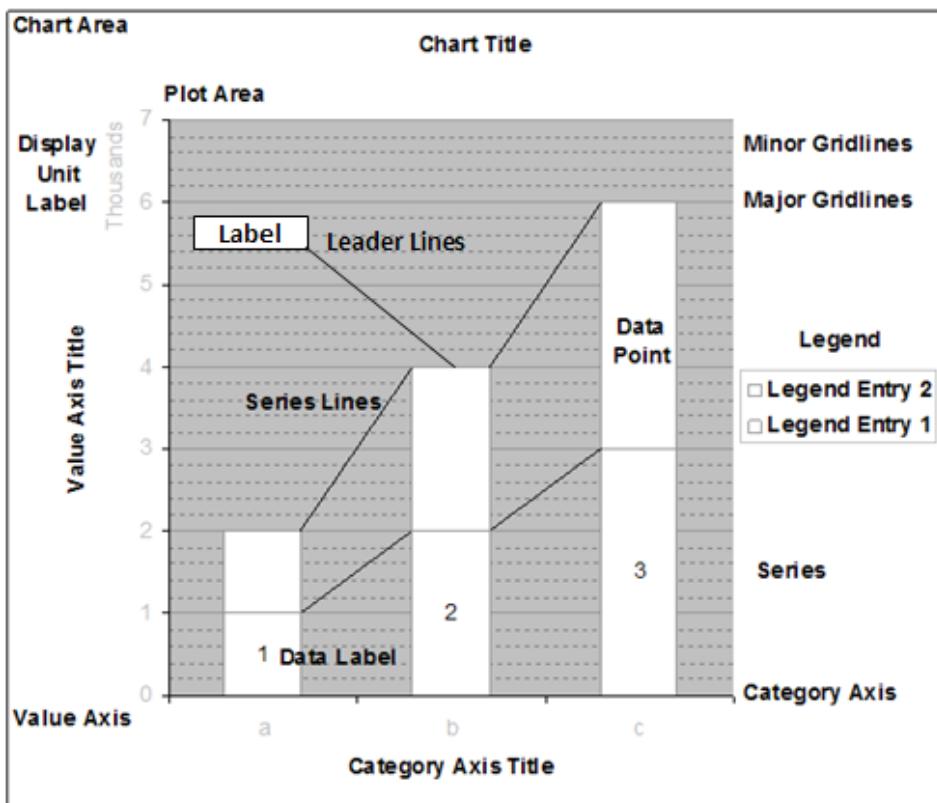


4  
5 Radar Chart (shown above)

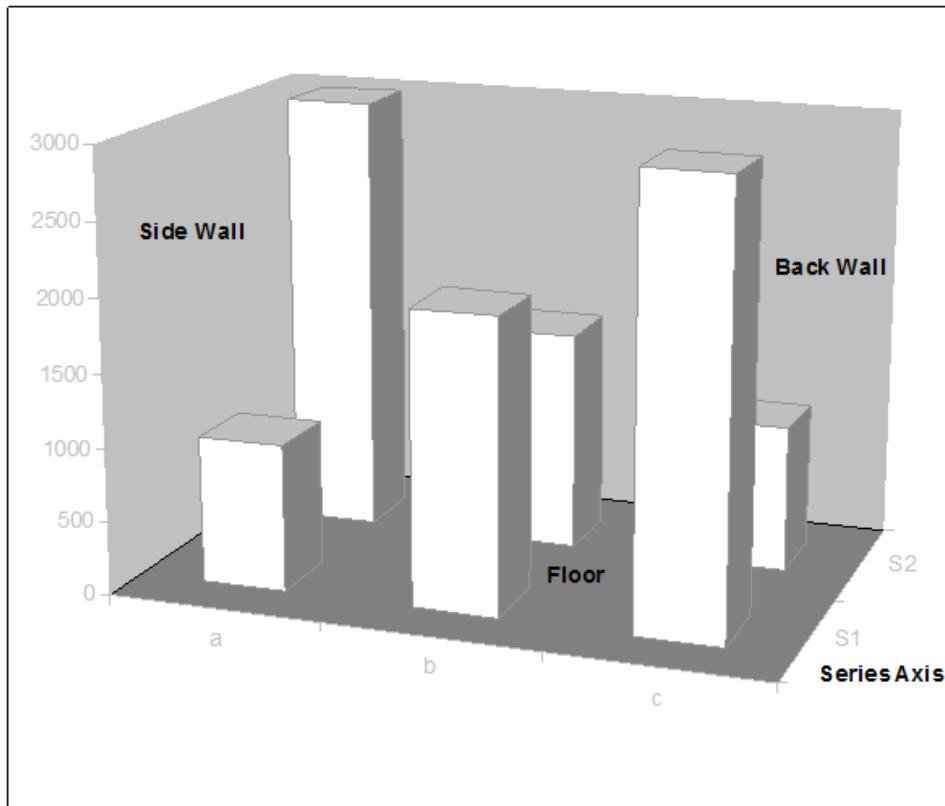
6

### 7 5.13.1.2 Basic Chart Components

8 Here are some diagrams that label the different individual components of a chart. Some chart components,  
9 such as drop lines, are only shown on certain types of charts.



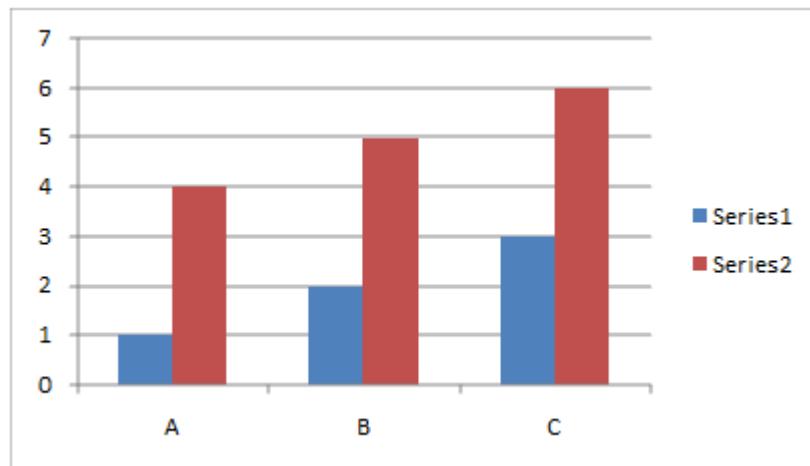
2



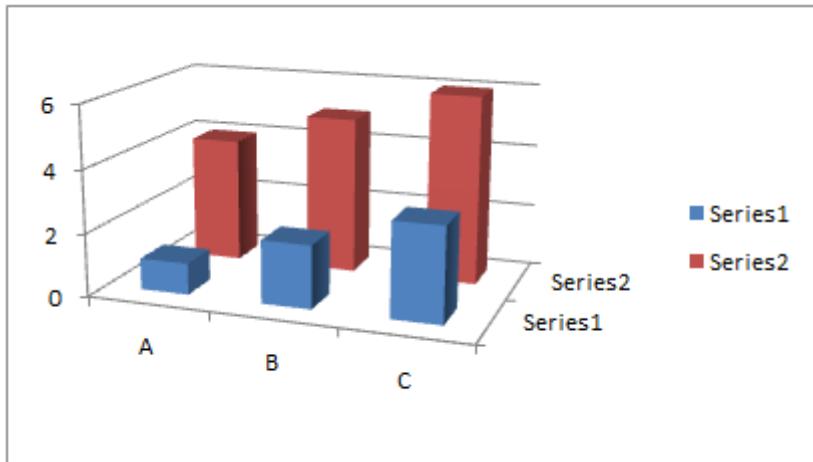
1

### 5.13.1.3 3D Charts

Most chart types also have three-dimensional representations. 3D charts have extra properties to describe depth, floor, or walls, as well as some other rendering effects. Below is a 2D column chart shown with its 3D counterpart.



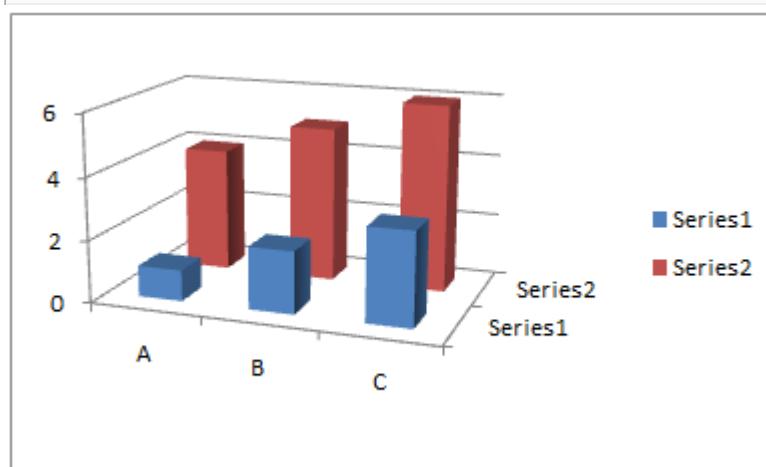
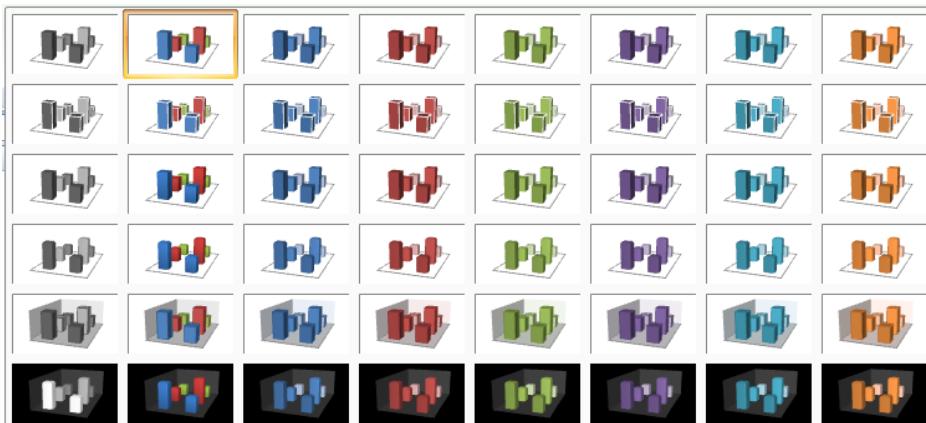
6

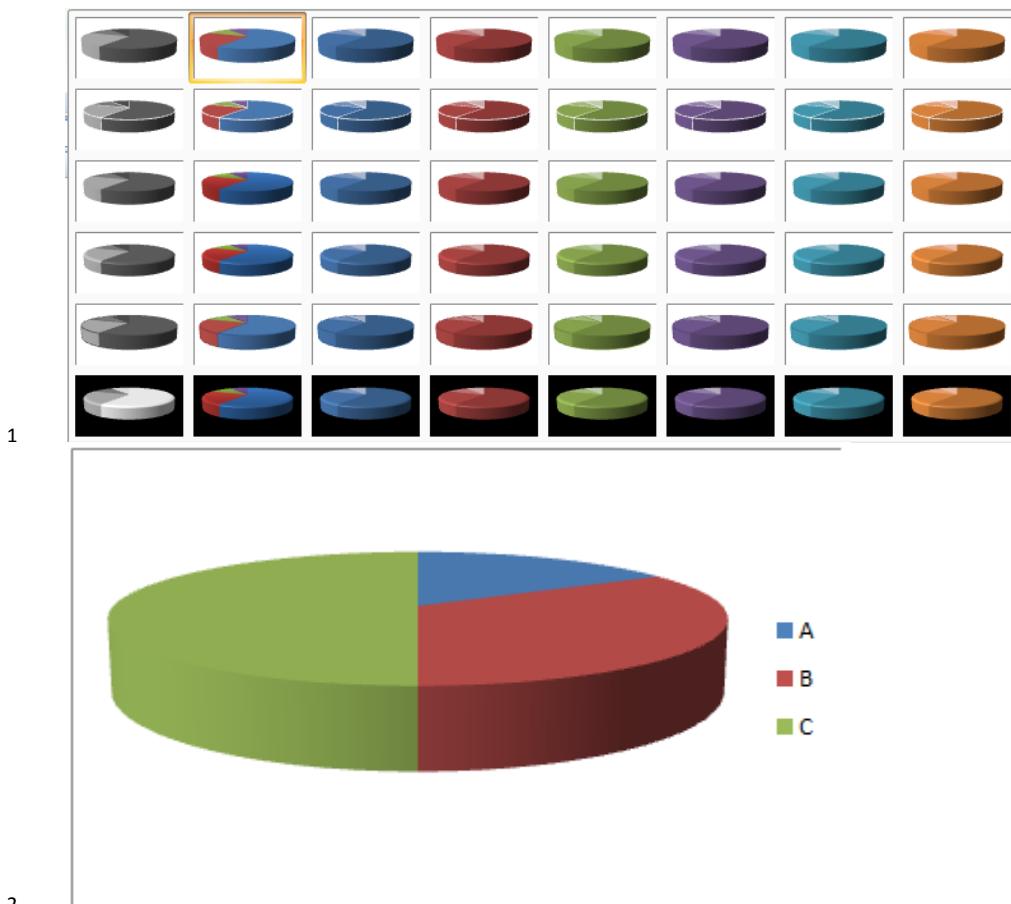


1 |

#### 2 5.13.1.4 Chart Styles

3 Charts may have different styles applied to them. This is essentially just a coordinated set of coloring and  
4 formatting that is applied across an entire chart and all its elements. Styles allow a quick and easy way to  
5 coordinate the look and feel of a chart with the rest of the document. Below is a rendering of a column chart  
6 and pie chart showing the same data, with the same style applied. There is also a shot of many different types  
7 of styles. It is shown for both charts to illustrate how the idea of a style can apply, and be consistent, across  
8 different chart types.





### 5.13.2 XML Overview

#### 5.13.2.1 Relationships

A Drawing XML part contains a chart element, which expresses a relationship ID. This ID is referenced by the drawing.rels part, which points to the corresponding chart XML part. Chart XML contains the core definition of the chart.

#### 5.13.2.2 Chart

Different chart types can have many different components defined in the XML, and not all are shown here. For many charts though, at a very high level, the chart XML is composed of the following pieces:

```

1   <chartSpace>
2     <chart>
3       <view3D>
4       <plotArea>
5         <layOut>
6         <barChart>
7           <cat>
8             <val>
9             <catAx>
10            <valAx>
11         </plotArea>
12         <legend>
13       </chart>
14     <printSettings>
15   </chartSpace>

```

16 chartSpace is the root node, which contains an element defining the chart, and an element defining the print  
 17 settings for the chart.

18 chart is the root element for the chart. If the chart is a 3D chart, then a view3D element is contained, which  
 19 specifies the 3D view. It then has a plot area, which defines a layout and contains an element that corresponds  
 20 to, and defines, the type of chart.

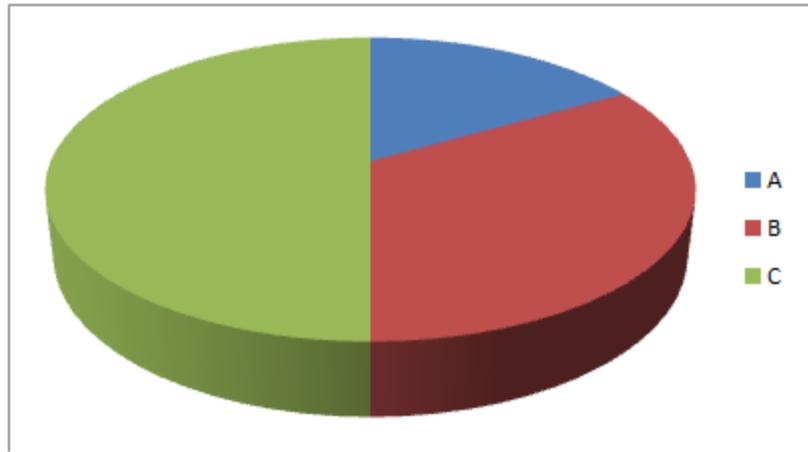
21 The element that defines the type of chart, barChart in this case, also specify caches for both category axis  
 22 data (which is really just strings for the categories), as well as the for the numbers, or values, shown on the  
 23 chart. The cat element defines the string cache for the category axis, and the val element defines the number  
 24 caches.

25 Depending on the type of the chart, the plot area may optionally contain elements that define the axes—such  
 26 as the value axis and category axis—or shape properties. In the example above, the category axis, catAx, and  
 27 value axis, valAx are defined. These define things like positioning, orientation, label position, and tick marks  
 28 for the axis. The actual strings and data that the axis corresponds to are defined by the cat and val elements.

29 Lastly, the chart element contains a legend element which defines the chart legend.

### 30 5.13.3 Example

31 The XML in this clause defines the following 3D chart:



1

2 For this example, the xml for the chart element will be shown in detail. The xml for the chart element follows:

```
3 <c:chart>
4   <c:view3D>
5     <c:rotX val="30"/>
6     <c:perspective val="30"/>
7   </c:view3D>
8   <c:plotArea>
9     <c:layout>
10    <c:lastLayoutOuter>
11      <c:x val="4.5"/>
12      <c:y val="4.5"/>
13      <c:w val="324.75"/>
14      <c:h val="206.25"/>
15    </c:lastLayoutOuter>
16    <c:lastLayout>
17      <c:x val="10.5"/>
18      <c:y val="10.5"/>
19      <c:w val="312.75"/>
20      <c:h val="194.25"/>
21    </c:lastLayout>
22  </c:layout>
```

```
1 <c:pie3DChart>
2   <c:varyColors val="1"/>
3   <c:ser>
4     <c:idx val="0"/>
5     <c:order val="0"/>
6     <c:cat>
7       <c:strRef>
8         <c:f>Sheet1!$A$1:$C$1</c:f>
9         <c:strCache>
10        <c:pt idx="0">
11          <c:v>A</c:v>
12        </c:pt>
13        <c:pt idx="1">
14          <c:v>B</c:v>
15        </c:pt>
16        <c:pt idx="2">
17          <c:v>C</c:v>
18        </c:pt>
19      </c:strCache>
20    </c:strRef>
21  </c:cat>
22  <c:val>
23    <c:numRef>
24      <c:f>Sheet1!$A$2:$C$2</c:f>
25      <c:numCache>
26        <c:pt idx="0">
27          <c:v>1</c:v>
28        </c:pt>
29        <c:pt idx="1">
30          <c:v>2</c:v>
31        </c:pt>
32        <c:pt idx="2">
33          <c:v>3</c:v>
34        </c:pt>
35      </c:numCache>
36    </c:numRef>
37  </c:val>
38 </c:ser>
39 </c:pie3DChart>
```

```

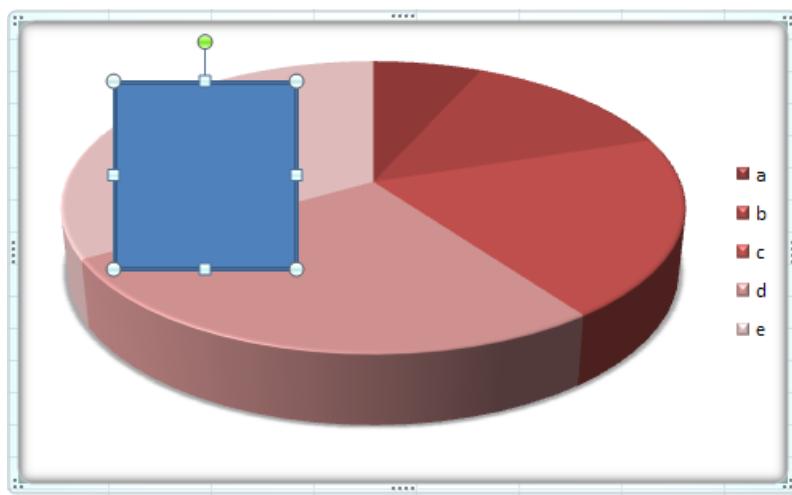
1   <c:spPr>
2     <a:noFill/>
3     <a:ln w="25400">
4       <a:noFill/>
5     </a:ln>
6   </c:spPr>
7 </c:plotArea>
8 <c:legend>
9   <c:legendPos val="r"/>
10  <c:layout>
11    <c:lastLayout>
12      <c:x val="333.75"/>
13      <c:y val="81.75"/>
14      <c:w val="19.5"/>
15      <c:h val="51.75"/>
16    </c:lastLayout>
17  </c:layout>
18 </c:legend>
19   <c:plotVisOnly val="1"/>
20 </c:chart>

```

## 5.14 Chart Drawing

### 5.14.1 Introduction

This subclause provides a high-level overview of the content described in the dml-chartDrawing.xsd schema. The elements in this schema specify how drawing elements are to be described within a chart. For example, suppose we want to specify a rectangle drawing shape within a chart to look like the following:



## 1    5.14.2    Overview

2    The elements that specify the drawing objects within a chart are all located within its respective drawing XML  
 3    file. This file is located under the "drawings" folder inside the spreadsheet file.

## 4    5.14.3    Chart Drawings

5    Within the drawing\*.xml file there is a single drawing file that contains the userShapes element. This element  
 6    is the parent element for all the drawing elements within a single chart. Its child specifies the anchoring  
 7    properties of the drawing elements. It is within this element that the main specifications for the drawing  
 8    elements are located. For example in the above screenshot with a simple shape located in the chart, the XML  
 9    for this would look like:

```
10   <c:userShapes>
11     <cdr:relSizeAnchor>
12       <cdr:from>
13         <cdr:x>0.125</cdr:x>
14         <cdr:y>0.13194</cdr:y>
15       </cdr:from>
16       <cdr:to>
17         <cdr:x>0.36042</cdr:x>
18         <cdr:y>0.53472</cdr:y>
19       </cdr:to>
20       ...
21     </cdr:sp>
22   </cdr:relSizeAnchor>
23 </c:userShapes>
```

24    In the ChartDrawingML code above, there is a single drawing specified almost exactly as it would within the  
 25    regular DrawingML framework. However, the ChartDrawingML wrapper that is used allows for the specifying  
 26    of chart specific properties in addition to the normal drawing properties. The most interesting of these are the  
 27    two anchoring types that define the placement behavior of a drawing within a chart.

### 28    5.14.3.1    Anchoring Types

29    To define a drawing within a chart an anchoring type must be chosen. There are two different anchoring types  
 30    allowed for use within a chart: Absolute Anchoring and Relative Anchoring. These two types are described in  
 31    the following two subclauses.

#### 32    5.14.3.1.1    Absolute Size Anchoring

33    Absolute Anchoring describes the placement of the drawing within the chart based upon absolute chart  
 34    coordinates. The absSizeAnchor element specifies anchoring behavior, using percentage-based position  
 35    coordinates for the anchor location and extent coordinates (in EMUs) for drawing objects, as shown in the  
 36    example below.<cdr:absSizeAnchor>

```
37   <cdr:from>
38     <cdr:x>0.125</cdr:x>
```

```

1   <cdr:y>0.13194</cdr:y>
2   </cdr:from>

3       <cdr:ext cx="1238250" cy="1123950"/>
4       ...
5       </cdr:sp>
6   </cdr:absSizeAnchor>
```

7 In this example, there is a single shape specified using absolute anchoring as its anchoring method.

#### 8 5.14.3.1.2 Relative Size Anchoring

9 Relative Anchoring describes the placement of the drawing within the chart based upon relative chart  
10 coordinates. For instance, if the chart increases in size then the shape will grow as well. This positioning  
11 information includes from and to elements which specify a percentage-based coordinate within the chart  
12 bounding box. The relSizeAnchor element is what specifies this anchoring behavior and a sample usage is  
13 shown below.

```

14   <cdr:relSizeAnchor>
15     <cdr:from>
16       <cdr:x>0.125</cdr:x>
17       <cdr:y>0.13194</cdr:y>
18     </cdr:from>
19     <cdr:to>
20       <cdr:x>0.36042</cdr:x>
21       <cdr:y>0.53472</cdr:y>
22     </cdr:to>
23     ...
24     </cdr:sp>
25   </cdr:relSizeAnchor>
```

26 In this example, there is a single shape specified using relative anchoring as its anchoring method.

### 27 5.15 Diagrams

#### 28 5.15.1 Introduction

29 This clause provides a high-level overview of the content described in the following schemas: dml-  
30 diagramTypes.xsd, dml-diagramDataModel.xsd, dml-diagramStyleDefinition.xsd, dml-  
31 diagramLayoutVariables.xsd, dml-diagramElementPropertySet.xsd, dml-diagramColorTransform.xsd, and dml-  
32 diagramDefinition.xsd.

33 The DrawingML diagram file format is broken down into the following subjects:

- 34 • Data Model
- 35 • Colors
- 36 • Quick Styles

- 1        • Layout
- 2     The best way to understand the above subjects will be to cover them in the ordering above. The seven  
 3     schemas can be grouped into the subjects as seen in table 1 below.

Data	Colors	Quick Styles	Layout
dml-diagramDataModel.xsd	dml-diagramColorTransform.xsd	dml-diagramStyleDefinition.xsd	dml-diagramTypes.xsd
dml-diagramElementPropertySet.xsd			dml-diagramDefinition.xsd
			dml-diagramLayoutVariables.xsd
			dml-diagramElementPropertySet.xsd

4     Table 1: DrawingML schemas grouped by subject.

## 5     5.15.2     Element Property Set

6     The schema dml-diagramElementPropertySet.xsd defines a complex type, CT\_ElemPropSet, which is a catch-all  
 7     for holding element properties and customizations, and is used throughout certain complex types in  
 8     DrawingML. This type contains many properties, and these are explained in subsequent subclauses. The  
 9     definition of CT\_ElemPropSet is as follows:

```
10    <xsd:complexType name="CT_ElemPropSet">
11      <xsd:sequence>
12       <xsd:element name="presLayoutVars"
13          type="CT_LayoutVariablePropertySet" minOccurs="0"
14          maxOccurs="1" />
15       <xsd:element name="style" type="a:CT_ShapeStyle"
16          minOccurs="0" maxOccurs="1" />
17      </xsd:sequence>
18      <xsd:attribute name="presAssocID" type="ST_ModelId" use="optional" />
19      <xsd:attribute name="presName" type="xsd:string" use="optional" />
20      <xsd:attribute name="presStyleLbl" type="xsd:string" use="optional" />
21      <xsd:attribute name="presStyleIdx" type="xsd:int" use="optional" />
22      <xsd:attribute name="presStyleCnt" type="xsd:int" use="optional" />
```

```

1   <xsd:attribute name="loTypeID" type="xsd:string" use="optional" />
2   <xsd:attribute name="loCatID" type="xsd:string" use="optional" />
3   <xsd:attribute name="qsTypeID" type="xsd:string" use="optional" />
4   <xsd:attribute name="qsCatID" type="xsd:string" use="optional" />
5   <xsd:attribute name="csTypeID" type="xsd:string" use="optional" />
6   <xsd:attribute name="csCatID" type="xsd:string" use="optional" />
7   <xsd:attribute name="coherent3DOFF" type="xsd:boolean" use="optional" />
8   <xsd:attribute name="phldrT" type="xsd:string" use="optional" />
9   <xsd:attribute name="phldr" type="xsd:boolean" use="optional" />
10  <xsd:attribute name="custAng" type="xsd:int" use="optional" />
11  <xsd:attribute name="custFlipVert" type="xsd:boolean" use="optional" />
12  <xsd:attribute name="custFlipHor" type="xsd:boolean" use="optional" />
13  <xsd:attribute name="custSzX" type="xsd:int" use="optional" />
14  <xsd:attribute name="custSzY" type="xsd:int" use="optional" />
15  <xsd:attribute name="custScaleX" type="xsd:int" use="optional" />
16  <xsd:attribute name="custScaleY" type="xsd:int" use="optional" />
17  <xsd:attribute name="custT" type="xsd:boolean" use="optional" />
18  <xsd:attribute name="custLinFactX" type="xsd:int" use="optional" />
19  <xsd:attribute name="custLinFactY" type="xsd:int" use="optional" />
20  <xsd:attribute name="custLinFactNeighborX" type="xsd:int" use="optional" />
21  <xsd:attribute name="custLinFactNeighborY" type="xsd:int" use="optional" />
22  <xsd:attribute name="custRadScaleRad" type="xsd:int" use="optional" />
23  <xsd:attribute name="custRadScaleInc" type="xsd:int" use="optional" />
24  </xsd:complexType>

```

### 5.15.2.1 Presentation Element Properties

The following attributes deal with presentation elements:

- presLayoutVars – The layout variable property set.
- style – The link to the permutation of the style matrix.
- presAssocID – The semantic element associated with this presentation element. This ID is used together with the presName to create a unique key for presentation element indexing.
- presName – The layout node name of this presentation element. This name is used together with presAssocID to create a unique key for presentation element indexing.
- presStyleLbl – The layout node style label of this presentation element..
- presStyleIdx – The layout node style index of this presentation element..
- presStyleCnt – The layout node style count of this presentation element.

### 5.15.2.2 Document Element Properties

The following attributes deal with the document element:

- loTypeID – The ID of the current diagram type.
- loCatID – The ID of the current diagram category.

- 1     • qsTypeID – The ID of the current style type.
- 2     • qaCatID – The ID of the current style category.
- 3     • csTypeID – The ID of the current color transform.
- 4     • csCatID – The ID of the current color transform category.
- 5     • coherent3Doff – Enables or disables coherent 3D behavior for styles that have such behavior defined.

### 6     5.15.2.3      Semantic Element Properties

7     The following attributes relate to the semantic element properties:

- 8     • phldrT – The text used for display in the element if the placeholder flag is set to true. If this field is  
9       not set, then the default placeholder text will be used.
- 10    • phldr – Indicates that the element is a placeholder or sample item.

### 11    5.15.2.4      Customization Properties

12    The following are customization properties or tweaks:

- 13    • custAng – The amount rotation is customized by, in 60,000th of a degree.
- 14    • custFlipVert – Vertical flip.
- 15    • custFlipHor – Horizontal flip.
- 16    • custSzX – Fixed width override for a shape, in emus.
- 17    • custSzY – Fixed height override for a shape, in emus.
- 18    • custScaleX – Amount that the width is scaled by, in 1,000th of a percent.
- 19    • custScaleY – Amount that the height is scaled by, in 1,000th of a percent.
- 20    • custT – If text has been customized then layout will no longer change it.
- 21    • custLinFactX – A percentage of the shape width that is used for offsetting the shape, in 1,000th of a  
22      percent.
- 23    • custLinFactY – A percentage of the shape height that is used for offsetting the shape, in 1,000th of a  
24      percent.
- 25    • custLinFactNeighborX – A percentage of the neighbor's height used for offsetting the shape, in  
26      1,000th of a percent.
- 27    • custLinFactNeighborY – A percentage of the neighbor's height used for offsetting the shape, in  
28      1,000th of a percent.
- 29    • custRadScaleRad – Defines how much the radius has been scaled by, in 1,000th of a percent.
- 30    • custRadScaleInc – Defines how much the include angle has been scaled by, in 1,000th of a percent.

### 31    5.15.3      Data Model

32    The schema dml-diagramDataModel.xsd defines the data model in a diagram. The purpose of the data model  
33    is twofold. The first use of the data model is to hold the information contained in a diagram. For example, in  
34    figure 1 below, the purpose of the data model would be to hold the information, "one", "two" and "three" for  
35    the diagram.



1

2 Figure 11: Example diagram with data.

3 The second use of the data model is to define an initial state of the diagram. This initial state consists of what  
 4 can be thought of as placeholder data, which an application uses to display a diagram initially before any data  
 5 has been entered. Figure 2 shows an example of what a diagram might look like in an initial state containing  
 6 three empty nodes. In this example, the placeholder data consists of three nodes and two connections, which  
 7 will be explained shortly.



8

9 Figure 12: An empty diagram in its initial state.

10 

### 5.15.3.1 Structural Elements

11 

#### 5.15.3.1.1 Element Type

12 There is a single simple type, ST\_PtType, used to define a type of element; this is defined later. Element types  
 13 hold the data associated with a diagram and are defined in relation to one-another through relationship types.  
 14 Seven different types of elements are available to the user:

- 15 • doc – A document element. The document element is the root element within a diagram and can be  
 16 thought of as the canvas which the diagram is drawn on.
- 17 • node – A model element. This is the basic element type which is used and can be used to hold text for  
 18 example.
- 19 • asst – This is used in hierarchy diagrams and represents an assistant element.
- 20 • pres – A presentation element. This element defines the visual aspects associated with a node, or  
 21 rather the presentation aspects of an element.
- 22 • parTrans – A parent transition element. This element holds the data for a parent-child relationship  
 23 between two elements of type node.

- sibTrans – A sibling transition element. This element holds the data for the relationship defined between two elements of type node whom are peers of one another.
- unknown – An element type that is used to maintain backward compatibility.

#### 5.15.3.1.2 Relationship Type

There are defined relationships or connections between two model elements. Four types of relationships are defined in the simple type ST\_CxnType:

- parOf – Parent-child relationship.
- presOf – Presentation relationship.
- presParOf – Presentation parent of relationship.
- unknownRelationship – An unknown relationship type.

#### 5.15.3.1.3 Element

An element is a single item, such as a node or transition in the data model. Within the realm of DrawingML, the complex type CT\_Pt holds information describing an element within a diagram. Within this description lies both the data held within the element, and any formatting on that element. A CT\_Pt is defined as follows:

```

15  <xsd:complexType name="CT_Pt">
16    <xsd:sequence>
17      <xsd:element name="prSet" type="CT_ElemPropSet" minOccurs="0"
18        maxOccurs="1" />
19      <xsd:element name="spPr" type="a:CT_ShapeProperties"
20        minOccurs="0" maxOccurs="1" />
21      <xsd:element name="style" type="a:CT_ShapeStyle"
22        minOccurs="0" maxOccurs="1" />
23      <xsd:element name="t" type="a:CT_TextBody" minOccurs="0"
24        maxOccurs="1" />
25    </xsd:sequence>
26    <xsd:attribute name="modelId" type="ST_ModelId" use="required" />
27    <xsd:attribute name="type" type="ST_PtType" use="optional"
28      default="node" />
29    <xsd:attribute name="cxnId" type="ST_ModelId" use="optional"
30      default="0" />
31  </xsd:complexType>
```

The attribute modelId holds a unique id for a particular element. This unique id can be referenced elsewhere, for example, from within a connection list. This attribute is required for every point defined in the data model.

The last two attributes of the CT\_Pt are optional. The first defines the type of point with the default being a node. The second defines a connection id. This connection id is only used if the point type is of type parTrans, or sibTrans. The connection id refers to a relationship that is defined elsewhere in the data model.

1    5.15.3.1.4         Relationship

2    A relationship is a connection between any two model elements. An example of where a relationship would  
 3    be used can be seen in figures 1 and 2. In each of those examples, the arrows between the nodes have  
 4    relationships defined. A relationship is defined as follows:

```

5      <xsd:complexType name="CT_Cxn">
6          <xsd:attribute name="modelId" type="ST_ModelId" use="required" />
7          <xsd:attribute name="type" type="ST_CxnType" use="optional"
8              default="parOf" />
9          <xsd:attribute name="srcId" type="ST_ModelId" use="required" />
10         <xsd:attribute name="destId" type="ST_ModelId" use="required" />
11         <xsd:attribute name="srcOrd" type="xsd:unsignedInt" use="required" />
12         <xsd:attribute name="destOrd" type="xsd:unsignedInt" use="required" />
13         <xsd:attribute name="parTransId" type="ST_ModelId" use="optional"
14             default="0" />
15         <xsd:attribute name="sibTransId" type="ST_ModelId" use="optional"
16             default="0" />
17         <xsd:attribute name="presId" type="xsd:string" use="optional"
18             default="" />
19     </xsd:complexType>
```

20   The relationship, as with the element, has a unique id associated with it referred to as the modelID. The srcId  
 21   and destId attributes refer to ids of the source element and destination element, respectively, that this  
 22   relationship is defined between.

23   The srcOrd and destOrd refer to the ordinality of siblings for a given connection. For example, if a node had  
 24   three siblings, A, B, and C, then the srcOrd would define if they were to show up as A, B, C, or perhaps B, C,  
 25   and then A.

26   The presId attribute contains the presentation that is associated with this particular relationship.

27   5.15.3.1.5         Element List

28   The complex type CT\_PtList is simply a sequence of elements. Its definition is as follows:

```

29     <xsd:complexType name="CT_PtList">
30         <xsd:sequence>
31             <xsd:element name="pt" type="CT_Pt" minOccurs="0" maxOccurs="unbounded"/>
32         </xsd:sequence>
33     </xsd:complexType>
```

34   5.15.3.1.6         Relationship List

35   This complex type, CT\_CxnList, is simply a sequence of connections. Its definition is as follows:

```

1   <xsd:complexType name="CT_CxnList" oxsd:cname="Relationships">
2     <xsd:sequence>
3       <xsd:element name="cxn" type="CT_Cxn" minOccurs="0"
4         maxOccurs="unbounded" />
5     </xsd:sequence>
6   </xsd:complexType>
```

### 5.15.3.1.7 Data Model

The complex type CT\_DataModel defines the data model and contains a sequence of elements. It is defined as follows:

```

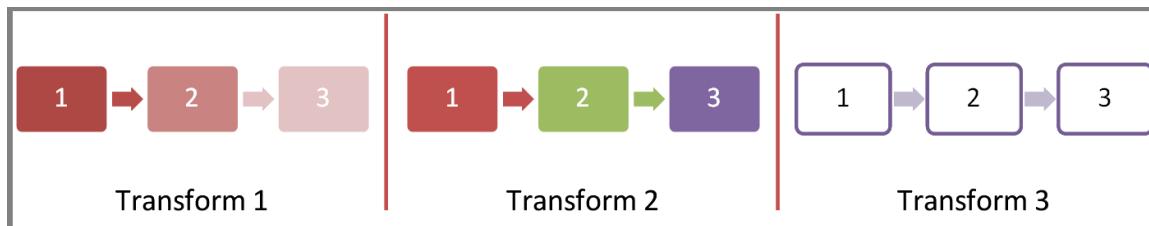
10  <xsd:complexType name="CT_DataModel">
11    <xsd:sequence oxsd:emitArgs="flattenSequence">
12      <xsd:element name="ptLst" type="CT_PtList" />
13      <xsd:element name="cxnLst" type="CT_CxnList" minOccurs="0"
14        maxOccurs="1" />
15      <xsd:element name="bg" type="a:CT_BackgroundFormatting"
16        minOccurs="0" />
17      <xsd:element name="whole" type="a:CT_WholeE2oFormatting"
18        minOccurs="0" />
19    </xsd:sequence>
20  </xsd:complexType>
```

The data model contains a list of elements, a list of connections, and formatting properties for the background object and the diagram container. This complex type is responsible for holding all data-bound information of the diagram being created.

### 5.15.4 Color Transforms

Color transforms define how colors are applied to diagrams. Color transforms define how color is used in the diagram as a whole, and they mandate things such as which theme color or colors will be used, if there is a tint or shade applied to a certain color or part of the diagram, or if color is even used at all. Some examples of what color transforms can do to a simple diagram can be seen in figure 3.

29



31 Figure 13: Different examples of a color transform applied to a diagram

1    5.15.4.1       Structural Elements

2    The structural elements which come together to create a color transform, or rather, the complex type  
 3    CT\_ColorTransform, are as follows:

- 4       • CT\_CTName
- 5       • CT\_CTDscription
- 6       • CT\_CTCategories
- 7       • CT\_CTCategories
- 8       • ST\_ClrAppMethod
- 9       • ST\_HueDir
- 10      • CT\_CColors
- 11      • CT\_CTSStyleLabel
- 12      • CT\_CTVerson
- 13      • CT\_CColorTransformHeader
- 14      • CT\_CColorTransformHeaderLst

15     The complex types CT\_CTName (name), CT\_CTDscription (description), CT\_CTCategories (category), and  
 16     CT\_CTCategories (list of categories) work together to name, describe and categorize the particular color  
 17     transform. These types are mirrored elsewhere throughout DrawingML in the different subjects in order to  
 18     perform the same tasks of naming, describing, and categorizing.

19     The name consists simply of two strings, one of a name for the color transform, which is required, and an  
 20     optional language tag. The language allows someone to specify a language for a given title. It is possible to  
 21     specify multiple titles that are language dependant. The description also has the optional language attribute  
 22     as in the name, along with a second required string attribute which holds the actual description. The usage of  
 23     this is exactly the same as within CT\_CTName. CT\_CTName and CT\_CTDscription are defined in the  
 24     following way:

```
25       <xsd:complexType name="CT_CTName">
26         <xsd:attribute name="lang" type="xsd:string" use="optional" />
27         <xsd:attribute name="val" type="xsd:string" use="required" />
28       </xsd:complexType>
29       <xsd:complexType name="CT_CTDscription">
30         <xsd:attribute name="lang" type="xsd:string" use="optional" />
31         <xsd:attribute name="val" type="xsd:string" use="required" />
32       </xsd:complexType>
```

33     The category and categories complex types , CT\_CTCategories and CT\_CTCategories, respectively, define how  
 34     the color transform is categorized within the user interface of the application. The category contains a name,  
 35     or type, along with a priority that defines the ordering of the color transform. The lower the priority, the  
 36     earlier in the category it will display. If there is a tie, the unique id associated with the color transform will  
 37     decide the order alphabetically. CT\_CTCategories is simply a list of CT\_CTCategories. The two complex types  
 38     are defined as follows:

```

1   <xsd:complexType name="CT_CTCategories">
2     <xsd:attribute name="type" type="xsd:anyURI" use="required" />
3     <xsd:attribute name="pri" type="xsd:unsignedInt" use="required"/>
4   </xsd:complexType>
5   <xsd:complexType name="T_CTCategories">
6     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
7       <xsd:element name="cat" type="CT_CTCategories" minOccurs="0"
8         maxOccurs="unbounded" />
9     </xsd:sequence>
10    </xsd:complexType>

```

#### 5.15.4.1.1 Color Application Method

The simple type ST\_ClrAppMethod lists the different options for color to be applied to a diagram. There are three options available to the user: span, cycle, and repeat. Given a list of colors, which go from color A to color B, the differences in these three options can be defined. These options are shown below in Figure 4.

The span option will, from the start of the diagram to the end of the diagram, interpolate between the colors A through B for every node along the way.

The cycle option will interpolate from A to B then back to A from the start of the diagram to the end of the diagram.

The repeat option applies colors A through B one at a time for each point in the diagram, then repeats A through B as needed.

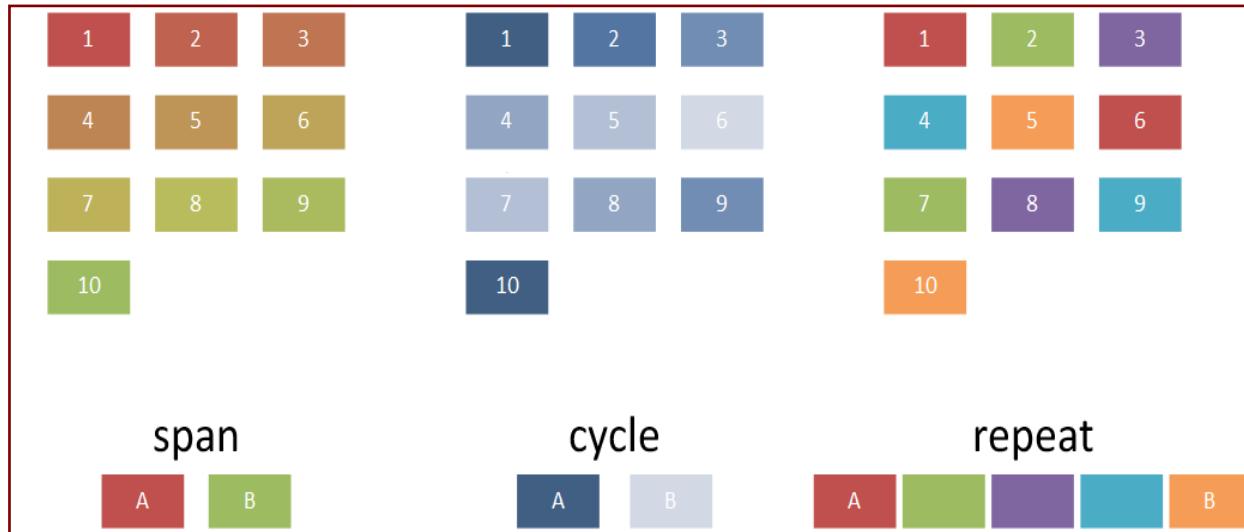
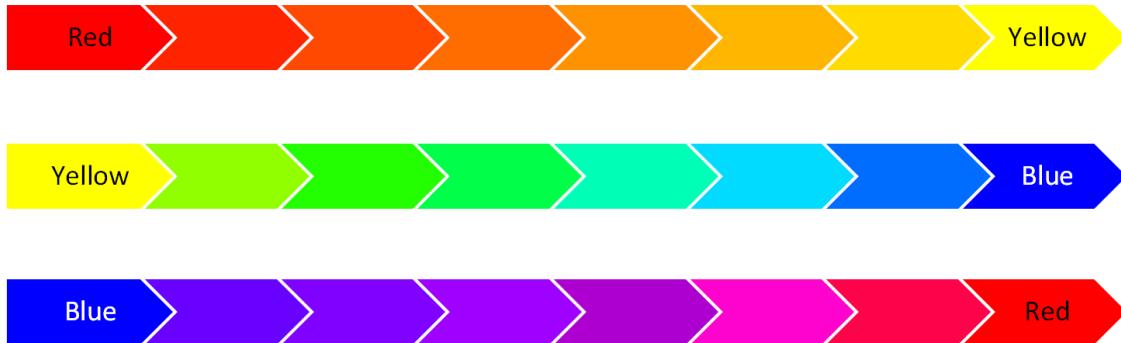


Figure 14: Examples of the three different ways a color transform is applied to a diagram.

#### 5.15.4.1.2 Hue Direction

The simple type ST\_HueDir defines the direction of a hue color shift around a color wheel. A user can either define the shift to occur in the clockwise (cw) direction, or in the counterclockwise (ccw) direction. For

example, in Figure 4, the span colors are red and green. The behavior shown in figure 4 is a shift in the cw direction. If the hue shift had been defined in the ccw direction, the colors interpolated between colors A and B would have been in the hues purple and blue. Another example of a hue direction shift in the clockwise can be seen in Figure 5 below along with the color shift from red to yellow and then from yellow to blue along the primary colors. Counterclockwise shifts would occur in the direction of yellow to red and blue to yellow in the examples below.



## Hue Direction = Clockwise

7

8 Figure 15: Example hue shifts in the clockwise direction around a color wheel applied to two diagrams. The  
9 three primary colors, red, yellow, and blue are used to represent the three major sections of a color wheel  
10 which ranges between red to yellow to blue to red.

11 5.15.4.1.3 Colors

12 The complex type CT\_Colors holds the actual color values that are to be applied to a given diagram and how  
13 those colors are to be applied. It contains the color application method and hue shift direction, and is defined  
14 as follows:

```
15 <xsd:complexType name="CT_Colors">
16   <xsd:sequence>
17     <xsd:group ref="a:EG_ColorChoice" minOccurs="0" maxOccurs="unbounded" />
18   </xsd:sequence>
19   <xsd:attribute name="meth" type="ST_ClrAppMethod" use="optional"
20     default="span" />
21   <xsd:attribute name="hueDir" type="ST_HueDir" use="optional"
22     default="cw" />
23 </xsd:complexType>
```

24 The sequence of colors is defined via the sequence of EG\_ColorChoices.

1    5.15.4.1.4        Style Label

2    The complex type CT\_CStyleLabel packages together colors for the different pieces of a diagram. There are  
 3    six different aspects to a diagram that can be colored independently of one another. Each of the six parts is of  
 4    type CT\_Colors. They are:

- 5        • Fill Colors – The colors that actually fill the shapes in the diagram
- 6        • Line Colors – The colors of the lines on the shapes in the diagram.
- 7        • Effect Colors – The colors of the effects applied to the shapes within the diagram (eg. Glow).
- 8        • Text Line Colors – The colors of the lines on the text within the diagram.
- 9        • Text Fill Colors – The color of the text within the diagram.
- 10      • Text Effect Colors – The colors of the effects applied to the text within the diagram.

11     The final piece of a style label is simply its name, which is a string. CT\_CStyleLabel is defined as follows:

```

12   <xsd:complexType name="CT_CStyleLabel">
13     <xsd:sequence>
14       <xsd:element name="fillClrLst" type="CT_Colors"
15         minOccurs="0" maxOccurs="1" />
16       <xsd:element name="linClrLst" type="CT_Colors"
17         minOccurs="0" maxOccurs="1" />
18       <xsd:element name="effectClrLst" type="CT_Colors"
19         minOccurs="0" maxOccurs="1" />
20       <xsd:element name="txLinClrLst" type="CT_Colors"
21         minOccurs="0" maxOccurs="1" />
22       <xsd:element name="txFillClrLst" type="CT_Colors"
23         minOccurs="0" maxOccurs="1" />
24       <xsd:element name="txEffectClrLst" type="CT_Colors"
25         minOccurs="0" maxOccurs="1" />
26     </xsd:sequence>
27     <xsd:attribute name="name" type="xsd:string" use="required" />
28   </xsd:complexType>
```

29    5.15.4.1.5        Version

30     The simple type ST\_CTVVersion defines the minimum version of an application that the color transform will  
 31    work with. The version corresponds to build numbers in the major.minor.build.revision format and is defined  
 32    as follows:

33     [0-9]?[0-9])?(.[0-9]?[0-9])?(.[0-9]{4})?(.[0-9]{4}

34    5.15.4.1.6        Color Transform

35     The complex type CT\_ColorTransform brings together all of the pieces into one cohesive structure. This is the  
 36    actual definition of a color transform, which can be applied to any diagram; it is defined as follows:

```

1   <xsd:complexType name="CT_ColorTransform">
2     <xsd:sequence>
3       <xsd:element name="title" type="CT_CTName" minOccurs="0"
4         maxOccurs="unbounded" />
5       <xsd:element name="desc" type="CT_CTDscription"
6         minOccurs="0" maxOccurs="unbounded" />
7       <xsd:element name="catLst" type="CT_CTCategories"
8         minOccurs="0" />
9       <xsd:element name="styleLbl" type="CT_CTSyleLabel"
10        minOccurs="0" maxOccurs="unbounded" odoc />
11     </xsd:sequence>
12     <xsd:attribute name="uniqueId" type="xsd:anyURI" use="optional"/>
13     <xsd:attribute name="minVer" type="ST_CTVerson" use="optional"
14       default="12.0" />
15   </xsd:complexType>

```

16 A color transform contains a title, description, category list, and style label in a sequence along with a unique  
17 id and a minimum version.

#### 18 5.15.4.2 Color Transform Header

19 Two complex types, CT\_ColorTransformHeader and CT\_ColortransformHeaderLst, help alleviate potential  
20 performance concerns with initially loading in a large number of color transforms. The header information  
21 contains the minimum information required to load a color transform into the application. Because of this,  
22 color transforms themselves can be loaded only when needed, and other initialization work can progress  
23 quickly without loading unneeded information.

24 CT\_ColorTransformHeader contains information about the title of the color transform, the description, how it  
25 is categorized, the unique id, minimum version, and resId. It is defined in the following way:

```

26   <xsd:complexType name="CT_ColorTransformHeader">
27     <xsd:sequence>
28       <xsd:element name="title" type="CT_CTName" minOccurs="1"
29         maxOccurs="unbounded" />
30       <xsd:element name="desc" type="CT_CTDscription"
31         minOccurs="1" maxOccurs="unbounded" />
32       <xsd:element name="catLst" type="T_CTCategories"
33         minOccurs="0" o:cname="categories" />
34     </xsd:sequence>
35     <xsd:attribute name="uniqueId" type="xsd:anyURI" use="required"/>
36     <xsd:attribute name="minVer" type="ST_CTVerson" use="optional"
37       default="12.0" />
38     <xsd:attribute name="resId" type="xsd:int" use="optional"
39       default="0" />
40   </xsd:complexType>

```

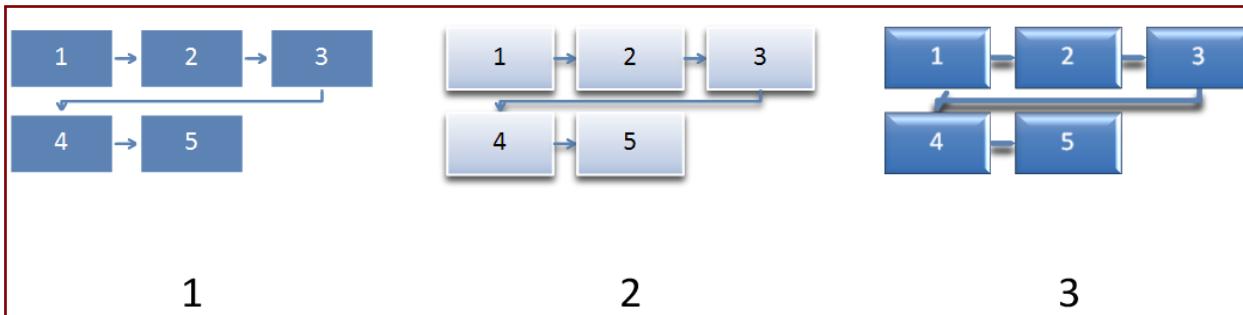
1 The complex type CT\_ColorTransformHeaderLst simply contains a list of color transform headers. It is  
 2 defined as follows:

```
3 <xsd:complexType name="CT_ColorTransformHeaderLst">
4   <xsd:sequence>
5     <xsd:element name="colorsDefHdr" type="CT_ColorTransformHeader"
6       minOccurs="0" maxOccurs="unbounded" />
7   </xsd:sequence>
8 </xsd:complexType>
```

## 9 5.15.5 Style Definition

10 A style definition is similar to a color transform. A style definition defines items such as the font used, the  
 11 thickness of a contour line, 3-D properties on the diagram, among other things. Style definitions work in  
 12 combination with color transforms to give an overall look and feel for the diagram. Some examples of style  
 13 definitions in use are in figure 5.

14



15 16 Figure 16: Examples of using three different style definitions on a diagram.

### 17 5.15.5.1 Structural Elements

18 The structural elements which come together to create a style definition are as follows:

- 19 • CT\_SDName
- 20 • CT\_SDDescription
- 21 • CT\_SDCategory
- 22 • CT\_SDCategories
- 23 • CT\_TextProps
- 24 • CT\_StyleLabel
- 25 • ST\_SDVersion
- 26 • CT\_StyleDefinition
- 27 • CT\_StyleDefinitionHeader
- 28 • CT\_StyleDefinitionHeaderLst

1 CT\_SDName, CT\_SDDescription, CT\_SDCategory, CT\_SDCategories, and ST\_SDVersion are all defined  
 2 exactly as they are within color transforms. These types were recreated within the style definition area to  
 3 allow slight differentiations to be made, although, at this time they are defined exactly the same.

4 The text properties, style label, and then the style definition combine together to create a style definition,  
 5 which is applied to a diagram.

#### 6 5.15.5.1.1 Text Properties

7 The complex type CT\_TextProps holds any 3-D associated properties of the text that is to be held in the  
 8 diagram. A CT\_TextProps is defined as follows:

```
9 <xsd:complexType name="CT_TextProps">
10   <xsd:sequence>
11     <xsd:group ref="a:EG_Text3D" minOccurs="0" maxOccurs="1" />
12   </xsd:sequence>
13 </xsd:complexType>
```

14 All that is contained within the text properties is an EG\_Text3D complex type. The usage of the text properties  
 15 complex type allows 3-D text properties to be defined for a diagram style.

#### 16 5.15.5.1.2 Style Label

17 The style label contains information pertaining to the styleable elements within a diagram. These elements  
 18 include the shape properties and text properties along with any references to the style matrix or document  
 19 theme. The shape properties are defined in two different ways: the scene3d element that pertains to the  
 20 scene on a whole (and includes lighting effects, rotations, and the like), and any 3-D and material settings are  
 21 held in the sp3d element. CT\_StyleLabel is defined as follows:

```
22 <xsd:complexType name="CT_StyleLabel">
23   <xsd:sequence>
24     <xsd:element name="scene3d" type="a:CT_Scene3D"
25       minOccurs="0" maxOccurs="1" />
26     <xsd:element name="sp3d" type="a:CT_Shape3D" minOccurs="0"
27       maxOccurs="1" />
28     <xsd:element name="txPr" type="CT_TextProps" minOccurs="0"
29       maxOccurs="1" />
30     <xsd:element name="style" type="a:CT_ShapeStyle"
31       minOccurs="0" maxOccurs="1" />
32   </xsd:sequence>
33   <xsd:attribute name="name" type="xsd:string" use="required" />
34 </xsd:complexType>
```

35 As with many other complex types, the style label has an attribute reserved for a name. This is simply a string  
 36 that names the particular style label. This style label can then be referenced from within a diagram definition,  
 37 as we shall see below.

1 Note that the scene3d element contained within a style label acts on the level of an individual shape, rather  
 2 than the diagram as a whole. A second scene3d element is defined within the style definition; that acts on the  
 3 diagram level and allows for scene coherent 3-D to be applied to the diagram. A style definition contains a  
 4 style label.

5 5.15.5.1.3 Style Definition

6 The style definition complex type, CT\_StyleDefinition, is the root element used to define a style definition. As  
 7 with the root element of a color transform, within the style definition there exists a title, description, category,  
 8 unique id, and minimum version number. These elements serve the same purpose as they do within the color  
 9 transform.

10 The interesting aspects of a style definition complex type are that it holds a style label, another style index  
 11 (which is contained within the style label as well), and another scene3d (which is also contained within the  
 12 style label as has been previously mentioned). The scene3d element applies to the diagram on a whole and  
 13 allows for scene coherent 3-D to be applied to the diagram. The duplication of the style index is two-fold. If a  
 14 style index is not defined within the style label, then the default style index, or rather, the index defined in this  
 15 complex type is used. Since a diagram definition can reference a style label, and not a style definition, the  
 16 style index is also required within the style label. A CT\_StyleDefinition is defined as follows:

```

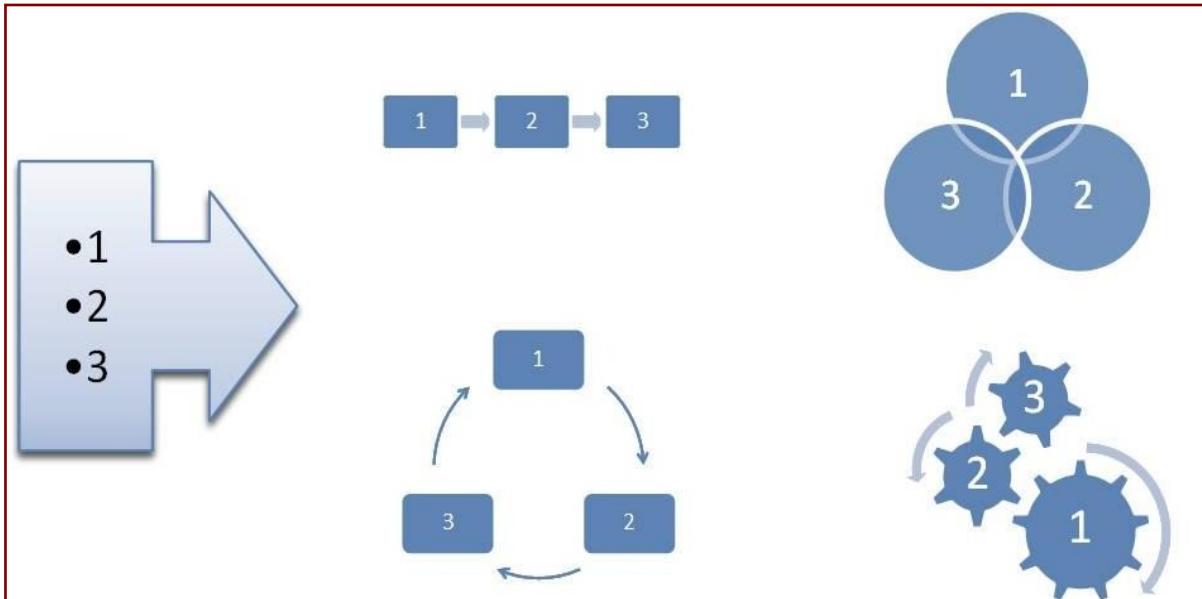
17 <xsd:complexType name="CT_StyleDefinition">
18   <xsd:sequence>
19     <xsd:element name="title" type="CT_SDName" minOccurs="0"
20       maxOccurs="unbounded" />
21     <xsd:element name="desc" type="CT_SDDescription"
22       minOccurs="0" maxOccurs="unbounded" />
23     <xsd:element name="catLst" type="CT_SDCategories"
24       minOccurs="0" />
25     <xsd:element name="scene3d" type="a:CT_Scene3D"
26       minOccurs="0" maxOccurs="1" />
27     <xsd:element name="style" type="a:CT_ShapeStyle"
28       minOccurs="0" maxOccurs="1" />
29     <xsd:element name="styleLbl" type="CT_StyleLabel"
30       minOccurs="1" maxOccurs="unbounded" />
31   </xsd:sequence>
32   <xsd:attribute name="uniqueId" type="xsd:anyURI" use="optional"/>
33   <xsd:attribute name="minVer" type="ST_SDVersion" use="optional" ^
34     default="12.0" />
35 </xsd:complexType>
```

36 5.15.5.1.4 Style Definition Header

37 The complex types CT\_StyleDefinitionHeader and CT\_StyleDefinitionHeaderLst perform the same function  
 38 as the two complex header types in color transforms. They are used to pre-load required information so the  
 39 actual loading of the style definition can happen only when needed.

## 1 5.15.6 Layout

2 The single largest aspect of DrawingML is Layout. Ultimately, layout is responsible for defining all aspects of  
 3 the diagram outside of color and style. It defines how the diagram looks, how it behaves, and how the data is  
 4 to be mapped. Figure 6 shows different examples of how layout works to create a diagram holding the  
 5 data '1', '2', and '3'.



6  
 7 Figure 17: Different layouts mapping the data 1, 2, and 3 to four different diagrams

8 There are two aspects to layout: defining the numerous complex types utilized by diagram definitions, and the  
 9 diagram definitions themselves. Diagram definitions are a fundamental part of layout, and utilize everything in  
 10 order to define new diagrams.

### 11 5.15.6.1 Basic Layout Types

12 There are a very large number of simple types defined in this subject. These simple types are all associated  
 13 with properties of a diagram that can be modified to create a desired behavior. These simple types, along with  
 14 an explanation of what each does, follow.

#### 15 5.15.6.1.1 Algorithm Type

16 ST\_AlgorithmType is responsible for which algorithm will be used to layout the diagram. The algorithm layout  
 17 chosen determines if the diagram behaves as if it were a simple list, a circular cycle, or some other type of  
 18 diagram. The algorithms available are:

- 19 • unknown – Unknown algorithm type. This type is used for extensibility reasons. Use of this algorithm  
 20 type can be used by future versions/implementations which define new algorithm types other than the  
 21 above mentioned algorithm types. By using this type, the current implementations know that they will  
 22 not be able to correctly render the diagram if they have only implemented the above mentioned  
 23 algorithms for layout.

- composite – The composite algorithm specifies the size and position for all child layout nodes. You can use it to create graphics with a predetermined layout or in combination with other algorithms to create more complex shapes.
- conn – the connector algorithm lays out and routes connecting lines, arrows, and shapes between layout nodes
- cycle – the cycle algorithm lays out child layout nodes around a circle or portion of a circle using equal angle spacing
- hierChild – the hierarchy child algorithm works with the hierRoot algorithm to create hierarchical tree layouts. This algorithm aligns and positions its child layout nodes in a linear path under the hierRoot layout node.
- hierRoot – the hierarchy root algorithm works with the hierChild algorithm to create hierarchical tree layouts. The hierRoot algorithm aligns and positions the hierRoot layout node in relation to the hierChild layout nodes.
- pyra – the pyramid algorithm lays out child layout nodes along a vertical path and works with the trapezoid shape to create a pyramid
- lin – the linear algorithm lays out child layout nodes along a linear path
- sp – the space algorithm is used to specify a minimum space between other layout nodes or as an indication to do nothing with the layout node's size and position
- tx – manges layout of text within a shape
- snake – the snake algorithm lays out child layout nodes along a linear path in two dimensions, allowing the linear flow to continue across multiple rows or columns

#### 5.15.6.1.2 Axis Type

The simple type ST\_AxisType defines how layout maps data to the diagram for a given point in the diagram. The different ways this can be mapped is as follows:

- self – the layout maps to the current data point
- ch – the layout node can map to the children of the current data point, but not to descendants lower in the hierarchy
- des – the layout node can map to a descendant of the current data point
- desOrSelf – the layout node can map the current data point, or to a descendant of the current data point
- par – the layout node maps to the parent data point
- ancst – the layout node can map to the ancestors of the current data point (parents, grandparents, great grandparents, etc)
- ancstOrSelf – the layout node can map an ancestor data point or the current data point
- followSib – the layout node can map to a following sibling peer to the current data point
- precedSib – the layout node can map to a preceding sibling peer to the current data point.
- root – the layout node can map to the root
- none – the layout node doesn't map to any data point

1    5.15.6.1.3              Boolean Operators

2    Boolean type operators, defined by ST\_BooleanOperator, are for layout.

- 3         • none – no operator defined  
 4         • equ – ‘equal to’ operator, returns true if the two compared values are equal, false otherwise  
 5         • gte – ‘greater than or equal to’ operator  
 6         • lte – ‘less than or equal to’ operator

7    5.15.6.1.4              Child Order Type

8    The simple type ST\_ChildOrderType specifies the child order type for a layout node.

- 9         • b – bottom  
 10        • t – top

11   5.15.6.1.5              Constraint Types

12   The simple type ST\_ConstraintTypes defines the constraints that can be used as limits or modifications to the diagram or the nodes held within the diagram. These constraints manage the behavior of many properties that can be defined within the diagram. The different constraints that can be applied to a diagram are as follows:

- 16         • unknown – An unknown constraint. This can be used by implementing applications to define a constraint type outside of the scope of what is defined by this simple type.  
 17         • alignOff – specifies the alignment offset  
 18         • begMarg – specifies the beginning margin  
 19         • bendDist – specifies the distance at which a connector will bend  
 20         • begPad – specifies the distance between the edge of the transition node and the connector shape  
 21         • b – bottom alignment  
 22         • bMarg – the bottom margin  
 23         • bOff – specifies the amount of offset relative to the bottom of the node  
 24         • ctrX – specifies the center of the node in the X-direction  
 25         • ctrXOff – specifies the amount of offset of the center of the node in the X-direction  
 26         • ctrY – specifies the center of the node in the Y-direction  
 27         • ctrYOff – specifies the amount of offset of the center of the node in the Y-direction  
 28         • connDist – specifies the distance between connectors. Intended for use with boolean and reference constraints. Overrides values specified in the layout definition part  
 29         • diam – specifies the diameter, and is used within the cycle algorithm type  
 30         • endMarg – specifies the end margins  
 31         • endPad – specifies the distance between the edge of the transition node and the connector shape  
 32         • h – specifies the height  
 33         • hArH – specifies the arrowhead height  
 34         • hOff – specifies the height offset

- 1     • l – specifies the left
- 2     • lMarg – specifies the amount of left margin
- 3     • lOff – specifies the amount of left offset
- 4     • r – specifies right
- 5     • rMarg – specifies the amount of right margin
- 6     • rOff – specifies the amount of right offset
- 7     • primFontSz – specifies the primary font size
- 8     • pyraAcctRatio - specifies the fraction of the width of the diagram that is reserved for the fly-outs at their shortest distance
- 9     • secFontSiz – specifies the secondary font size
- 10    • stemThick – specifies the thickness of the shaft on an arrow
- 11    • t – specifies the top
- 12    • tMarg – specifies the amount of top margin
- 13    • tOff – specifies the amount of top offset
- 14    • userA through userZ – User defined information. This set of enumerations allow a user to define specific values which can be referenced at a later time within a diagram definition. For example, if the value 5.1 was important to the layout and definition of a diagram, the user could define userA to be equal to 5.1 and then use the userA constraint within the diagram definition. This would allow a single place to update the value across the entire diagram definition.
- 15    • w – specifies the width
- 16    • wArH – specifies the width of an arrowhead
- 17    • wOff – specifies the amount of width offset

### 23    5.15.6.1.6       Constraint Relationships

24    The simple type ST\_ConstraintRelationship defines the application of retrieval data the constraint is applied  
25    to. The following relationships are available:

- 26    • self – the constraint is applicable to the current point
- 27    • ch – the constraint is applicable to a child of the current point
- 28    • des – the constraint is applicable to a descendant of the current point

### 29    5.15.6.1.7       Element Type

30    The simple type ST\_ElementType defines the type of element, or point which get created and how they are  
31    created from the data at hand. The different ways to pull from the data to create points are as follows:

- 32    • all – use all of the data points, nodes and transitions
- 33    • doc – use the document level, or root data point
- 34    • node – use only data nodes input by the user
- 35    • norm – in place for extensibility and behaves exactly opposite of the asst element type
- 36    • nonNorm – in place for extensibility and behaves exactly opposite of the nonAsst element type
- 37    • asst – use assistant data nodes within hierarchy algorithm

- 1     • nonAsst – use non-assistant nodes within the hierarchy algorithm
- 2     • parTrans – Use only parent transitions between nodes. Parent transitions are similar to sibling
- 3        transitions, except that they represent parent/child relationships. Parent transitions are most
- 4        commonly used in hierarchy diagrams, such as organization charts, to draw lines between parent and
- 5        child nodes.
- 6     • pres – specifies that the node is related to the presentation level
- 7     • sibTrans – Use only sibling transitions between data nodes. These transitions represent sibling
- 8        relationships between nodes, and are frequently mapped to arrows between shapes in the drawing. A
- 9        sibTrans value is sometimes used to create white space between nodes.

#### 10    5.15.6.1.8           Parameter ID

11    The simple type ST\_ParameterId defines parameters that can be used to modify the behavior or algorithms.

12    The modifications are as follows:

- 13     • horzAlign – specifies the horizontal alignment
- 14     • vertAlign – specifies the vertical alignment
- 15     • chDir – specifies the child direction
- 16     • chAlign – specifies the alignment of the children
- 17     • secChAlign – specifies a secondary child alignment
- 18     • linDir – specifies whether children are arranged from left to right, right to left, top to bottom, or
- 19        bottom to top
- 20     • secLinDir – specifies a secondary linear direction in which children are
- 21        arranged from left to right, right to left, top to bottom, or bottom to top
- 22     • stElem – specifies the point type of the layout node to use as the first shape in the cycle
- 23     • bendPt – specifies where the bend point is to be located on connectors
- 24     • connRout – specifies whether the connector is drawn as a single straight line, orthogonal lines with a
- 25        single bend, or a curve that uses the diam constraint
- 26     • begSty – specifies whether the beginning of the connector has an arrowhead
- 27     • endSty – specifies whether the end of the connector has an arrowhead
- 28     • dim – specifies the connector dimension, 2-D, 3-D, or custom
- 29     • rotPath - if rotPath=alongPath, the algorithm rotates all children perpendicular to the line from the
- 30        cycle's center to the child node; otherwise they are not rotated. The alongPath value does not take
- 31        rotation into account when determining if shapes overlap
- 32     • ctrShpMap - None specifies to place nodes around a circle. First node (fNode) specifies to place the
- 33        first node in the center and the remaining nodes around the circle
- 34     • nodeHorzAlign – specifies the horizontal node alignment
- 35     • nodeVertAlign – specifies the vertical node alignment
- 36     • fallback – specifies if the fallback occurs in a single dimension (e.g. vertically) or if it occurs in two
- 37        dimensions
- 38     • txDir – specifies where the text of the first node starts
- 39     • pyraAcctPos – specifies the placement of the fly-out grandchildren

- 1     • pyraAcctTxMar – specifies the placement of one edge of the child text
- 2     • txBIDir – Specifies the text block direction, vertical or horizontal
- 3     • txAnchorHorz – Specifies the horizontal text anchor position.
- 4     • txAnchorVert – Specifies the vertical text anchor position.
- 5     • txAnchorHorzCh – Specifies the horizontal text anchor position for child text.
- 6     • txAnchorVertCh – Specifies the vertical text anchor position for child text.
- 7     • parTxLTRAlign – Specifies the paragraph alignment of parent text when the shape has only parent
- 8       text. This parameter applies when the text direction is left to right.
- 9     • partTxRTLAlign – Specifies the paragraph alignment of parent text when the shape has only parent
- 10      text. This parameter applies when the text direction is right to left.
- 11     • shpTxLTRAlignCh – Specifies the paragraph alignment of all text within the shape when the shape
- 12       contains both parent and child text. This parameter applies when the text direction is left to right.
- 13     • shpTxRTLAlignCh – Specifies the paragraph alignment of all text within the shape when the shape
- 14       contains both parent and child text. This parameter applies when the text direction is right to left.
- 15     • autoTxRot – specifies how text is oriented relative to the shape
- 16     • grDir – Specifies the direction of growth for the snake algorithm.
- 17     • flowDir – specifies whether nodes are arranged in rows or columns for the snake algorithm.
- 18     • contDir – Specifies the direction of subsequent rows or columns in the snake algorithm.
- 19     • bkpt – specifies the point at which the diagram starts to snake
- 20     • off – specifies whether each row and column is centered or offset from the previous row or column
- 21     • hierAlign – specifies the alignment of the hierarchy
- 22     • bkPtFixedVal – specifies where the sname should break if bkpt is set to fixed
- 23     • stBulletLvl – Specifies the level at which to start using bullets for incoming text.
- 24     • stAng – Specifies the angle at which the first shape is placed. Angles are in degrees, measured
- 25       clockwise from a line pointing straight upward from the center of the cycle.
- 26     • spanAng – Specifies the angle the cycle spans. Final shapealign text is placed at stAng+spanAng,
- 27       unless spanAng=360. In that case, the algorithm places the text so that shapes do not overlap
- 28     • ar – Specifies the aspect ratio (width to height) of the composite node to use when determining child
- 29       constraints. A value of 0 means leave the width and height constraints as is. The algorithm may
- 30       temporarily shrink one dimension to achieve that ratio
- 31     • lnSpPar – specifies the line spacing of the parent
- 32     • lnSpAfParP – specifies the line spacing after the parent paragraph
- 33     • LnSpCh specifies the line spacing of a child
- 34     • lnSpAfChP – specifies the line spacing after the child paragraph
- 35     • rtShortDist – Specifies the routing to use the shortest distance for connectors.
- 36     • alignTx – specifies if to hold text or not
- 37     • pyraLvlNode – If pyramid has a composite child node, specifies the name of the node that is a child of
- 38       the composite that makes up the pyramid itself. If the node specifies a trapezoid shape, it modifies the
- 39       adjustment handles to construct a pyramid.

- 1     • pyraAcctBkgdNode – If pyramid has a composite child node, specifies the child node that should hold  
2       the child text.
- 3     • pyraAcctTxNode – If pyramid has a composite child node, specifies the name of the node that is a  
4       child of the composite that makes up the child flyout shape.
- 5     • srcNode – Specifies the name of the layout node from which to start the connection.
- 6     • dstNode – Specifies the name of the layout node from which to end the connection from.
- 7     • begPts – Specifies the point type for the beginning of a connector.
- 8     • endPts – Specifies the point type for the end of a connector.
- 9     • unknown – An unkown parameter id. This can be used by implementing applications to define a  
10       parameter id outside of the scope of what is defined by this simple type.

#### 11    5.15.6.1.9           Function Type

12    The simple type ST\_FunctionType defines different types of conditional expressions that can be utilized. The  
13       different types of expressions are:

- 14     • cnt – Specifies the count of items.
- 15     • pos - Retrieves the position of the node in the specified set of nodes.
- 16     • revPos - Reverse position function.
- 17     • posEven - Returns 1 if the specified node is at an even numbered position in the data model.
- 18     • posOdd - Returns 1 if the specified node is in an odd position in the data model.
- 19     • var - Used to reference a variable.
- 20     • depth - Specifies the depth of items.
- 21     • maxDepth - Defines the maximum depth of items.

#### 22    5.15.6.1.10          Function Operator

23    The simple type ST\_FunctionOperator defines the different condition expression operators that can be used.  
24    The different operators are as follows:

- 25     • equ – equal
- 26     • neq – not equal
- 27     • gt – greater than
- 28     • lt – less than
- 29     • gte – greater than or equal to
- 30     • lte – less than or equal to

#### 31    5.15.6.1.11          Horizontal Alignment

32    The simple type, ST\_HorizontalAlignment, specifies the different options available for alignment horizontally.  
33    The options are:

- 34     • l – left
- 35     • ctr – center
- 36     • r – right

- 1       • none – none

2    5.15.6.1.12      Vertical Alignment

3    The simple type, ST\_VerticalAlignment, specifies the different options available for alignment vertically. The  
4    options are:

- 5       • t – top  
6       • mid – middle  
7       • b – bottom  
8       • none – none  
9       • Child Direction

10   The simple type ST\_ChildDirection is used to specify the direction the children are laid out. The different  
11   options are:

- 12       • horz – horizontally  
13       • vert – vertically

14   5.15.6.1.13      Child Alignment

15   The simple type ST\_ChildAlignment defines the alignment parameter types for children. The different types  
16   are:

- 17       • t – top  
18       • b – bottom  
19       • l – left  
20       • r – right

21   5.15.6.1.14      Secondary Child Alignment

22   The simple type ST\_SecondaryChildAlignment defines secondary alignment parameter types for children.  
23   The simple type ST\_ChildAlignment is mirrored here with the addition of the none type.

24   5.15.6.1.15      Linear Direction

25   The simple type ST\_LinearDirection defines the linear direction parameter types. The types are as follows:

- 26       • fromL – from left  
27       • fromR – from right  
28       • fromT – from top  
29       • fromB – from bottom

30   5.15.6.1.16      Secondary Linear Direction

31   The simple type ST\_SecondaryLinearDirection defines a secondary linear direction parameter. This simple  
32   type mirrors exactly the simple type ST\_LinearDirection with the addition of the none type.

1    5.15.6.1.17      Starting Element

2    The simple type ST\_StartElement specifies the first node point type for a cycle diagram. The different  
3    starting elements are:

- 4        • node – node  
5        • trans – transition

6    5.15.6.1.18      Rotation Path

7    The simple type ST\_RotationPath specifies the way in which the algorithm rotates children. The different  
8    rotation types are:

- 9        • none – no rotation is performed  
10       • alongPath – the children are rotated perpendicular to the line from the cycle's center to the child  
11                  node

12   5.15.6.1.19      Center Shape Mapping

13   The simple type ST\_CenterShapeMapping specifies how the first node of a cycle diagram is laid out within the  
14   diagram. The different places to put the first node are:

- 15       • none – the node is laid out around the circle  
16       • fNode – the node is placed in the center of the circle and the remaining nodes along the outside of the  
17                  circle

18   5.15.6.1.20      Bend Point

19   The simple type ST\_BendPoint specifies where the bend point is to be located along elbow connectors. The  
20   different options are:

- 21       • beg – beginning  
22       • def – default  
23       • end – end

24   5.15.6.1.21      Connector Routing

25   The simple type ST\_ConnectorRouting defines how the routing of a connector happens within the diagram.  
26   The different routing options are:

- 27       • stra – straight  
28       • bend – an elbow connection  
29       • curve – a curved connection  
30       • longCurve – a curved connection with a larger radius than simple curve

31   5.15.6.1.22      Arrowhead Style

32   The simple type ST\_ArrowheadStyle defines the style of the arrowhead used on a connector. The different  
33   options are:

- 1     • auto – automatic
- 2     • arr – an arrowhead is used
- 3     • noArr – no arrowhead is used

4     **5.15.6.1.23       Connector Dimension**

5     The simple type ST\_ConnectorDimension defines the dimension of a connector used in a diagram. The  
6     different dimension types are:

- 7     • 1D – a single dimension connector, for example, a line
- 8     • 2D – a two dimensional connector, for example, an arrow
- 9     • cust – custom

10    **5.15.6.1.24       Connector Point**

11    The simple type ST\_ConnectorPoint defines the point at which the connector starts and ends. The different  
12    beginning and ending types are:

- 13    • auto – automatic
- 14    • bCtr – bottom center
- 15    • ctr – center
- 16    • midL – middle left
- 17    • midR – middle right
- 18    • tCtr – top center
- 19    • bL – bottom left
- 20    • bR – bottom right
- 21    • tL – top left
- 22    • tR – top right
- 23    • radial – radial

24    **5.15.6.1.25       Node Horizontal Alignment**

25    The simple type ST\_NodeHorizontalAlignment defines the alignment of a node in the horizontal direction.  
26    The different alignments are:

- 27    • l – left
- 28    • ctr – center
- 29    • r – right

30    **5.15.6.1.26       Node Vertical Alignment**

31    The simple type ST\_NodeVerticalAlignment defines the alignment of a node in the vertical direction. The  
32    different alignments are:

- 33    • t – top
- 34    • mid – mid

- 1       • b – bottom

2   **5.15.6.1.27      Fallback Dimension**

3   The simple type ST\_FallbackDimension defines how many dimensions fallback will resize the diagram in. The  
4   different options for fallback dimension are:

- 5       • 1D – fallback occurs in a single dimension (X or Y)  
6       • 2D – fallback occurs in two dimensions (X and Y)

7   **5.15.6.1.28      Text Direction**

8   The simple type ST\_TextDirection specifies where the text on the first node starts. The different text  
9   directions are:

- 10      • fromT – from top  
11      • fromB – from bottom

12   **5.15.6.1.29      Pyramid Accent Position**

13   The simple type ST\_PyramidAccentPosition defines where the position of the fly-out grandchildren. The  
14   possible positions are:

- 15      • bef – before  
16      • after – after

17   **5.15.6.1.30      Pyramid Text Margin**

18   The simple type ST\_PyramidAccentTextMargin specifies the alignment of the text in the fly-out  
19   grandchildren. The different alignments are:

- 20      • step – the text is against the edge of the pyramid  
21      • stack – the text aligns

22   **5.15.6.1.31      Text Block Direction**

23   The simple type ST\_TextBlockDirection defines the text block direction. The different direction the text can  
24   have are:

- 25      • vert – vertical  
26      • horz – horizontal

27   **5.15.6.1.32      Text Anchor Horizontal**

28   The simple type ST\_AnchorHorizontal is responsible for anchoring text horizontally. The available options  
29   are:

- 30      • none – no anchor set  
31      • ctr – the text is anchored the center

## 1    5.15.6.1.33      Text Anchor Vertical

2    The simple type ST\_AnchorVertical is responsible for anchoring the text vertically. The available options are:

- 3        • t – top anchor
- 4        • mid – middle anchor
- 5        • b – bottom anchor

## 6    5.15.6.1.34      Text Alignment

7    The simple type ST\_TextAlignment defines the text alignment. The available options are:

- 8        • l – left
- 9        • ctr – ctr
- 10      • r – right

## 11    5.15.6.1.35      Auto Text Rotation

12    The simple type ST\_AutoTextRotation defines the behavior of the text as the containing shape is rotated. The  
13    following options are available:

- 14      • none – no rotation, the text rotates with the shape
- 15      • upr – upright
- 16      • grav – gravity

## 17    5.15.6.1.36      Grow Direction

18    The simple type ST\_GrowDirection defines the growing behavior of the snake algorithm. The following  
19    options are available:

- 20      • tL – grow from the top left
- 21      • tR – grow from the top right
- 22      • bL – grow from the bottom left
- 23      • gR – grow from the bottom right

## 24    5.15.6.1.37      Flow Direction

25    The simple type ST\_FlowDirection Specifies whether nodes are arranged in rows or columns for the snake  
26    algorithm. The following options are available:

- 27      • row – Row
- 28      • col – column

## 29    5.15.6.1.38      Continue Direction

30    The simple type ST\_ContinueDirection specifies the direction of the subsequent row or column in the snake  
31    algorithm. The following options are available:

- 32      • revDir – reverse direction

- 1       • sameDir – same direction

2     5.15.6.1.39      Breakpoint

3     The simple type ST\_Breakpoint defines the behavior of a snake diagram's breaking behavior. The available  
4     options are:

- 5       • endCnv – end of canvas  
6       • bal – balanced  
7       • fixed – fixed

8     5.15.6.1.40      Offset

9     The simple type ST\_Offset defines the behavior of whether each row or column in the snake algorithm is offset  
10    from the previous row or column. The available options are:

- 11       • ctr – the offset is center based  
12       • off – there is an offset defined

13    5.15.6.1.41      Hierarchy Alignment

14    The simple type ST\_HierarchyAlignment specifies the relationship between parent and children in a hierarchy  
15    diagram. The following options exist:

- 16       • tL – top left  
17       • tR – top right  
18       • tCtrCh – top center children  
19       • tCtrDes – top center descendants  
20       • bL – bottom left  
21       • bR – bottom right  
22       • bCtrCh – bottom center children  
23       • bCtrDes – bottom center descendants  
24       • lT – left top  
25       • lB – left bottom  
26       • lCtrCh – left center children  
27       • lCtrDes – left center descendants  
28       • rT – right top  
29       • rB – right bottom  
30       • rCtrCh – right center children  
31       • rCtrDes – right center descendants

32    5.15.6.2      Variable Type

33    The simple type ST\_VariableType defines the type of the conditional expression. These variables turn user  
34    interface options on and off. The available variable types are:

- 1     • unknown – Unknown variable type. This can be used by implementing applications to define a  
2       variable type outside of the scope of what is defined by this simple type.
- 3     • orgChart – organizational chart command
- 4     • chMax – used for the insert shape dropdown commands
- 5     • chPref – used for the insert shape button
- 6     • bulEnabled – used for the insert bullet command
- 7     • dir – diagram direction, RTL or LTR
- 8     • hierBranch – stores the different layouts for org chart
- 9     • animOne – exposes options for animation
- 10    • animLvl – exposes options for animation

#### 11    5.15.6.2.1      Output Shape Type

12   The simple type ST\_OutputShapeType defines special shape types which are unique to diagrams. The unique  
13   types are:

- 14    • none – none
- 15    • conn – connector

#### 16   5.15.6.3      Diagram Definitions

17   Diagram definitions define the look of a diagram. They utilize almost all the aspects of the file format  
18   discussed thus far in order to create layout properties which get translated into visual diagrams.

19   There are a few more simple types that need to be defined before talking about the larger aspects of what it  
20   takes to create a diagram definition. Many of these simple types are provided as wrappers or lists of the above  
21   mentioned simple types.

#### 22   5.15.6.3.1      Lists

23   There are a group of simple types which act as lists of the simple types already mentioned. They are all  
24   defined in the following general way:

```
25   <xsd:simpleType name= NAME OF TYPE >
26    <xsd:list itemType= NAME OF SIMPLE TYPE />
27   </xsd:simpleType>
```

28   The list of these list types are the following simple types:

- 29    • ST\_AxisTypes – list of ST\_AxisType
- 30    • ST\_ElementTypes – list of ST\_ElementType
- 31    • ST\_Ints – list of xsd:int
- 32    • ST\_UnsignedInts – list of xsd:unsignedInt
- 33    • ST\_Booleans – list of xsd:boolean

1    5.15.6.3.2              Function Value

2    The simple type ST\_FunctionValue is a value for a condition expression. It is defined as:

```
3       <xsd:simpleType name="ST_FunctionValue" final="restriction">
4        <xsd:union memberTypes="xsd:int xsd:boolean ST_Direction
5          ST_HierBranchStyle ST_AnimOneStr ST_AnimLvlStr" />
6       </xsd:simpleType>
```

7    5.15.6.3.3              Direction

8    The simple type ST\_Direction defines the direction the diagram is to be laid out. The directions available are:

- 9       • norm – normal
- 10      • rev – reversed

11   5.15.6.3.4              Hierarchy Branch Style

12   The simple type ST\_HierBranchStyle changes the behavior of the branch style in hierarchy, or org chart, diagrams. This value can be modified by a user directly from the user interface. The different types of branch styles are:

- 15      • l – left
- 16      • r – right
- 17      • hang – hanging
- 18      • std – standard
- 19      • init – initial

20   5.15.6.3.5              One by One Animation

21   The simple type ST\_AnimOneStr allows for differentiation in the way a one-by-one animation is displayed in the user interface. The following options are available:

- 23      • none – nothing is displayed
- 24      • one – the term one-by-one is used
- 25      • branch – the term branch one-by-one is used to distinguish a hierarchy diagram

26   5.15.6.3.6              Level Animation

27   The simple type ST\_AnimLvlStr acts very much like the type ST\_AminOneStr, as it allows for two different descriptions of a single animation depending upon the desired behavior for a particular diagram. The following allows for differentiation of radial diagrams. The different options are:

- 30      • none – nothing
- 31      • lvl – normal depth first traversal
- 32      • ctr – allows a radial diagram to be shown with the center node first

1    5.15.6.3.7        Org Chart Flag

2    The complex type CT\_OrgChart defines that the diagram will be an organizational chart. Organizational charts  
 3    contain special behavior in that assistants can now be utilized correctly. The complex type is defined as  
 4    follows:

```
5       <xsd:complexType name="CT_OrgChart">
6        <xsd:attribute name="val" type="xsd:boolean" default="false"
7        use="optional" />
8     </xsd:complexType>
```

9    5.15.6.3.8        Node Count

10   The simple type ST\_NodeCount holds a value that is used by the complex types CT\_ChildMax and  
 11   CT\_ChildPref.

12   5.15.6.3.9        Child Max

13   This complex type defines when the user interface for inserting a child shape is to become disabled for a given  
 14   node, or rather the max number of children that the user interface will be enabled for. This complex type is  
 15   defined as:

```
16      <xsd:complexType name="CT_ChildMax">
17        <xsd:attribute name="val" type="ST_NodeCount" default="-1"
18        use="optional" />
19     </xsd:complexType>
```

20   5.15.6.3.10        Child Preference

21   This complex type defines how many children are inserted with a single action through the user interface to  
 22   add a child. This is useful in hierarchy diagrams in which one would like to specify that every shape should  
 23   have three children. A single click of the add shape button would add three children. The complex type is  
 24   defined as follows:

```
25      <xsd:complexType name="CT_ChildPref">
26        <xsd:attribute name="val" type="ST_NodeCount" default="-1"
27        use="optional" />
28     </xsd:complexType>
```

29   5.15.6.3.11        Bullets Enabled

30   This complex type defines if the user interface for inserting a bullet into a shape is enabled or disabled for a  
 31   given node. The complex type is defined as follows:

```
32      <xsd:complexType name="CT_BulletEnabled">
33        <xsd:attribute name="val" type="xsd:boolean" default="false"
34        use="optional" />
35     </xsd:complexType>
```

1    5.15.6.3.12      Direction

2    This complex type defines the direction of the diagram, be it normal or reversed. The complex type is defined  
3    as:

```
4       <xsd:complexType name="CT_Direction">
5        <xsd:attribute name="val" type="ST_Direction" default="norm"
6          use="optional" />
7       </xsd:complexType>
```

8    5.15.6.3.13      Hierarchy Branch Style

9    This complex type defines the hierarchy branch style for a diagram. The complex type is defined as:

```
10      <xsd:complexType name="CT_HierBranchStyle">
11       <xsd:attribute name="val" type="ST_HierBranchStyle" default="std"
12          use="optional" />
13       </xsd:complexType>
```

14    5.15.6.3.14      Animate as One

15    This complex type defines the animate as one value for a diagram. The complex type is defined as:

```
16       <xsd:complexType name="CT_AnimOne" >
17       <xsd:attribute name="val" type="ST_AnimOneStr" default="one"
18          use="optional" />
19       </xsd:complexType>
```

20    5.15.6.3.15      Animate by Level

21    This complex type defines the animate by level value for a diagram. The complex type is defined as:

```
22       <xsd:complexType name="CT_AnimLvl">
23       <xsd:attribute name="val" type="ST_AnimLvlStr" default="none"
24          use="optional" />
25       </xsd:complexType>
```

26    5.15.6.3.16      Layout Property Set

27    The complex type CT\_LayoutPropertySet holds all of the layout properties for a given diagram. The layout  
28    property set is a single structure which contains most of what has been talked about thus far in a diagram  
29    definition. The layout property set is defined as:

```
30       <xsd:complexType name="CT_LayoutVariablePropertySet">
31       <xsd:sequence>
32        <xsd:element name="orgChart" type="CT_OrgChart"
33           minOccurs="0" maxOccurs="1" />
34        <xsd:element name="chMax" type="CT_ChildMax" minOccurs="0"
35           maxOccurs="1" />
```

```

1   <xsd:element name="chPref" type="CT_ChildPref"
2     minOccurs="0" maxOccurs="1" />
3   <xsd:element name="bulletEnabled" type="CT_BulletEnabled"
4     minOccurs="0" maxOccurs="1" />
5   <xsd:element name="dir" type="CT_Direction" minOccurs="0"
6     maxOccurs="1" />
7   <xsd:element name="hierBranch" type="CT_HierBranchStyle"
8     minOccurs="0" maxOccurs="1" />
9   <xsd:element name="animOne" type="CT_AnimOne" minOccurs="0"
10    maxOccurs="1" />
11  <xsd:element name="animLvltype" type="CT_AnimLvl" minOccurs="0"
12    maxOccurs="1" />
13  </xsd:sequence>
14 </xsd:complexType>
```

15 Because all of the contents of this complex type have already been discussed, no further detail on this complex  
16 type needs to be given.

#### 17 5.15.6.3.17 Iterators

18 The attribute group AG\_IteratorAttributes defines the attributes used by the iterators forEach, presOf, and if.  
19 The attribute group is defined as follows:

```

20  <xsd:attributeGroup name="AG_IteratorAttributes">
21    <xsd:attribute name="axis" type="ST_AxisTypes" use="optional"
22      default="none" />
23    <xsd:attribute name="ptType" type="ST_ElementTypes"
24      use="optional" default="all" />
25    <xsd:attribute name="hideLastTrans" type="ST_Booleans"
26      use="optional" default="true" />
27    <xsd:attribute name="st" type="ST_Ints" use="optional" default="1" />
28    <xsd:attribute name="cnt" type="ST_UnsignedInts" use="optional"
29      default="0" />
30    <xsd:attribute name="step" type="ST_Ints" use="optional" default="1" />
31  </xsd:attributeGroup>
```

#### 32 5.15.6.3.18 Constraints

33 The attribute group AG\_ConstraintAttributes defines the attributes used to specify a constraint. The attribute  
34 group is defined as:

```

35  <xsd:attributeGroup name="AG_ConstraintAttributes">
36    <xsd:attribute name="type" type="ST_ConstraintType" use="required" />
37    <xsd:attribute name="for" type="ST_ConstraintRelationship"
38      use="optional" default="self" />
39    <xsd:attribute name="forName" type="xsd:IDREF" use="optional" />
```

```

1   <xsd:attribute name="ptType" type="ST_ElementType" use="optional"
2     default="all" />
3   </xsd:attributeGroup>
```

#### 4 5.15.6.3.19 Constraint References

5 The attribute group AG\_ConstraintRefAttributes defines the attributes used to specify a constraint reference.  
6 The attribute group is defined as:

```

7   <xsd:attributeGroup name="AG_ConstraintRefAttributes">
8     <xsd:attribute name="refType" type="ST_ConstraintType"
9       use="optional" default="unknown" />
10    <xsd:attribute name="refFor" type="ST_ConstraintRelationship"
11      use="optional" default="self" />
12    <xsd:attribute name="refForName" type="xsd:IDREF"
13      use="optional" />
14    <xsd:attribute name="refPtType" type="ST_ElementType"
15      use="optional" default="all" />
16  </xsd:attributeGroup>
```

#### 17 5.15.6.3.20 Constraint

18 The complex type CT\_Constraint define a constraint within the layout framework. A constraint acts as a limit  
19 or sets a value to a given parameter in a diagram definition, for example, it can be used to specify that all  
20 nodes of a give point type are the same size. A constraint is defined as:

```

21   <xsd:complexType name="CT_Constraint">
22     <xsd:attributeGroup ref="AG_ConstraintAttributes" />
23     <xsd:attributeGroup ref="AG_ConstraintRefAttributes" />
24     <xsd:attribute name="op" type="ST_BooleanOperator" use="optional"
25       default="none" />
26     <xsd:attribute name="val" type="xsd:double" use="optional"
27       default="0" />
28     <xsd:attribute name="fact" type="xsd:double" use="optional"
29       default="1" />
30   </xsd:complexType>
```

#### 31 5.15.6.3.21 Constraint List

32 The complex type CT\_Constraints is a sequence of CT\_Constraint complex types. It is defined as:

```

33   <xsd:complexType name="CT_Constraints">
34     <xsd:sequence>
35       <xsd:element name="constr" type="CT_Constraint" minOccurs="0"
36         maxOccurs="unbounded" />
37     </xsd:sequence>
38   </xsd:complexType>
```

1    5.15.6.3.22        Rule

2    The complex type CT\_NumericRule defines a layout framework constraint rule. Rules are run after the  
 3    diagram is created in order to specify what happens when the diagram doesn't fully fit within the bounds. This  
 4    allows for specific behavior to be defined rather than using default rules for fitting the diagram. A rule is  
 5    defined in the following way:

```
6      <xsd:complexType name="CT_NumericRule" >
7          <xsd:attributeGroup ref="AG_ConstraintAttributes" />
8          <xsd:attribute name="val" type="xsd:double" use="optional"
9              default="NaN" />
10         <xsd:attribute name="fact" type="xsd:double" use="optional"
11             default="NaN" />
12         <xsd:attribute name="max" type="xsd:double" use="optional"
13             default="NaN" />
14     </xsd:complexType>
```

15    5.15.6.3.23        Rule List

16    The complex type CT\_Rules is simply a list of CT\_NumericRule complex types. It is defined in the following  
 17    manner:

```
18      <xsd:complexType name="CT_Rules">
19          <xsd:sequence>
20              <xsd:element name="rule" type="CT_NumericRule"
21                  minOccurs="0" maxOccurs="unbounded" />
22          </xsd:sequence>
23      </xsd:complexType>
```

24    5.15.6.3.24        Presentation Of

25    The complex type CT\_PresentationOf defines the mapping between data and the diagram. The complex type  
 26    is defined in the following manner:

```
27      <xsd:complexType name="CT_PresentationOf">
28          <xsd:attributeGroup ref="AG_IteratorAttributes" />
29      </xsd:complexType>
```

30    5.15.6.3.25        Layout Shape

31    The simple type ST\_LayoutShapeType is a simple type that contains all of the shapes available which can be  
 32    used within a diagram. The simple type is defined as a union of ST\_OutputShapeType and an externally  
 33    defined ST\_ShapeType.

34    5.15.6.3.26        Index1

35    The simple type ST\_Index1 defines a 1-based index that is used to index values elsewhere. The simple type is  
 36    defined as:

```

1   <xsd:simpleType name="ST_Index1">
2     <xsd:restriction base="xsd:unsignedInt">
3       <xsd:minInclusive value="1" />
4     </xsd:restriction>
5   </xsd:simpleType>
```

#### 6 5.15.6.3.27 Adjust Handle

7 The complex type CT\_Adj specifies a shape adjust handle modification. The shapes within a diagram can be  
8 modified based on their adjust handles, for example, the radius of the corner rounding in a rounded rectangle  
9 can be adjusted using this complex type. The complex type is defined in the following manner:

```

10  <xsd:complexType name="CT_Adj">
11    <xsd:attribute name="idx" type="ST_Index1" use="required" />
12    <xsd:attribute name="val" type="xsd:double" use="required" />
13  </xsd:complexType>
```

#### 14 5.15.6.3.28 Adjust Handle List

15 The complex type CT\_AdjLst holds all of the adjust handles for a given shape. The number of adjust handles  
16 accessible varies shape by shape, but there are usually less than four for a given shape. The complex type is  
17 defined in the following way:

```

18  <xsd:complexType name="CT_AdjLst" o:cname="CAdjList">
19    <xsd:sequence>
20      <xsd:element name="adj" type="CT_Adj" minOccurs="0"
21        maxOccurs="unbounded" />
22    </xsd:sequence>
23  </xsd:complexType>
```

#### 24 5.15.6.3.29 Shape

25 The complex type CT\_Shape specifies a shape for a layout node. The shape complex type holds all of the  
26 information associated with the particular layout node and all of the adjustments or modifications that can be  
27 made to the shape. The rot attribute specifies a rotation on the shape. The blip attribute specifies an image  
28 that is used as a background fill for the shape and the blipPhldr attribute specifies whether or not the shape  
29 shows up with an image placeholder. The zOrderOff attribute specifies an offset to be used for the z-ordering  
30 of this shape, while the lkTxEntry attribute prevents text editing within the shape. A shape is defined in the  
31 following manner:

```

32  <xsd:complexType name="CT_Shape">
33    <xsd:sequence>
34      <xsd:element name="adjLst" type="CT_AdjLst" minOccurs="0"
35        maxOccurs="1" />
36    </xsd:sequence>
37    <xsd:attribute name="rot" type="xsd:double" use="optional" default="0" />
```

```

1   <xsd:attribute name="type" type="ST_LayoutShapeType"
2     use="optional" default="none" />
3   <xsd:attribute ref="r:blip" use="optional" />
4   <xsd:attribute name="zOrderOff" type="xsd:int" use="optional"
5     default="0" />
6   <xsd:attribute name="hideGeom" type="xsd:boolean" use="optional"
7     default="false" />
8   <xsd:attribute name="lkTxEntry" type="xsd:boolean" use="optional"
9     default="false" />
10  <xsd:attribute name="blipPhldr" type="xsd:boolean" use="optional"
11    default="false" />
12  </xsd:complexType>
```

### 5.15.6.3.30 Parameter

The complex type CT\_Parameter holds the information regarding an algorithm parameter. The complex type is defined as:

```

16  <xsd:complexType name="CT_Parameter">
17    <xsd:attribute name="type" type="ST_ParameterId" use="required" />
18    <xsd:attribute name="val" type="xsd:string" use="required" />
19  </xsd:complexType>
```

### 5.15.6.3.31 Algorithm

The complex type CT\_Algorithm defines the algorithm which the diagram will use to layout the nodes which contain the data. Also defined here are the optional list of parameters which are associated with this algorithm and modify its behavior. An algorithm is defined in the following manner:

```

24  <xsd:complexType name="CT_Algorithm" >
25    <xsd:sequence>
26      <xsd:element name="param" type="CT_Parameter" minOccurs="0"
27        maxOccurs="unbounded" />
28    </xsd:sequence>
29    <xsd:attribute name="type" type="ST_AlgorithmType" use="required" />
30    <xsd:attribute name="rev" type="xsd:unsignedInt" use="optional"
31      default="0" />
32  </xsd:complexType>
```

### 5.15.6.3.32 Layout Node

The complex type CT\_LayoutNode is the main building block of a diagram. A layout node contains enough information to lay out itself and its children. The name attribute is simply a unique string given to the layout node. The styleLbl attribute references the style label that is used to style the layout node. This style label has already been defined in this document. A layout node is defined in the following manner:

```

1   <xsd:complexType name="CT_LayoutNode">
2     <xsd:choice minOccurs="0" maxOccurs="unbounded">
3       <xsd:element name="alg" type="CT_Algorithm" minOccurs="0"
4         maxOccurs="1" />
5       <xsd:element name="shape" type="CT_Shape" minOccurs="0"
6         maxOccurs="1" />
7       <xsd:element name="presOf" type="CT_PresentationOf"
8         minOccurs="0" maxOccurs="1" />
9       <xsd:element name="constrLst" type="CT_Constraints"
10        minOccurs="0" maxOccurs="1" />
11      <xsd:element name="ruleLst" type="CT_Rules" minOccurs="0"
12        maxOccurs="1" />
13      <xsd:element name="varLst"
14        type="CT_LayoutVariablePropertySet" minOccurs="0"
15        maxOccurs="1" />
16      <xsd:element name="forEach" type="CT_ForEach" />
17      <xsd:element name="layoutNode" type="CT_LayoutNode" />
18      <xsd:element name="choose" type="CT_ChOOSE" />
19    </xsd:choice>
20    <xsd:attribute name="name" type="xsd:ID" use="optional" />
21    <xsd:attribute name="styleLbl" type="xsd:string" use="optional" />
22    <xsd:attribute name="chOrder" type="ST_ChildOrderType"
23      use="optional" default="b" />
24    <xsd:attribute name="moveWith" type="xsd:IDREF" use="optional" />
25  </xsd:complexType>

```

#### 26 5.15.6.3.33 For Each

27 The complex type CT\_ForEach defines a for each iterator. The iteration behaves as if it were a for each loop.  
28 The complex type is defined as:

```

29  <xsd:complexType name="CT_ForEach">
30    <xsd:choice minOccurs="0" maxOccurs="unbounded">
31      <xsd:element name="alg" type="CT_Algorithm" minOccurs="0"
32        maxOccurs="1" />
33      <xsd:element name="shape" type="CT_Shape" minOccurs="0"
34        maxOccurs="1" />
35      <xsd:element name="presOf" type="CT_PresentationOf"
36        minOccurs="0" maxOccurs="1" />
37      <xsd:element name="constrLst" type="CT_Constraints"
38        minOccurs="0" maxOccurs="1" />
39      <xsd:element name="ruleLst" type="CT_Rules" minOccurs="0"
40        maxOccurs="1" />

```

```

1   <xsd:element name="forEach" type="CT_ForEach" />
2   <xsd:element name="layoutNode" type="CT_LayoutNode" />
3   <xsd:element name="choose" type="CT_ChOOSE" />
4 </xsd:choice>
5   <xsd:attribute name="name" type="xsd:ID" use="optional" />
6   <xsd:attribute name="ref" type="xsd:IDREF" use="optional" />
7   <xsd:attributeGroup ref="AG_IteratorAttributes" />
8 </xsd:complexType>
```

#### 9 5.15.6.3.34 When

10 The complex type CT\_When defines an if conditional expression. The complex type is usually used in  
11 conjunction with the else counterpart which is defined next. The CT\_When complex type is defined in the  
12 following manner:

```

13  <xsd:complexType name="CT_When">
14    <xsd:choice minOccurs="0" maxOccurs="unbounded">
15      <xsd:element name="alg" type="CT_Algorithm" minOccurs="0"
16        maxOccurs="1" />
17      <xsd:element name="shape" type="CT_Shape" minOccurs="0"
18        maxOccurs="1" />
19      <xsd:element name="presOf" type="CT_PresentationOf"
20        minOccurs="0" maxOccurs="1" />
21      <xsd:element name="constrLst" type="CT_Constraints"
22        minOccurs="0" maxOccurs="1" />
23      <xsd:element name="ruleLst" type="CT_Rules" minOccurs="0"
24        maxOccurs="1" o:cname="Rules" />
25      <xsd:element name="forEach" type="CT_ForEach" />
26      <xsd:element name="layoutNode" type="CT_LayoutNode" />
27      <xsd:element name="choose" type="CT_ChOOSE" />
28    </xsd:choice>
29    <xsd:attribute name="name" type="xsd:ID" use="optional" />
30    <xsd:attributeGroup ref="AG_IteratorAttributes" />
31    <xsd:attribute name="func" type="ST_FunctionType"
32      use="required" />
33    <xsd:attribute name="arg" type="ST_FunctionArgument"
34      use="optional" />
35    <xsd:attribute name="op" type="ST_FunctionOperator"
36      use="required" />
37    <xsd:attribute name="val" type="ST_FunctionValue"
38      use="required" />
39 </xsd:complexType>
```

1    5.15.6.3.35      Otherwise

2    The complex type CT\_Otherwise is the else counterpart to the already defined if conditional expression. The  
3    complex type is defined as:

```

4      <xsd:complexType name="CT_Otherwise" o:cname="DDOtherwise">
5          <xsd:choice minOccurs="0" maxOccurs="unbounded">
6              <xsd:element name="alg" type="CT_Algorithm" minOccurs="0"
7                  maxOccurs="1" />
8              <xsd:element name="shape" type="CT_Shape" minOccurs="0"
9                  maxOccurs="1" />
10             <xsd:element name="presOf" type="CT_PresentationOf"
11                 minOccurs="0" maxOccurs="1" />
12             <xsd:element name="constrLst" type="CT_Constraints"
13                 minOccurs="0" maxOccurs="1" />
14             <xsd:element name="ruleLst" type="CT_Rules" minOccurs="0"
15                 maxOccurs="1" />
16             <xsd:element name="forEach" type="CT_ForEach" />
17             <xsd:element name="layoutNode" type="CT_LayoutNode" />
18                 <xsd:element name="choose" type="CT_ChOOSE" />
19             </xsd:choice>
20             <xsd:attribute name="name" type="xsd:ID" use="optional" />
21         </xsd:complexType>
```

22    5.15.6.3.36      Choose Statement

23    The complex type CT\_ChOOSE packages together the if and else conditions into an actual if/else statement.  
24    The complex type is defined in the following manner:

```

25     <xsd:complexType name="CT_ChOOSE" o:cname="DDChoose">
26         <xsd:sequence>
27             <xsd:element name="if" type="CT_When" maxOccurs="unbounded" />
28             <xsd:element name="else" type="CT_Otherwise" minOccurs="0" />
29         </xsd:sequence>
30         <xsd:attribute name="name" type="xsd:ID" use="optional" />
31     </xsd:complexType>
```

32    5.15.6.3.37      Sample Data

33    The complex type CT\_SampleData defines how the data model is to be populated in an initial manner. The  
34    complex type holds a temporary data model when there is no data model present in order to display a diagram  
35    on an initial insert. The complex type is defined by:

```

36     <xsd:complexType name="CT_SampleData">
37         <xsd:sequence>
38             <xsd:element name="dataModel" type="CT_DataModel" minOccurs="0" />
39         </xsd:sequence>
```

```

1      <xsd:attribute name="useDef" type="xsd:boolean" use="optional"
2          default="false" />
3      </xsd:complexType>
```

#### 4 5.15.6.3.38 Common Structures

5 CT\_Category, CT\_Categories, CT\_Name, CT\_Description, and ST\_Version are defined just as their counterparts  
6 in the subclauses above, and they perform the same tasks.

#### 7 5.15.6.3.39 Diagram Definition

8 The complex type CT\_DiagramDefinition is the root element for a diagram definition. It is defined in the  
9 following manner:

```

10     <xsd:complexType name="CT_DiagramDefinition">
11         <xsd:sequence>
12             <xsd:element name="title" type="CT_Name" minOccurs="0"
13                 maxOccurs="unbounded" />
14             <xsd:element name="desc" type="CT_Description"
15                 minOccurs="0" maxOccurs="unbounded" />
16             <xsd:element name="catLst" type="CT_Categories" minOccurs="0" />
17             <xsd:element name="sampData" type="CT_SampleData" minOccurs="0" />
18             <xsd:element name="styleData" type="CT_SampleData" minOccurs="0" />
19             <xsd:element name="clrData" type="CT_SampleData" minOccurs="0" />
20             <xsd:element name="layoutNode" type="CT_LayoutNode" />
21         </xsd:sequence>
22         <xsd:attribute name="uniqueId" type="xsd:anyURI" use="optional" />
23         <xsd:attribute name="minVer" type="ST_Version" use="optional"
24             default="12.0" />
25         <xsd:attribute name="defStyle" type="xsd:anyURI" use="optional" />
26     </xsd:complexType>
```

# 1    6. Introduction to VML

2    **This clause is informative.**

3    This clause contains a detailed introduction to the components of Vector Markup Language (VML).

## 4    6.1    Introduction

5    This section provides an overview of the most common parts of VML. The VML format is a legacy format  
6    originally introduced with Office 2000 and is included and fully defined in this specification for backwards  
7    compatibility reasons. The DrawingML format is a newer and richer format created with the goal of eventually  
8    replacing any uses of VML in the Office Open XML formats. VML should be considered a deprecated format  
9    included in Office Open XML for legacy reasons only and new applications that need a file format for drawings  
10   are strongly encouraged to use preferentially DrawingML .

11   VML is an XML-based exchange, editing, and delivery format for high-quality vector graphics. VML facilitates  
12   the exchange and subsequent editing of vector graphics between a wide variety of productivity and design  
13   applications. VML is based on XML 1.0, which is an open, simple, text-based language for describing structured  
14   data. VML also supports other World Wide Web Consortium standards, such as Cascading Style Sheets 2.0  
15   (CSS), which specifies style information and 2-D positioning.

16   As the VML format is a format provided for backward compatibility, many VML elements are defined in the  
17   same urn:schemas-microsoft-com:vml namespace that is currently used by millions of documents  
18   already using VML. In the documentation this is typically shortened to a v: prefix in the VML tag by defining  
19   xmlns:v="urn:schemas-microsoft-com:vml". The namespaces used for VML are legacy namespaces.  
20   Once again, VML should be considered a deprecated format included in Office Open XML for legacy reasons  
21   only and new applications that need a file format for drawings are strongly encouraged to use preferentially  
22   DrawingML .

23   Additional elements and attributes are defined in namespaces that reflect how they are used (all VML  
24   namespaces defined in this Standard maintain the legacy namespace structure for backward compatibility):

- 25   • urn:schemas-microsoft-com:office:office (office document)
- 26   • urn:schemas-microsoft-com:office:word (word-processing document)
- 27   • urn:schemas-microsoft-com:office:excel (spreadsheet document)
- 28   • urn:schemas-microsoft-com:office:powerpoint (presentation document)

## 29   6.2    Shape Element

30   The Shape element is the basic building block of VML. A shape may exist on its own or within a Group  
31   element. Shape defines many attributes and sub-elements that control the look and behavior of the shape. A

- 1 shape must define at least a Path and size (Width, Height). VML also uses properties of the CSS2 style attribute to specify positioning and sizing.
- 3 Note that this subclause also applies to the set of pre-defined shape primitives provided by the VML elements Arc, Curve, Image, Line, Oval, Polyline, Rect, and RoundRect.
- 5 The following attributes are used to define a minimal shape:

Attribute	Description
FillColor	Brush color that fills the closed path of a shape.
Position	Type of positioning used to place an element.
Top	Position of the shape relative to the element above it in the flow of the page.
Left	Position of the shape relative to the element left of it in the document flow.
Width	Width of the shape.
Height	Height of the shape.
Path	Line that makes up the edges of a shape.

- 6
  - 7 The following example creates a minimal shape:
- ```
<v:shape fillcolor="green"
  style="position:relative;top:1;left:1;width:50;
  height:50" path="m 1,1 l 1,50, 50,50, 50,1 x e">
</v:shape>
```
- 

- 8 Although there is no official categorization of the Shape element's attributes or sub-elements, it is useful to think of them in groups. The following sections broadly describe the characteristics of the Shape element. A few fundamental attributes and elements are introduced here. For complete details, see the VML reference in Part 4.

## 6.2.1 Geometry

- 13 The following attributes affect the basic structure or outline of the shape.

| Attribute | Description                                           |
|-----------|-------------------------------------------------------|
| Adj       | Adjustment value used to define values for a formula. |
| Height*   | Height of the shape.                                  |
| Path      | Line that makes up the edges of a shape.              |
| Width*    | Width of the shape.                                   |

- 14
- 15 \* indicates a CSS2 style property

| Element | Description |
|---------|-------------|
|         |             |

| Element   | Description                       |
|-----------|-----------------------------------|
| Callout   | Defines a callout for a shape.    |
| Extrusion | Defines an extrusion for a shape. |
| Path      | Defines a path for a shape.       |
| Skew      | Defines a skew for a shape.       |
| Stroke    | Defines a stroke for a shape.     |
| TextBox   | Defines a textbox for a shape.    |
| TextPath  | Defines a text path for a shape.  |

1    **6.2.1.1      Height and Width Attributes**

2    Height and Width may be specified using any of the following units. If no unit is specified, pixels is assumed.

| Relative |                              |
|----------|------------------------------|
| em       | Height of the element's font |
| ex       | Height of the letter "x"     |
| px       | Pixels                       |
| %        | Percentage                   |

3

| Absolute |             |
|----------|-------------|
| in       | Inches      |
| cm       | Centimeters |
| mm       | Millimeters |
| pt       | Points      |
| pc       | Picas       |

4

5    For example:

```
6       style="position:relative;top:1;left:1;width:50;height:50"
7       style="position:relative;top:1;left:1;width:10%;height:10%"
```

8    **6.2.1.2      Path Attribute**

9    The Path attribute contains specially formatted text that describes a set of points and line connections  
10   between them that define the shape's outline. The path defined must be closed. A path is begun by  
11   specifying **m** and a coordinate. This indicates a moveto the given coordinate. Line segments are drawn using **l**  
12   (**lineto**) and specifying subsequent coordinates. A line is closed with **x** after the closing coordinate. The path is  
13   ended with **e**.

14   For example:

1        path="m 1,1 l 1,50, 50,50, 50,1 x e"

2        This starts at (1,1), draws a line to (1,50), (50,50) and (50,1), where the line is closed and the path ended.

3        The coordinates specified correspond to relative coordinate space (the size of units in relative space can be set by the CoordSize attribute). The shape's actual size is determined by the Height and Width attributes.

5        For example:

```
<v:shape style="position:relative;top:1;left:1;width:5000;
height:5000" fillcolor="teal"
path="m 1,1 l 1,10 10,10 10,1 1,1 x e" />
```



```
<v:shape style="position:relative;top:1;left:1;width:2500;
height:2500" fillcolor="teal"
path="m 1,1 l 1,10 10,10 10,1 1,1 x e" />
```



6        More than one closed line path may be specified in the Path attribute and each closed region is filled.

```
<v:shape style="position:relative;top:1;left:1;width:5000;
height:5000" fillcolor="teal"
path="m 1,1 l 1,10 10,10 10,1 1,1 x m 20,20 l 20,40 40,40
40,20 20,20 x e" />
```



7        The optional Path element, which allows for the creation of more complex paths and regions, overrides the Path attribute if it is specified.

### 9      6.2.2     Placement

10      These attributes affect the layout and placement of shapes. Placement may be defined relative to other shapes or non-VML content that also exists in the container holding the shape.

| Attribute      | Description                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------|
| AllowOverlap   | Determines if a shape can overlap other shapes.                                               |
| CoordOrigin    | Specifies the coordinate unit origin of the rectangle that bounds a shape.                    |
| CoordSize      | Specifies the horizontal and vertical units of the rectangle that bounds a shape.             |
| Flip*          | Switches the orientation of a shape.                                                          |
| Left*          | Determines the position of the shape relative to the element left of it in the document flow. |
| Margin-Bottom* | Specifies the bottom edge of the shape's containing rectangle relative to the shape anchor.   |
| Margin-Left*   | Specifies the left edge of the shape's containing rectangle relative to the shape anchor.     |

| Attribute                         | Description                                                                                 |
|-----------------------------------|---------------------------------------------------------------------------------------------|
| Margin-Right*                     | Specifies the right edge of the shape's containing rectangle relative to the shape anchor.  |
| Margin-Top*                       | Specifies the top edge of the shape's containing rectangle relative to the shape anchor.    |
| MSO-Position-Horizontal*          | Specifies the horizontal positioning data for objects in WordprocessingML.                  |
| MSO-Position-Horizontal-Relative* | Specifies relative horizontal position data for objects in WordprocessingML.                |
| MSO-Position-Vertical*            | Specifies the vertical position data for objects in WordprocessingML.                       |
| MSO-Position-Vertical-Relative*   | Specifies relative vertical position data for objects in WordprocessingML.                  |
| MSO-Wrap-Distance-Bottom*         | Defines the distance from the bottom side of the shape to the text that wraps around it.    |
| MSO-Wrap-Distance-Left*           | Defines the distance from the left side of the shape to the text that wraps around it.      |
| MSO-Wrap-Distance-Right*          | Defines the distance from the right side of the shape to the text that wraps around it.     |
| MSO-Wrap-Distance-Top*            | Defines the distance from the shape top to the text that wraps around it.                   |
| MSO-Wrap-Edited*                  | Determines whether the wrap coordinates were customized by the user.                        |
| MSO-Wrap-Mode*                    | Defines the wrapping mode for text.                                                         |
| Position*                         | Defines the type of positioning used to place an element.                                   |
| RelativePosition                  | Defines a relative position for an object.                                                  |
| Rotation*                         | Defines the angle that a shape is rotated.                                                  |
| Top*                              | Defines the position of the shape relative to the element above it in the flow of the page. |
| Z-Index*                          | Determines the display order of overlapping shapes.                                         |

1

2 \* indicates a CSS2 style property

### 6.2.2.1 CoordOrigin and CoordSize Attributes

4 These attributes define the relative coordinate space of a shape. This space is scaled up or down to match the  
 5 specified Width and Height of the shape. Coordinates in the Path attribute (or element) are relative to the  
 6 space defined by CoordOrigin and CoordSize, so the Path definition never needs to change simply to scale the  
 7 shape.

1 CoordSize defines the “width” and “height” of the local coordinate space. CoordOrigin defines the top-left  
2 coordinate of this space.

3 For example:

```
coordorigin="0,0" Extents of local space are (0,0) to (200,200)
coordsize="200,200"
```

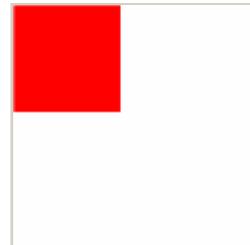
```
coordorigin="-100,-100" Extents of local space are (-100,-100) to (100,100)
coordsize="200,200"
```

4

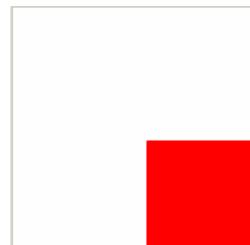
5 This local space definition affects the position of the shape. Changing the CoordOrigin translates the shape  
6 within the local space. Changing the CoordSize affects the size of the shape by changing the size of the local  
7 space relative to the shape’s Width and Height.

8 For example:

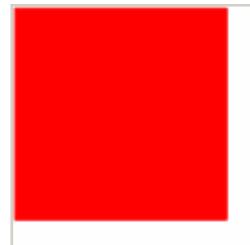
```
coordorigin="0,0"
coordsize="500,500"
style="position:absolute;top:0;left:0;width:100pt;
height:100pt"
```



```
coordorigin="-250,-250"
coordsize="500,500"
style="position:absolute;top:0;left:0;width:100pt;
height:100pt"
```



```
coordorigin="0,0"
coordsize="250,250"
style="position:absolute;top:0;left:0;width:100pt;
height:100pt"
```



### 9 6.2.2.2 Position Attribute

10 Position can be specified as “static”, “relative” or “absolute”. Static positioning keeps the shape inline with the  
11 current flow of the surrounding content – the Top and Left attributes are ignored. Relative uses the Top and

- 1 Left attributes to position the shape relative to its position in the current flow. Absolute uses the Top and Left attributes to position the shape with respect to its container.

### 6.2.3 Formatting

- 4 These attributes and elements affect the fill and line properties of the shape.

| Attribute         | Description                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------|
| BorderBottomColor | Bottom border color of an inline shape.                                                       |
| BorderLeftColor   | Left border color of an inline shape.                                                         |
| BorderRightColor  | Right border color of an inline shape.                                                        |
| BorderTopColor    | Top border color of an inline shape.                                                          |
| BWMode            | Determines how a shape will render for black-and-white output devices.                        |
| BWNormal          | Defines the black-and-white mode for normal black-and-white output devices.                   |
| BWPure            | Defines the black-and-white mode for pure black-and-white output devices.                     |
| Chromakey         | Defines a color that will be transparent and show anything behind the shape.                  |
| FillColor         | Defines the brush color that fills the closed path of a shape.                                |
| Filled            | Determines whether the closed path will be filled.                                            |
| ForceDash         | Determines whether a dashed outline is used to draw a shape when a shape has no line or fill. |
| HR                | Specifies that a shape is a horizontal rule.                                                  |
| HRAign            | Defines the alignment of a horizontal rule.                                                   |
| HRHeight          | Defines the thickness of a horizontal rule.                                                   |
| HRNoShade         | Determines whether a horizontal rule will be displayed with 3-D shading.                      |
| HRPct             | Defines the length of a horizontal rule as a percentage of page width.                        |
| HRStd             | Determines whether a shape is a standard horizontal rule.                                     |
| HRWidth           | Defines the length of a horizontal rule.                                                      |
| StrokeColor       | Defines the brush color that strokes the path of a shape.                                     |
| Stroked           | Defines whether the path will be stroked.                                                     |
| StrokeWeight      | Defines the brush thickness that strokes the path of a shape.                                 |

5

| Element   | Description                     |
|-----------|---------------------------------|
| Fill      | Defines a fill for a shape.     |
| Imagedata | Defines image data for a shape. |
| Shadow    | Defines a shadow for a shape.   |

### 6.2.4 Other

- 7 These are miscellaneous attributes and elements.

| Attribute         | Description                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Alt               | Defines alternative text to be displayed instead of a graphic.                                                              |
| AllowInCell       | Determines whether a shape can be placed in a table.                                                                        |
| Bullet            | Determines whether a shape is a graphical bullet.                                                                           |
| Button            | Determines whether a shape will be processed as a button.                                                                   |
| Class             | Refers to a definition of a CSS style.                                                                                      |
| ConnectorType     | Indicates the type of connector used for joining shapes.                                                                    |
| DoubleClickNotify | Sends an event message when a shape is double-clicked.                                                                      |
| HRef              | Defines a URL for a shape. When the shape is clicked, the browser will load the URL.                                        |
| ID                | Provides a unique identifier for an element.                                                                                |
| InsetMode         | Specifies whether the host calculates the internal text margin instead of using the inset attribute of the textbox element. |
| OLE               | Specifies whether the shape is an embedded OLE object.                                                                      |
| OLEIcon           | Determines whether an OLE object will be displayed as an icon.                                                              |
| OnEd              | Determines whether the extra handles of a shape are hidden.                                                                 |
| OnMouseOver       | Triggers a mouse event for a shape.                                                                                         |
| PreferRelative    | Determines whether the original size of an object is saved after reformatting.                                              |
| Print             | Determines whether the shape will be printed.                                                                               |
| ReGroupID         | Defines a previous group for a shape.                                                                                       |
| RuleInitiator     | Determines whether a rules engine will be used.                                                                             |
| RuleProxy         | Determines whether a proxy for the rules engine will be used.                                                               |
| Spt               | Defines a number used to identify types of shapes.                                                                          |
| TableLimits       | List of minimum height values for each row in a table.                                                                      |
| TableProperties   | Determines table properties.                                                                                                |
| Target            | Defines a frame or window that a URL will be displayed in.                                                                  |
| Title             | Defines the text displayed when the mouse pointer moves over the shape.                                                     |
| Type              | Defines a reference to the ID of a ShapeType element.                                                                       |
| UserDrawn         | Determines whether the user has added the shape to a master slide.                                                          |
| UserHidden        | Determines whether a script anchor is hidden.                                                                               |
| Visibility        | Determines whether a shape is displayed.                                                                                    |
| WrapCoords        | Defines the bounding polygon that surrounds a shape.                                                                        |

1

| Element  | Description                   |
|----------|-------------------------------|
| Formulas | Defines formulas for a shape. |

| Element | Description                  |
|---------|------------------------------|
| Handles | Defines handles for a shape. |
| Locks   | Defines a lock for a shape.  |

## 1    6.3    Group Element

- 2    The Group element is used to collect multiple objects so they can be positioned and transformed as a single  
 3    unit. Objects that reference their parent container's coordinate space become relative to the group's local  
 4    space when inserted into a group. Using groups supports creation of complex shapes, composed of many sub-  
 5    shapes, that can be treated as a single entity.
- 6    Group supports a subset of the Shape element's attributes.

| Attribute         |                   |                  |                 |
|-------------------|-------------------|------------------|-----------------|
| AllowInCell       | Class             | HRPct            | Style           |
| AllowOverlap      | CoordOrig         | HRStd            | TableLimits     |
| Alt               | CoordSize         | HRWidth          | TableProperties |
| BorderBottomColor | DoubleClickNotify | ID               | Target          |
| BorderLeftColor   | HR                | OnEd             | Title           |
| BorderRightColor  | HRAlign           | OnMouseOver      | UserDrawn       |
| BorderTopColor    | HRef              | Print            | UserHidden      |
| Bullet            | HRHeight          | ReGroupID        | WrapCoords      |
| Button            | HRNoShade         | RelativePosition |                 |

7

- 8    The following elements are valid inside a Group:

| Element |       |           |           |
|---------|-------|-----------|-----------|
| Arc     | Image | Polyline  | Shape     |
| Curve   | Line  | Rect      | ShapeType |
| Group   | Oval  | RoundRect |           |

## 9    6.4    ShapeType Element

- 10   The ShapeType element defines a definition, or template, for a shape. Such a template is "instantiated" by  
   11   creating a Shape element that references the ShapeType. The shape can override any value specified by its  
   12   ShapeType, or define attributes and elements the ShapeType does not provide. A ShapeType may not  
   13   reference another ShapeType.
- 14   The attributes and elements a ShapeType uses are identical to those of the Shape element, with these  
   15   exceptions.
- 16   ShapeType may not use the Type element.

- 1 CSS positioning attributes are ignored and not passed to individual Shape instances.
- 2 Visibility is always hidden.
- 3 A VML authoring agent may make the ShapeType visible, in which case the CSS positioning attributes are meaningful.
- 4 The ShapeType element is used to define a shape once and reference it multiple times throughout a document. One of the most useful attributes or elements a ShapeType defines is a complex Path. Since Path coordinates are defined in a relative coordinate space that scales with a shape's height and width, this is very flexible for defining a shape outline that can be custom scaled and formatted for a given use.

## 9     **6.5     VML Usage in the Office Open XML Format**

### 10    **6.5.1   OfficeArt Shapes**

- 11 WordprocessingML takes advantage of the template-based shape definition VML provides. This example shows how the two shapes in the screenshot below are created.
- 12



- 13
- 14 The star is first defined using a ShapeType.

```
<v:shapetype id="_x0000_t12" coordsize="21600,21600" o:spt="12"
    path="m10800,18280,8259,,8259r6720,514614200,21600r6600,
    -5019117400,21600,14880,13405,21600,8259r-8280,xe">
    <v:stroke joinstyle="miter" />
    <v:path gradientshapeok="t" o:connecttype="custom"
    ...
    o:connectlocs="10800,0;0,8259;4200,21600;17400,21600;21600,8259"
    textboxrect="6720,8259,14880,15628" />
</v:shapetype>
```

- 15
- 16 The first star is created by referencing the ShapeType via the Type attribute. It sets its own positioning and scaling.
- 17

```
<v:shape id="_x0000_s1026" type="#_x0000_t12"
    style="position:absolute;margin-left:33pt;margin-top:25.5pt;
    width:47.25pt;height:47.25pt;z-index:251656704" />
```

- 1 The second star is created by referencing the ShapeType and providing its own positioning, scaling and
- 2 formatting.

```
<v:shape id="_x0000_s1027" type="#_x0000_t12"
style="position:absolute;margin-left:145.5pt;margin-top:25.5pt;
width:47.25pt;height:47.25pt;z-index:251657728"
fillcolor="#4f81bd [3204]" strokecolor="#f2f2f2 [3041]"
strokeweight="3pt">
<v:shadow on="t" type="perspective" color="#27405e [1604]"
opacity=".5" offset="1pt" offset2="-1pt" />
</v:shape>
```

- 3 The example contains only the two star shapes. What follows is the entire document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:document ">
<w:body>
<w:p>
<w:r w:rsidR="00496863">
<w:rPr>
<w:noProof />
</w:rPr>
<w:pict>
<v:shapetype id="_x0000_t12" coordsize="21600,21600"
o:spt="12"
path="m10800,18280,8259,,8259r6720,514614200,21600r6600,
-5019l17400,21600,14880,13405,21600,8259r-8280,xe">
<v:stroke joinstyle="miter" />
<v:path gradientshapeok="t" o:connecttype="custom"
o:connectlocs="10800,0;0,8259;4200,21600;
17400,21600;21600,8259"
textboxrect="6720,8259,14880,15628" />
</v:shapetype>
<v:shape id="_x0000_s1026" type="#_x0000_t12"
style="position:absolute;margin-left:33pt;
margin-top:25.5pt;
width:47.25pt;height:47.25pt;z-index:251656704" />
</w:pict>
</w:r>
<w:r w:rsidR="00496863">
<w:rPr>
<w:noProof />
</w:rPr>
<w:pict>
```

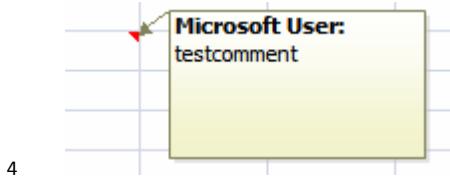
```

<v:shape id="_x0000_s1027" type="#_x0000_t12"
    style="position:absolute;margin-left:145.5pt;
    margin-top:25.5pt;width:47.25pt;height:47.25pt;
    z-index:251657728" fillcolor="#4f81bd [3204]"
    strokecolor="#f2f2f2 [3041]" strokeweight="3pt">
    <v:shadow on="t" type="perspective"
        color="#27405e [1604]"
        opacity=".5" offset="1pt" offset2="-1pt" />
</v:shape>
</w:pict>
</w:r>
</w:p>
<w:sectPr w:rsidR="00953D70" w:rsidSect="00667294">
    <w:pgSz w:w="12240" w:h="15840" />
    <w:pgMar w:top="1440" w:right="1440" w:bottom="1440"
        w:left="1440"
        w:header="720" w:footer="720" w:gutter="0" />
    <w:cols w:space="720" />
    <w:docGrid w:linePitch="360" />
</w:sectPr>
</w:body>
</w:document>

```

## 1 6.5.2 SpreadsheetML Comments

- 2 The visible box shown for comments attached to cells is persisted using VML. The comment contents are stored separately as part of SpreadsheetML.



- 5 The package item xl/worksheets/sheet1.xml contains the following reference:

```
<legacyDrawing r:id="rId1" />
```

- 6 This is a relationship defined in xl/worksheets/\_rels/sheet1.xml.rels:

```
<Relationship Id="rId1" Type=".../legacyDrawing"
    Target="../drawings/legacyDrawing1.vml" />
```

- 7 The package item xl/drawings/legacyDrawing1.vml defines the yellow gradient rectangle. Again, note that the basic rectangle is defined using a ShapeType. This is reused if multiple comments exist.

```

<xml ">
  <o:shapelayout v:ext="edit">
    <o:idmap v:ext="edit" data="1" />
  </o:shapelayout>
  <v:shapetype id="_x0000_t202" coordsize="21600,21600" o:spt="202"
    path="m,l,21600r21600,l21600,xe">
    <v:stroke joinstyle="miter" />
    <v:path gradientshapeok="t" o:connecttype="rect" />
  </v:shapetype>
  <v:shape id="_x0000_s1027" type="#_x0000_t202"
    style="position:absolute;
margin-left:107.25pt;margin-top:52.5pt;width:96pt;height:55.5pt;
z-index:1" fillcolor="#f2f3cb" strokecolor="#81835a"
o:insetmode="auto">
    <v:fill color2="#fefefb" type="gradient">
      <o:fill v:ext="view" type="gradientUnscaled" />
    </v:fill>
    <v:shadow on="t" color="silver" opacity=".5" obscured="t" />
    <v:path o:connecttype="none" />
    <v: textbox style="mso-direction-alt:auto">
      <div style="text-align:left" />
    </v: textbox>
    <x:ClientData ObjectType="Note">
      <x:MoveWithCells />
      <x:SizeWithCells />
      <x:Anchor>2, 15, 3, 10, 4, 15, 7, 4</x:Anchor>
      <x:AutoFill>False</x:AutoFill>
      <x:Row>4</x:Row>
      <x:Column>1</x:Column>
      <x:Visible />
    </x:ClientData>
  </v:shape>
</xml>

```

### <sup>1</sup> 6.5.3 WordprocessingML Text Box

- <sup>2</sup> WordprocessingML stores all textbox geometry using VML. This example shows how a simple text box is stored.



*Text box*

- 1 All the VML is embedded directly in the word/document.xml file as it is intermingled with other XML. VML is used to define the graphic content. Within the VML textbox tag, additional information about the text box text is added. The following is the section of the document.xml that defines the text box.

```

<w:r w:rsidR="00735D93">
  <w:rPr>
    <w:noProof />
  </w:rPr>
  <w:pict>
    <v:roundrect id="_x0000_s1027" style="position:absolute;
      margin-left:193.2pt;margin-top:-18pt;width:385.75pt;
      height:36.5pt;z-index:251660288;mso-width-percent:900;
      mso-position-horizontal-relative:page;
      mso-position-vertical-relative:margin;mso-width-percent:900;
      mso-width-relative:margin" arcsize="2543f" o:allowincell="f"
      stroked="f">
      <v:shadow on="t" type="perspective" color="#4f81bd [3204]"
        origin="- .5,- .5" offset="-3pt,-3pt" offset2="6pt,6pt"
        matrix=".75,,, .75" />
      <v:textbox style="mso-next-textbox:#_x0000_s1027;
        mso-fit-shape-to-text:t" inset=",,36pt,18pt">
        <w:txbxContent>
          <w:p>
            <w:pPr>
              <w:rPr>
                <w:i />
                <w:color w:val="7F7F7F" w:themeColor="background1"
                  w:themeShade="7F" />
              </w:rPr>
            </w:pPr>
            <w:r w:rsidR="00CA19B3">
              <w:rPr>
                <w:i />
                <w:color w:val="7F7F7F" w:themeColor="background1"
                  w:themeShade="7F" />
              </w:rPr>
              <w:t>Text box</w:t>
            </w:r>
          </w:p>
        </w:txbxContent>
      </v:textbox>
      <w10:wrap type="square" anchorx="page" anchory="margin" />
    </v:roundrect>
  
```

```
</w:pict>  
</w:r>
```

- 1 This general format is used for any type of textbox, such as those added automatically when a cover page is added to a document.
- 2
- 3 **End of informative text.**

# 1    7. Introduction to Shared MLs

2    This clause is informative.

## 3    7.1    Math

4    In this subclause, every mathematical expression is called an *equation*, even if the expression is merely a string  
 5    of variables or a single object such as a fraction. In XML, an equation is called *OMath*. A Math Paragraph is a  
 6    group of one or more equations separated by soft carriage returns; that is, they are separate equations that  
 7    comprise a single paragraph. A Math Paragraph carries its own justification that can be separate from the  
 8    justification of the paragraph that contains it. Different equations within a Math Paragraph cannot have  
 9    different types of justification.

10   Equations can be Display (the only text on the line) or Inline (on a line with text outside of the equation). The  
 11   Display vs. Inline state is not specified by this standard; instead, a text processor determines how to format  
 12   Display and Inline equations. Display and Inline equations innately carry different formatting characteristics;  
 13   Inline equations consume less vertical space so as not to disrupt line spacing with adjacent lines. This means,  
 14   for example, reducing the size of fractions and n-ary objects that grow.

15   The following subclauses introduce each of the objects (also called *functions*) that comprise the majority of the  
 16   Equations schema. As a text processor and not a calculation engine, when converting equations into XML  
 17   representation, more attention is given to the layout and appearance than to the mathematical meaning of  
 18   the expressions. That is,  $\overrightarrow{abc}$  and  $\widehat{abc}$  are represented with the same object, although they carry  
 19   different mathematical meanings, because both consist of text paired with a stretching character. Similarly,  $\frac{n}{k}$   
 20   and  $\frac{n^m}{k}$  are represented as the same object. Though mathematically they have different meaning, their layout  
 21   is similar.

22   Although the functionality described in this clause is purely about the appearance of equations, other markup  
 23   defined in this Office Open XML Standard provides independent functionality enabling calculation of  
 24   mathematical expressions. Formulas in SpreadsheetML (§3.15.1) and Fields in WordprocessingML (§2.17.1)  
 25   are two examples.

### 26    7.1.1    Accent Object

27   Consider the following letters having diacritical marks:

28    $\dot{a}$   $\ddot{a}$   $\tilde{a}$   $\hat{a}$   $\vec{a}$

1 The accent object is used to represent any baseline text having a combining diacritical mark placed above the  
 2 base. The accent has only one child, the base element. The accent mark itself is stored as a property. In the  
 3 examples above, the only difference in the XML representations is the character.

4 

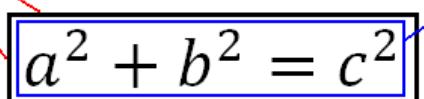
## 5 7.1.2 Bar Object

6 The bar object consists of baseline text with a bar drawn above or below the base. The bar has only one child,  
 7 the base element. The location of the bar is stored as a property. For example:

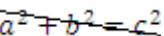
8 

## 9 7.1.3 Border Box Object

10 The Border Box object consists of math text—often a formula the author wishes to call out or give special  
 11 attention—surrounded by a border. Any combination of the edges of the border can be hidden. For example:

12 

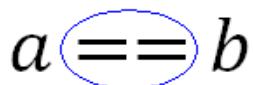
13 The Border Box can also be used to "cross out" text with a horizontal, vertical, or diagonal (from top-left to  
 14 bottom-right or from top-right to bottom-left) strikethrough, as follows:

15 

## 16 7.1.4 Box Object

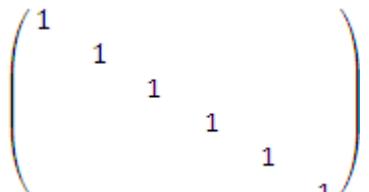
17 The Box object is used to group components of an equation, to apply a single property to everything in the  
 18 box. The Box serves a number of distinct purposes, including grouping characters to form a single operator (an  
 19 operator emulator), and thereby inheriting the alignment and manual break properties of operators; grouping  
 20 a differential such as  $\frac{dx}{dt}$ ; preventing line breaks from occurring within; and allowing text inside to be reduced  
 21 in script level.

22 An example of a Box serving as an operator emulator is:

23 

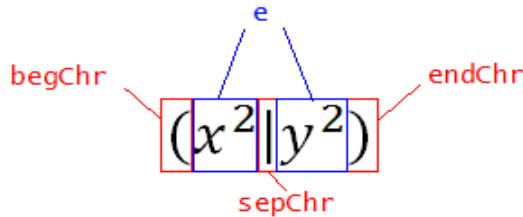
### 7.1.5 Delimiters

- 1 Delimiters consist of opening and closing delimiting characters (such as parentheses, braces, brackets, and vertical bars), and an element contained inside. If two or more elements are contained within delimiters, separating characters are used.
- 5 Delimiters can grow to the height of the object they contain. For example, parentheses could grow quite tall to



- 6 enclose this multi-row matrix: Or, at the user's discretion, they can maintain  
7 their height regardless of the content inside, as in

- 8 Delimiters have a single type of child, the base argument, which can be used multiple times in the object to  
9 signify that a separator character is to be used. For example:



- 10  
11 If the separator character is not specified in XML, the vertical bar is used.

### 7.1.6 Equation Array Object

- 12 The Equation Array object consists of one or more equations grouped as an object. Within the equation array,  
14 multiple components can be aligned to each other. Examples of equation arrays are:

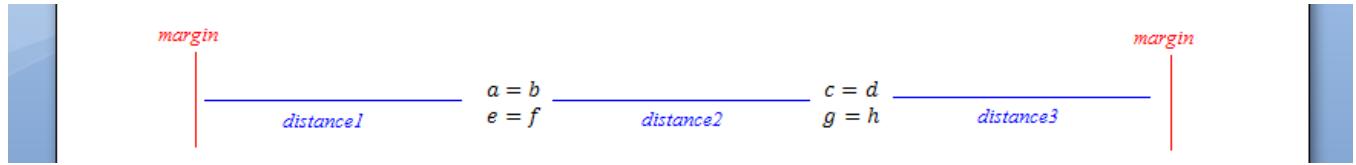
$$\begin{aligned} x - y + z &= 10 \\ 3x + y + 2z &= 34 \\ -5x + 2y - z &= -14 \end{aligned} \quad \text{and} \quad f(x) = \begin{cases} -x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

- 15  
16 Equation arrays can have "maximum distribution" such that they occupy the entire width of the column that  
17 contains them, as in:



18

- 1 Or, they can have "object distribution" such that there is even spacing between the margin and text (distance1  
 2 = distance2 = distance 3):



### 7.1.7 Fraction Object

- 5 The Fraction object consists of a numerator and denominator separated by a fraction bar. The Fraction object  
 6 is used to classify the different styles of fractions. It is also used to classify the stack object, which places one  
 7 element above another, with no fraction bar. The four types of fractions are shown below:

Stacked Fraction:  $\frac{a}{b}$

Skewed Fraction:  $a/b$

Linear Fraction:  $a/b$

Stack (noBar) Fraction:  $n/k$

8

### 7.1.8 Function Apply Object

- 10 The Function Apply object consists of a function name (or object) applied to a base. The function name, by  
 11 default, does not use math italics. The Function Apply object consists of a function name (a string or object)  
 12 and a base element acted upon, as in:

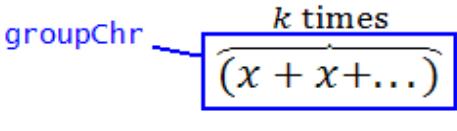
fName  
  
 $\lim_{n \rightarrow \infty} x_n$

13

- 14 The user can modify the text in a function name, or can add strings to be recognized automatically by the text  
 15 processor as function names.

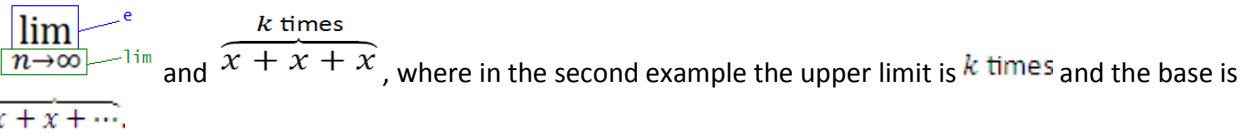
### 7.1.9 Group Character Object

- 17 The Group Character object consists of a character drawn above or below text, often with the purpose of  
 18 visually grouping items. In the following example, the text above the overbrace is not part of the group  
 19 character object; it is included only to demonstrate a real-world example of the object in use:

1      

## 2    7.1.10    Upper and Lower Limits

3    Upper Limits and Lower Limits are treated as separate (but similar) objects in the XML representation. Both  
4    consist of text on the baseline and reduced-size text immediately above or below it. Examples include:

5      

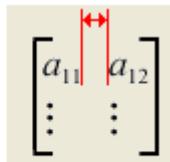
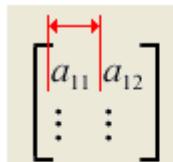
6      and  $\overbrace{x + x + x}^{k \text{ times}}$ , where in the second example the upper limit is *k times* and the base is  $x + x + \dots$ .

## 7    7.1.11    Matrix Object

8    The Matrix object consists of one or more elements laid out in one or more rows and one or more columns

9    (delimiters not included). Examples include  $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$  and  $\begin{bmatrix} 1 & \\ & 1 \end{bmatrix}$ .

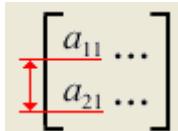
10   The entire matrix can be aligned, with respect to the surrounding text, at the center, with the top row, or with  
11   the bottom row. This property is defined as baseJc. Spacing between columns can be defined using cGp,  
12   cGpRule, and cSp. Column Gap refers to the space between the end of one column and the start of the next;  
13   column spacing refers to the space between two corresponding edges of adjacent columns.



14    *Column Spacing*

14    *Column Gap*

15   Row spacing can also be defined using rSp and rSpRule. Row spacing is defined as the distance between  
16   baselines on adjacent matrix rows:



18

19   Finally, a matrix can have hidden placeholders (hidePlc). The identity matrix above has hidden placeholders,

$$\begin{array}{ccc} 1 & \square & \square \\ \square & 1 & \square \\ \square & \square & 1 \end{array}$$

20   while the following matrix has placeholders showing:

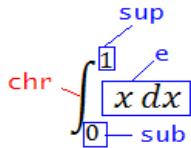
### 1 7.1.12 N-ary Object

2 The N-ary object consists of an n-ary object, a base (or operand), and optional upper and lower limits, as in:

$$\int_0^1 x \, dx \quad \sum_k \binom{n}{k} \quad \prod_{k=1}^n A_k \quad \bigcup_{n=1}^m (X_n \cap Y_n)$$

3

4 The components of an n-ary object are as follows:



5

6 Other properties of the n-ary object are:

- 7 • grow: specifies whether the n-ary object grows to the height of its operand, or stays a fixed height
- 8 • limLoc: specifies the placement of n-ary limits: either to the right of the n-ary operator (the subSup position) or centered above and below (the undOvr position).
- 9 • supHide: specifies that the upper limit is hidden and no placeholder shows
- 10 • subHide: specifies that the lower limit is hidden and no placeholder shows
- 11

### 12 7.1.13 Phantom Object

13 The Phantom object allows extra spacing, horizontal, vertical, or both, to be added or suppressed during layout  
14 for enhanced appearance.

15 In the following example, the two radicals are unbalanced:  $\sqrt{\frac{a}{b}} = \sqrt{x}$ . For enhanced typography, the radical  
16 bars and bottom points should line up. To accomplish this, the user should adjust the height of the second  
17 radical, to make it the height of the fraction. However, no extra padding should be added to the width. The

18 user can accomplish this by inserting a phantom of the fraction under the second radical, as in:  $\sqrt{\frac{a}{b}} = \sqrt{x}$ . In  
19 this case, the radicals line up, and the phantom fraction acts as ghost text that adds vertical space but no width  
20 (zeroWid). The phantom can also be used to add horizontal space, alone or in conjunction with vertical space.

21 Phantoms are not always invisible. The "smash" is a type of phantom in which the content remains visible.  
22 However, part or all of the smash can be ignored during layout of text around it. For example, examine the  
23 following two radicals:

$$\sqrt{x \, dx}$$

ascent of "d" causes radical to be higher.

$$\sqrt{x dx}$$

when the ascent of "d" is smashed, the gap is narrowed.

24

- 1 A discerning typographer might desire less vertical spacing between the tip of the "d" and the radical bar in the  
 2 first example. By placing the differential in a smash and assigning it zero height (zeroAsc), the spacing is  
 3 reduced.
- 4 Note that in this same example, when the differential term is placed inside a phantom, the spacing between  
 5 the first and second characters changes. Again, the discerning typographer wishes that despite the presence of  
 6 the phantom, differential spacing is retained. By assigning the phantom transparency for spacing (transp),  
 7 proper spacing is preserved.
- 8 Finally, zeroDesc phantom allows the descent of the phantom base to be ignored during layout. The following  
 9 example illustrates the usage of zeroDesc:

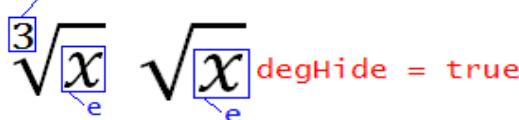
height of radical  $\sqrt{y} = \sqrt{y}$  height of radical when the descent  
 of y is suppressed

10

- 11 Each of the phantom properties can be applied whether the phantom is visible or hidden (the show property).

### 7.1.14 Radical Object

- 13 The Radical object consists of a radical, a base e, and an optional degree. When the degree is not shown and a  
 14 placeholder character is not to appear, the property degHide is used.

deg  
  
 15

### 7.1.15 Scripts (Superscript, Subscript, SubSuperscript, PreSubSuperscript)

- 17 There are four distinct but related objects that consist of a base and a smaller “script” term either raised or  
 18 lowered, on the left or right of the base. These are the Subscript, Superscript, SubSuperscript, and  
 19 PreSubSuperscript:

$x_n$   $x^n$   $x_m^n$   $_m^nx$

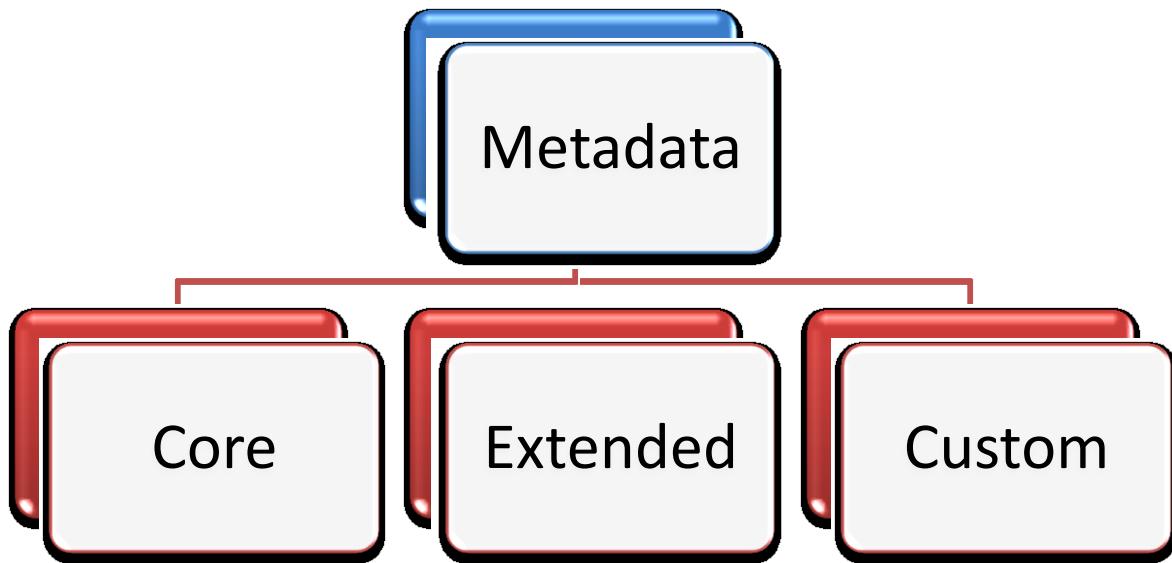
20

- 21 The SubSuperscript has the option of aligning scripts (alnScr), as in:

22 

## 1 7.2 Metadata

- 2 Office Open XML document metadata consists of 43 well-defined properties and user-defined custom  
 3 properties. Metadata properties are divided into three categories: Core, Extended, and Custom.



4

- 5 Each metadata category is represented by a document part with a corresponding relationship type, content  
 6 type, and schema. Each metadata property is associated with exactly one metadata part.  
 7 The following table lists all well-defined metadata properties:

Property	Category
category	Core
contentStatus	Core
contentType	Core
Created	Core
Creator	Core
description	Core
identifier	Core
keywords	Core
language	Core
lastModifiedBy	Core
lastPrinted	Core
modified	Core

Property	Category
revision	Core
subject	Core
title	Core
version	Core
Application	Extended
AppVersion	Extended
Characters	Extended
CharactersWithSpaces	Extended
Company	Extended
DigSig	Extended
DocSecurity	Extended
HeadingPairs	Extended
HiddenSlides	Extended
HLinks	Extended
HyperlinkBase	Extended
HyperlinksChanged	Extended
Lines	Extended
LinksUpToDate	Extended
Manager	Extended
MMClips	Extended
Notes	Extended
Pages	Extended
Paragraphs	Extended
PresentationFormat	Extended
ScaleCrop	Extended
SharedDoc	Extended
Slides	Extended
Template	Extended
TitlesOfParts	Extended
TotalTime	Extended
Words	Extended

### <sup>1</sup> 7.2.1      Metadata Properties

- <sup>2</sup> Metadata properties are represented as XML elements with associated name and type. There are two types of properties: simple and complex. Simple properties are singular XML elements whose type and value is defined

1 by the type and value of that XML element. Complex properties contain nested variant type XML elements  
 2 that define the type and value of complex data such as arrays and vectors. Metadata properties are non-  
 3 repeatable and must be defined within their associated metadata part. All metadata properties may be empty  
 4 or omitted. If all properties of a metadata part are omitted, that part may be excluded from the document.

5 Simple property and custom complex property

```
6 <dc:creator>John Smith<dc:creator>
7 <property fmtid="{D5CDD505-2E9C-101B-9397-08002B2CF9AE}" pid="2" name="Editor">
8   <vt:lpwstr>John Smith</vt:lpwstr>
9 </property>
```

## 10 **7.2.2 Core Properties**

11 Core properties are a predefined set of metadata properties common to all packages, and are discussed in  
 12 detail in §11 of Part 2 of this Standard: "Open Packaging Conventions".

## 13 **7.2.3 Extended Properties**

14 Extended properties are a predefined set of metadata properties that are specifically applicable to Office Open  
 15 XML documents. Extended properties consist of 24 simple properties and 3 complex properties stored in the  
 16 part targeted by the relationship of type:  
 17 <http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties>.

## 18 **7.2.4 Custom Properties**

19 Custom properties allow users to extend pre-defined metadata properties with user-defined properties.  
 20 Custom properties are stored in the part targeted by the relationship of type:  
 21 <http://schemas.openxmlformats.org/officeDocument/2006/relationships/custom-properties>. Each  
 22 property is represented as a property XML element and uniquely identified through the name, fmtid, and pid  
 23 attributes. All custom properties are considered complex properties. The type and value of custom properties  
 24 are specified by its child variant type XML elements.

## 25 **7.2.5 Variant Types**

26 Office Open XML defines 35 XML elements representing commonly-used variant types to enable the  
 27 representation and round-tripping of complex data. Variant type XML elements are used as child elements of  
 28 complex metadata properties to define the type and value.

## 29 **7.3 Custom XML Data**

30 Within an Office Open XML document, it is sometimes desirable or necessary to store custom XML data (that  
 31 is, data in a format not defined by this Office Open XML specification) within the contents of the package. To  
 32 accommodate this need, Office Open XML allows the storage of any arbitrary XML within a package as the  
 33 target of the <http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXml>  
 34 relationship (valid source parts are listed within Part 1 of the Standard).

35 The following examples illustrate potential uses of this mechanism:

- A document which collects and displays information from a backend data source might want to store the original form of that backend data, so it can be manipulated and uploaded to the original data source at a later date.
- A document author may wish to store additional metadata in an XML format not defined by this Office Open XML specification's existing metadata schemas.
- A document management system may wish to store data tracking the workflow status, retention policies, and so on for this document along with the document.

Once present within a package, this custom XML data shall be maintained as a distinct part separate from the contents of the document, spreadsheet, or presentation. If there are multiple distinct streams of custom XML data, each is maintained as a separate part within the package, so that each can be manipulated independently of all others.

Each Custom XML Data part may also have an implicit relationship to a Custom XML Data Properties part which stores:

- The target namespace of all XML schemas which shall be used to validate the content of the custom XML data part.
- A GUID which shall remain constant over the lifetime of this part (and can therefore be used to uniquely identify it).

## 7.4 Bibliography

Office Open XML offers functionality to store bibliography entries to permit automatic formatting of citations and bibliographies in the document according to a set of documentation rules defined in an XSLT.

### 7.4.1 Types of Sources

The Office Open XML formats support a collection of predefined source types for bibliography entries based on the categories most commonly used in various citation and bibliography style guidelines . The set of predefined source types can be extended as needed. The recommended approach for extending this set is to use the `Misc` type, and then leverage the methods described in Part 5 of this standard for extending the format with new attributes or elements. The following types of sources are predefined:

- `Book` (`Book`)
- `BookSection` (`Book Section`)
- `JournalArticle` (`Journal Article`)
- `MagOrNewsArticle` (`Magazine or Newspaper Article`)
- `ConferenceProceedings` (`Conference Proceedings`)
- `Report` (`Report`)
- `SoundRecording` (`Sound Recording`)
- `Performance` (`Performance`)
- `Art` (`Art`)
- `DocumentFromInternetSite` (`Document from Internet Site`)

- 1     • InternetSite (Internet Site)
- 2     • Film (Film)
- 3     • Interview (Interview)
- 4     • Patent (Patent)
- 5     • ElectronicSource (Electronic Source)
- 6     • Case (Case)
- 7     • Misc (Miscellaneous)

## 8     **7.4.2     Child Elements**

9     Each Source element has a number of elements as children, each of which represents a different piece of data  
10    for the bibliography entries. For example, a book might have an author, title, publisher, year, and city. Most  
11    are self-explanatory, but this document will pay special attention to some of the more complex children.

12    The child elements are:

- 13     • AbbreviatedCaseNumber
- 14     • AlbumTitle
- 15     • Author
- 16     • BookTitle
- 17     • Broadcaster
- 18     • BroadcastTitle
- 19     • CaseNumber
- 20     • ChapterNumber
- 21     • City
- 22     • Comments
- 23     • ConferenceName
- 24     • Country
- 25     • CountryRegion
- 26     • Court
- 27     • Day
- 28     • DayAccessed
- 29     • Department
- 30     • Distributor
- 31     • Edition
- 32     • Guid
- 33     • Institution
- 34     • InternetSiteTitle
- 35     • Issue
- 36     • JournalName
- 37     • LCID
- 38     • Medium

- 1     • Month
- 2     • MonthAccessed
- 3     • NumberVolumes
- 4     • Pages
- 5     • PatentNumber
- 6     • PeriodicalTitle
- 7     • PlacePublished
- 8     • ProductionCompany
- 9     • PublicationTitle
- 10    • Publisher
- 11    • RecordingNumber
- 12    • RefOrder
- 13    • Reporter
- 14    • SourceType
- 15    • ShortTitle
- 16    • StandardNumber
- 17    • StateProvince
- 18    • Station
- 19    • Tag
- 20    • Theater
- 21    • ThesisType
- 22    • Title
- 23    • Type
- 24    • URL
- 25    • Version
- 26    • Volume
- 27    • Year
- 28    • YearAccessed

29 An example of the XML defining a source of type Book with the title Office Open XML formats, and two  
 30 Authors named Jones, Brian and Davis, Tristan is:

```

31 <b:Source>
32   <b:Tag>Las07</b:Tag>
33   <b:SourceType>Book</b:SourceType>
34   <b:Author>
35     <b:Author>
36       <b:NameList>
37         <b:Person>
38           <b>Last>Jones</b>Last>
39           <b>First>Brian</b>First>
40         </b:Person>

```

```

1      <b:Person>
2          <b:Last>Davis</b:Last>
3          <b:First>Tristan</b:First>
4      </b:Person>
5      </b:NameList>
6  </b:Author>
7 </b:Author>
8 <b>Title>Office Open XML formats</b>Title>
9 <b:Year>2007</b:Year>
10 <b:City>Trondheim</b:City>
11 <b>Publisher>Publisher</b>Publisher>
12 <b>Comments>Comments</b>Comments>
13 <b>RefOrder>1</b>RefOrder>
14 <b>Guid>{DCC25FA1-67CC-4013-B56A-2D42CED7FF0C}</b>Guid>
15 <b>LCID>0</b>LCID>
16 </b:Source>

```

### 7.4.3 Author

There are two elements with the same name: Author. The first Author element is a container for the set of contributors attributed to the current source. The second Author element is a child of the first and is used to represent a single contributor. The valid set of contributors is defined as:

- Artist
- Author
- BookAuthor
- Compiler
- Composer
- Conductor
- Counsel
- Director
- Editor
- Interviewee
- Interviewer
- Inventor
- Performer
- ProducerName
- Translator
- Writer

For example, a bibliographic source with an author (Davis, Tristan), editor (Jaeschke, Rex), and translator (Jones, Brian) would be represented by a group of elements representing the three contributors and their specific roles inside an outer Author element, as in:

```

1   <b:Author>
2     <b:Author>
3       <b:NameList>
4         <b:Person>
5           <b>Last>Davis</b>Last>
6           <b>First>Tristan</b>First>
7         </b:Person>
8     </b:Author>
9     <b:Editor>
10    <b:NameList>
11      <b:Person>
12        <b>Last>Jaeschke</b>Last>
13        <b>First>Rex</b>First>
14      </b:Person>
15    </b:Editor>
16    <b:Translator>
17      <b:NameList>
18        <b:Person>
19          <b>Last>Jones</b>Last>
20          <b>First>Brian</b>First>
21        </b:Person>
22      </b:Translator>
23    </b:Author>

```

#### 24     **7.4.4     LCID, Guid, Tag, and RefOrder**

25     Four of the child elements for the Source element support important functionality for consuming applications  
26     that generate bibliographies using an externally defined stylesheet. The LCID element describes the language  
27     to be used when displaying the bibliography entry. This piece of data provides an instruction to the consuming  
28     application on the grammar of the citations and bibliography (including international name formats, date  
29     formats, and punctuation marks).

30     The Guid and Tag elements can be leveraged if an application wishes to uniquely identify the bibliography  
31     entry described in the Source element. For example, when a Source element is brought into a document and  
32     the Tag value for that Source element matches that of another Source element already in the document, the  
33     existing Source elements values could be overwritten with the new Source element. Two Source elements  
34     with the same Tag element value cannot exist in the same document. GUIDs can then be used in conjunction  
35     with Tags to indicate whether a Source element has been edited. When a Source element from one document  
36     has been edited, an application may decide to apply the edits to matching Source elements in other  
37     documents.

38     The RefOrder element for a source indicates the position, in numeric sequence, for the first reference to the  
39     source within the document text. This information is used in bibliography styles that sort sources by order in  
40     the document rather than alphabetical order.

- 1    **End of informative text.**

# 1    8. Miscellaneous Topics

2    This clause is informative.

## 3    8.1 Additional Characteristics

4    Office Open XML provides a way for a producer to provide information to consumers regarding how the data  
 5    was created and how it should be interpreted. This information is provided by one or more *additional  
 6    characteristics*.

7    A producing application has the option of writing out as many or as few additional characteristics as desired.

8    A consuming application has the option of acting on the additional characteristics or ignoring them

9    The additional characteristics are stored in a separate XML part, as follows:

```
10   <additionalCharacteristics>
11     <characteristic name='name of characteristic'
12       relation='well defined set of relation types'
13       val='string' vocabulary='uri' />
14   </additionalCharacteristics>
```

15   For example, consider the case in which numColumns is the characteristic name to specify the maximum  
 16   number of columns supported by the producing SpreadsheetML application, so that the consuming application  
 17   can understand how to distinguish cell references and variables unambiguously.

18   The relation attribute specifies the way in which the val attribute should be interpreted. The possible values  
 19   for relation are: lt | le | eq | gt | ge, which mean <, <=, =, >=, >, respectively, and relate to numerical  
 20   comparison for values and alphabetical comparison for ordering of strings. These relations permit expression  
 21   of the maximum value, the minimum value, the value, and so on.

22   The vocabulary attribute is a URI that provides a namespace for the specific characteristic names provided as  
 23   values of the name attribute. This allows for the creation of a vocabulary of characteristics of interest within a  
 24   given domain of application without concern for name conflict between vocabularies.

25   Another example use case would be for a producer to inform the consumer that the computations used to  
 26   calculate the stored numbers in the SpreadsheetML formulas have a particular numeric precision expressed by  
 27   the mantissa and exponent. A consumer can optionally check those values to determine whether, for example,  
 28   the values should be recalculated. The XML to represent these characteristics might look like the following:

```

1   <additionalCharacteristics>
2     <characteristic name='precisionMantissa'
3       relation='gt'
4       val='-9007199254740992' />
5     <characteristic name='precisionMantissa'
6       relation='lt' val='9007199254740992' />
7     <characteristic name='precisionExponent'
8       relation='ge' val='-1075' />
9     <characteristic name='precisionExponent'
10    relation='le' val='970' />
11  </additionalCharacteristics>
```

## 12 8.2 Embeddings

13 Office Open XML provides facilities allowing the embedding of any object within a document. For example, a  
 14 WordprocessingML document might include data as an embedded SpreadsheetML document rather than a  
 15 native WordprocessingML table, in order to allow that data to be edited and recalculated by a SpreadsheetML  
 16 calculation engine, rather than having it stored as a static table of data.

17 Office Open XML provides for two classes of embedded objects:

- 18 • *Embedded Packages* - An embedded Office Open XML document embedded within another Office  
 19 Open XML document, with both documents stored in the format defined by this Office Open XML  
 20 specification. For example, a PresentationML document embedded within a SpreadsheetML document  
 21 results in an embedded package.
- 22 • *Embedded Objects* - Any other embedded object data. The data stored in the object shall be identified  
 23 by a unique string, referred to as its *ProgID*. This string shall be used to determine both the type of  
 24 data and the application (if any) that shall be used to load and edit the embedded object data.

25 Office Open XML also allows an image to be optionally associated with the embedded object data, for use  
 26 when the embedded object application and data itself is not used by the consuming application (e.g. when the  
 27 object cannot be loaded – the object is from an unknown source; the object is known, but the application has  
 28 chosen not to load it for performance reasons, and so on).

### 29 8.2.1 Embedded Packages

30 Whenever an Office Open XML document is stored as an embedded object, the embedding shall be referred to  
 31 as an embedded package. Embedded packages shall be the target of the Embedded Package relationship  
 32 defined in Part 1: <http://schemas.openxmlformats.org/officeDocument/2006/relationships/package>. this  
 33 Office Open XML specification

### 34 8.2.2 Embedded Objects

35 For all other embeddings, the embedded object is stored in an arbitrary format defined by the application  
 36 whose data is being embedded. These generic embedded objects shall be the target of the Embedded Object  
 37 relationship: <http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject>. When

1 parsing the data stored in an embedded object part, an application shall use the associated ProgID (whose  
 2 location is described in the following subclauses) for the object.

### 3 8.2.3 Embeddings in a WordprocessingML Document

4 When an embedding is stored in a WordprocessingML document, it is stored in one of the following ways:

- 5     • In line with text - The object is displayed within the regular text stream (modifying line height and so  
 6       on to accommodate it).
- 7     • Floating – The object is positioned absolutely or relatively within the document and text flow is  
 8       modified as needed around it.

9 Each case permits the storage of both the object and the optional VML representation of the image that may  
 10 be used when the object data is not used by the hosting application as follows:

#### 11 8.2.3.1 Embeddings In Line With Text

12 When the embedding is present in line with text, it is stored as follows:

- 13     • The WordprocessingML object element specifies the presence of an embedded object in line with text.
- 14     • The child Office VML Drawing OLEObject element shall specify the details about the embedding itself,  
 15       including an explicit relationship to the appropriate Embedded Package or Embedded Object part.
- 16     • The child VML shape element shall specify the presence of the image which may be used to represent  
 17       the object.

18 For example, if we embed a SpreadsheetML worksheet in a WordprocessingML document, the following run  
 19 content would be present:

```
20   <w:r>
21     <w:object w:dxaOrig="7247" w:dyaOrig="2920">
22       <v:shape id="_x0000_i1026" type="#_x0000_t75"
23         style="width:362.25pt;height:146.25pt" o:ole="">
24           <v:imagedata r:id="rId6" o:title="" />
25         </v:shape>
26         <o:OLEObject Type="Embed" ProgID="Excel.Sheet.8"
27           ShapeID="_x0000_i1026" DrawAspect="Content" ObjectID="_1218026609"
28             r:id="rId7" />
29       </w:object>
30     </w:r>
```

31 If we examine this markup, it can be seen that:

- 32     • We have an inline embedded object, as defined by the object element.
- 33     • The OLEObject element specifies that that object is stored as an Embed, and that its ProgID is  
 34       Excel.Sheet.8 (the ProgID code for Microsoft Excel worksheets); it also specifies that the associated

- 1       image (when the object data cannot be used) is stored in the VML shape with a shape ID of  
 2       \_x0000\_i1026.
- 3       • The associated VML shape element with an id attribute value of \_x0000\_i1026 shall be used in  
 4       place of the object whenever it is not loaded - this shape is typically, but is not required to be, stored  
 5       in the same object element as the OLEObject element. This shape specifies its desired size and  
 6       provides an explicit relationship to the part that stores the image data.

### 7     8.2.3.2            Floating Embeddings

8     When the embedding is present as a floating object, it is stored as follows:

- 9       • The WordprocessingML pict element specifies the presence of a floating image in the document.  
 10      • The child Office VML Drawing OLEObject element shall specify the details about the embedding itself,  
       including an explicit relationship to the appropriate Embedded Package or Embedded Object part.  
 12      • The child VML shape element shall specify the presence of the image that may be used to represent  
 13       the object in place of loading the actual object data.

14     For example, if we embed a SpreadsheetML worksheet in a WordprocessingML document as a floating object,  
 15     the following run content would be present:

```
16      <w:r>
17          <w:pict>
18              <v:shapetype id="_x0000_t75" coordsize="21600,21600" o:spt="75"
19                  o:preferrelative="t" path="m@4@51@4@11@9@11@9@5xe" filled="f"
20                  stroked="f">
21                  ...
22          </v:shapetype>
23          <v:shape id="_x0000_s1028" type="#_x0000_t75"
24              style="position:absolute;margin-left:354.75pt;margin-
25              top:642.75pt;width:182.3pt;height:73.6pt;z-index:251660288">
26              <v:imagedata r:id="rId4" o:title="" />
27          </v:shape>
28          <o:OLEObject Type="Embed" ProgID="Excel.Sheet.8"
29              ShapeID="_x0000_s1028" DrawAspect="Content" ObjectID="_1218026611"
30              r:id="rId5" />
31      </w:pict>
32  </w:r>
```

33     If we examine this markup, it can be seen that:

- 34       • We have a floating image, as defined by the pict element.  
 35       • The OLEObject element specifies that that floating image is actually an embedding that is stored as an  
       Embed, and that its ProgID is Excel.Sheet.8 (the ProgID code for Microsoft Excel worksheets); it  
       also specifies that the associated image (when the object data cannot be used) is stored in the VML  
       shape with a shape ID of \_x0000\_s1028.

- 1     • The associated VML shape element with an id attribute value of \_x0000\_s1028 shall be used in  
 2       place of the object whenever it is not loaded - this shape is typically, but is not required to be, stored  
 3       in the same pict element as the corresponding OLEObject element. This shape specifies its desired  
 4       size and provides an explicit relationship to the part which stores the image data.

## 5    8.2.4      Embeddings in a SpreadsheetML Document

6    When an embedding is present in a SpreadsheetML document, it shall be stored as follows:

- 7     • In the worksheet, the oleObjects element shall store one or more oleObject child elements, one for  
 8       each embedding within the current worksheet. Each of those oleObject child elements shall also store:  
 9       an explicit relationship to the associated Embedded Package or Embedded Object part, the ProgID for  
 10      that embedded object, and (optionally) the last four digits of the shape ID for the associated VML  
 11      shape. The shape ID itself shall be of the form \_x0000\_s####, where # specifies a single Arabic  
 12      numeral, in order to be referenced as the alternate image for an embedding in a SpreadsheetML  
 13      document.  
 14     • In the worksheet, the sibling legacyDrawing element shall contain an explicit relationship to the VML  
 15      Drawing part that (optionally) contains the image data which may be used in place of loading the  
 16      actual object data.

17   For example, if we embed a Contoso Test object (an example for illustration) in a SpreadsheetML  
 18   document, the following markup would be stored in the appropriate Sheet part:

```
19 <s:worksheet>
20 ...
21 <s:legacyDrawing r:id="rId9" />
22 <s:oleObjects>
23   <s:oleObject progId="Contoso.Test.1" shapeId="1025" r:id="rId5"/>
24 </s:oleObjects>
25 </s:worksheet>
```

26 If we examine this markup, it can be seen that:

- 27     • The oleObject element specifies that we have one embedded object on the worksheet. Its attributes  
 28       specify that the object is of type Contoso.Test.1 and that the explicit relationship to the embedded  
 29       object is rId5.  
 30     • The sibling legacyDrawing element specifies that the Legacy Drawing part which contains the  
 31       associated legacy drawing data is contained at the target of the relationship with an ID of rId9.  
 32     • If we examine the VML Drawing part's contents, we'll see the shape which ends in 1025, which  
 33       contains the alternate image for the object:

```

1  <v:shape id="_x0000_s1025" type="#_x0000_t75" style='position:absolute;
2    margin-left:240.75pt;margin-top:105.75pt;width:334.5pt;height:253.5pt;
3    z-index:1' filled="t" fillcolor="window [65]" stroked="t"
4    strokecolor="windowText [64]" o:insetmode="auto">
5    <v:fill color2="window [65]"/>
6    <v:imagedata o:relid="rId1" o:title="" />
7    <x:ClientData ObjectType="Pict">
8      <x:SizeWithCells/>
9      <x:Anchor>5, 1, 7, 1, 11, 63, 23, 19</x:Anchor>
10     <x:CF>Pict</x:CF>
11   </x:ClientData>
12 </v:shape>
```

### 8.2.5 Embeddings in a PresentationML Document

When an embedding is present in a PresentationML document, it shall be stored as follows:

- In the slide, the embedding is stored as a graphic frame using the graphicFrame element in PresentationML.
- The graphicData element for the frame shall have the appropriate URI for its contents: <http://schemas.openxmlformats.org/presentationml/2006/ole>. Its child element shall be the PresentationML oleObj element, which stores an explicit relationship to the associated Embedded Package or Embedded Object part, the ProgID for that embedded object, and (optionally) the shape ID for the associated VML shape.
- The Slide part shall also have an implicit relationship to a VML Drawing part that (optionally) contains the image data to be used in place of loading the actual object data.

For example, if we embed the `Equation.3` object in a PresentationML document, the following markup would be stored in the shape tree of the appropriate Slide part:

```

26  <p:graphicFrame>
27  ...
28  <a:graphic>
29  C:\Documents and Settings\tristand\Local Settings\Temp\Temporary Directory 4 for
30  embeddedObject.pptx.zip\ppt\slides\slide1.xml  <a:graphicData
31    uri="http://schemas.openxmlformats.org/presentationml/2006/ole">
32    <p:oleObj spid="_x0000_s1026" name="Equation" r:id="rId3">
33      imgW="320" imgH="272" progId="Equation.3">
34      <p:embed />
35    </p:oleObj>
36    </a:graphicData>
37  </a:graphic>
38 </p:graphicFrame>
```

If we examine this markup, it can be seen that:

- 1     • The uri attribute on the graphicData element is  
 2        http://schemas.openxmlformats.org/presentationml/2006/ole, which dictates that this is an  
 3        embedded object
- 4     • It contains an oleObj element that specifies that the properties of the embedded object. Its attributes  
 5        specify that the object is of type **Equation.3** and that the explicit relationship to the embedded  
 6        object is rId3.
- 7     • The slide may also contain an implicit relationship to a Legacy Drawing part. If we examine the legacy  
 8        drawing part's contents, the shape with ID \_x0000\_s1026 (if present) defines the alternate image:

```

9   <v:shape id="_x0000_s1026" type="#_x0000_t75" style='position:absolute;
10  left:282pt;top:24pt;width:152pt;height:129.25pt'>
11  <v:imagedata o:relid="rId1" o:title="" />
12  </v:shape>
```

## 13    8.3    Future Extensibility

14 This clause provides a high-level overview of the extensibility model for Office Open XML documents, and a  
 15 description of packaging conventions in the context of DrawingML and PresentationML. Two main constructs  
 16 are described: extensibility lists (extLst/ext) and alternate content blocks (AlternateContent).

17 To illustrate certain points, a number of examples refer to versions of a (fictitious) PresentationML  
 18 consumer/producer called PML. The 2003 version is called PML 2003; the 2007 version is called PML 2007; and  
 19 so on.

### 20    8.3.1   Terminology

21 Here are some terms useful when discussing future extensibility.

- 22     • *Round tripping* involves the interchange of documents between different consumers/producers, as  
 23        well as between different versions of the same consumer/producer. The pair of consumers/producers  
 24        can be on the same or different platforms. Consider the case in which a document is created by the  
 25        PML 2007. This document is then opened by PML 2003, edited, and saved. The edited document is  
 26        now opened and used by PML 2007. In this case, the document originally created by PML 2007 has  
 27        been round-tripped through PML 2003. It is also possible to round-trip a document created by PML  
 28        2003 through PML 2007.
- 29     • A *Downrev* (or down-level) version of a consumer/producer refers to one that understands an older  
 30        version of a given schema. An *Uprev* (or up-level) version of a consumer/producer refers to one that  
 31        understands a newer version of a given schema. The terms Downrev and Uprev are typically used in  
 32        relative reference to one another. As an example, let's consider again, the two consumer/producer  
 33        PML 2003 and PML 2007, where PML 2007 was released sometime after PML 2003, and, consequently,  
 34        PML 2007 understands a newer revision of the DrawingML schema than does PML 2003. PML 2003 is  
 35        referred to as the Downrev version while PML 2007 is referred to as the Uprev version. It is assumed  
 36        that the Downrev version has less capability than the Uprev version.

### 1    8.3.2    What is Future Extensibility?

2    The main objective of future extensibility is to design an infrastructure that allows the file format to be  
3    extended for representation of data structures in future versions of a given consumer/producer.  
  
4    On the surface, that isn't hard to do; there could be a special extension bit bucket allocated across every  
5    existing schema element, and any future extension could be placed there. However, the problem is more  
6    complex than that. The infrastructure must allow document interoperability between current  
7    consumers/producers and future consumers/producers, some which have not yet even been designed or built.  
8    That is, future extensibility involves building forward compatibility into the document infrastructure while  
9    remaining compatible with the current version.

### 10    8.3.3    Future Extensibility Requirements

11    There are three design goals to be considered: visual fidelity, editability, and security.

- 12    • *Visual fidelity* involves the desire for users of two consumers/producers to see visually the same thing.  
13    This seems like a simple design goal to meet, but, in practice, is not easy to achieve. The difference lies  
14    in the capabilities of an Uprev and Downrev consumer/producer. Typically, an Uprev  
15    consumer/producer has been extended to have new base capabilities that are not present in the  
16    Downrev consumer/producer. As such, the Downrev client does not have the base primitives  
17    necessary to express visually the new capability introduced in the Uprev consumer/producer.

18    Consider the case in which PML 2007 has the capability to highlight text with a given color, while PML  
19    2003 does not. Given the desire to have PML 2007 documents interoperate with PML 2003, it is  
20    necessary for PML 2003 to some way to express visually that text highlight. For example, it might  
21    insert a picture of the highlighted text instead of inserting the text itself, since PML 2003 does know  
22    how to deal with pictures.

- 23    • *Editability* involves the desire for two consumers/producers to be able to edit the same content. Using  
24    the highlighted text example from above, despite the fact that PML 2003 and PML 2007 have different  
25    capabilities, one would still desire to edit highlighted text a regardless of which version of PML is in  
26    use. Again, this becomes difficult when the underlying capabilities of the consumer/producer versions  
27    are different.
- 28    • *Security* involves the desire to have multiple representations of the same data synchronized. This  
29    desire is referred to as security for the reason that out-of-sync representation can have dire  
30    consequences. For example, there might be multiple representations for a sensitive piece of  
31    information such as a Social Security number. If this piece of information were edited, it would be  
32    critical to keep all alternate representations in sync. What if that information were deleted  
33    altogether? If only one representation was deleted but others remained, it would be possible for one  
34    to have sensitive information in a document when the intent was to have it deleted.

35    One solution to try to solve the visual fidelity and editability goals is to have two equivalent representations for  
36    the same construct. In the highlighted text example above, a picture of the highlighted text (also called a  
37    rasterized version of the highlighted text) is an equivalent representation of the highlighted text itself. One

- 1 might use the highlighted text representation when the underlying consumer/producer is capable of  
 2 understanding it; otherwise, the picture version would be used.
- 3 Clearly, these design goals compete with each other. While a picture representation of text is capable of  
 4 capturing full visual fidelity of how extended text looks, obviously that representation doesn't offer the same  
 5 editability properties of text. One can't manipulate a picture of text nearly as easily as the text itself.
- 6 The competing nature of these design characteristics requires that one choose an extensibility construct that  
 7 offers the best mix of desired characteristics. It will not always be possible to have visual fidelity, editability,  
 8 and security, at the same time.

### 9    8.3.4      Future Extensibility Constructs

10 The two extensibility constructs used to represent extensions in OOXML schemas are:

#### 11    8.3.4.1      extLst/ext

12 The extLst construct is used for straight-up extension of existing schemas of a non-visual nature. The term  
 13 *straight up* refers to the notion that sometimes extension means refining the semantics of existing constructs.  
 14 In doing so, an extension sometimes overrides the meaning of previous schemas. extLst and ext were not  
 15 designed for this scenario. Instead of overriding existing meaning, these two constructs purely augment  
 16 existing schemas. The nature of the augmentation must be such that it does not overlap any semantics  
 17 embedded in existing schema constructs.

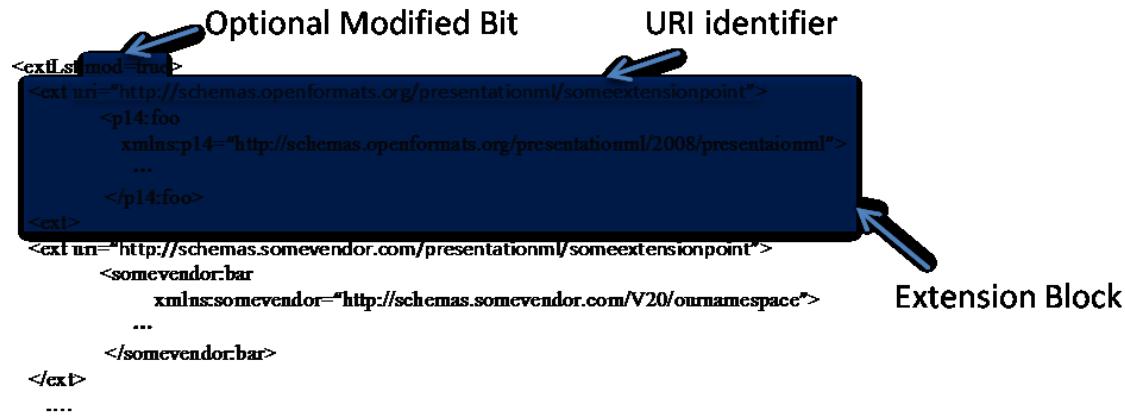
18 Consider a schema that represents an address, which contain a house number, a street name, a city, a state,  
 19 and a postal code. An example of a straight-up extension is the addition of a field that describes whether this  
 20 address is a business or residential location. This is a straight-up extension because the notion of whether an  
 21 address is business or residential does not conflict with any information that is embedded in the existing  
 22 schema. Now let's consider the case in which the Postal Service replaces a purely numeric postal code with one  
 23 that can contain alphanumeric characters. Such a change would not be a straight-up extension because the  
 24 new representation conflicts with the old representation of the same data, namely the postal code.

25 Some extensions are visual in nature. An example would be extending a schema to represent text that has  
 26 been highlighted. By definition, highlighting text is a visual extension. Contrast that to the case of adding a  
 27 business or residential classification. The latter does not necessarily involve any visual change to the way data  
 28 is presented.

29 The extLst and ext constructs are for extensions of a non-visual nature. The main reason their use is limited to  
 30 this scenario lies in the fact that they do not offer the capability to create alternative representations of the  
 31 same data.

#### 32    8.3.4.1.1      extLst/ext Syntax

33 The extLst and ext construct can be placed only at specific locations within the OOXML schemas. Its syntax is  
 34 as below:



- 1      **Optional Modified Bit**
- 2      An extLst is a list of extension blocks that are placed one after the other. Each extension block has an uri attribute, which serves as an identifier to indicate the kind of extension that has been placed here. Upon encountering an extension block, a processing consumer, will determine whether it knows how to process extensions matching that attribute. If the consumer knows how to process such an extension, the markup contained within that extension block is processed. Otherwise, the extension block is preserved so long as the underlying structure being extended by the extLst has not been deleted.
- 3      There is no limit to the number of ext extension block constructs. The order of extension blocks can be arbitrary.
- 4      An optional modified attribute, mod, is available on extLst. This attribute is set to true whenever an edit has occurred at the extended location. Its presence is to aid up-level clients that receive modified documents that have been edited in down-level consumers/producers.
- 5      **Round-Trip Behavior of ext Blocks**
- 6      When extLsts are processed, some consumers/producers will understand some extensions, but not others. The preservation model of ext blocks is that unprocessed extensions are always preserved and retained as long as the underlying schema extended by the structure remains.
- 7      **Example**
- 8      Consider the case in which the notion that each shape can be associated with a given layer, is to be added. The schema for this might look like the following:

```

<p:sld xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/3/main"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:p="http://schemas.openxmlformats.org/presentationml/2006/3/main">
    <p:cSld name="">
        <p:spTree>
            <p:nvGrpSpPr>...
            <p:grpSpPr>...
            <p:sp>
                <p:nvSpPr>
                    <p:cNvPr id="3" name="Rectangle 3" descr="" />
                    <p:cNvSpPr/>
                    <p:nvPr/>
                    <p:extLst mod="false">
                        <p:ext uri="http://.../layerExtension">
                            <p14:spLayer xmlns:p14="http://schemas.openxmlformats.org/drawingml/2009/3/main" layer="1"/>
                        </p:ext>
                    </p:extLst>
                </p:nvSpPr>
                <p:spPr>...
                <p:style>...
                <p:txBody>...
            </p:sp>
        </p:spTree>
    </p:cSld>
    <p:clrMapOvr>...
    <p:timing>...
</p:sld>

```

1

- 2 The extLst block is under the non-visual shape properties (i.e., p:nvSpPr). A uri attribute identifies the extension.
- 4 Now consider how this markup will be processed by PML 2007 and PML 2009, where PML 2009 is an up-level version of PML 2007.
- 6 PML 2007 processes the above markup, and ignores the ext block because it doesn't understand this extension. However, this block will be preserved for any other consumer/producer that may understand it.
- 8 PML 2009 processes the above markup, and understands how to deal with layer extensions as indicated by the uri. The spLayer extension is returned, PML 2009 processes the extension and is responsible for writing out any updates to this markup, as required. For example, layer might be changed from 1 to 2.
- 11 Note that the extension is a straight-up extension in that layer information is orthogonal to all other non-visual properties, such as the ID, name, and description.
- 13 Being non-visual in nature, the information in this extension does not directly affect the appearance of the shape.
- 15 **8.3.4.2 AlternateContent Blocks**
- 16 An *alternate content block* allows for an alternative representation of information. In some cases, the desire will be to revise a schema with a newer representation. It will also be common to express visual differences using alternate content blocks. Recall that typically lower-level clients will not have the same capability as their future cousins (i.e., the up level version). As such, any future extension done in the up-level version will need to be expressed in a form that the lower-level version can understand. Hence, the need for alternate representations.

## 1    8.3.4.2.1              AlternateContent Syntax

```

2   <AlternateContent>
3     <Choice Requires="namespacefoo">
4       <Somemarkup/>
5     <Choice Requires="namespacefoo namespacefoobar">
6       <Somealternatemarkup/>
7     <Fallback>
8       <Choiceoflastresort/>
9   </AlternateContent>
```

10    The `AlternateContent` element and its children, `Choice` and `Fallback`, are used to provide alternates for  
 11    specified content. Each `Choice` element is examined in turn. The `Requires` attribute specifies a set of space-  
 12    delimited namespaces that must be understood in order to select that choice. If there is a match between  
 13    required namespaces and what the consumer understands, the appropriate `Choice` is returned. If there are  
 14    multiple possible matches, only the first match is returned. An optional `Fallback` element can be used, and is  
 15    utilized as a default when no match occurs.

## 16    8.3.4.2.2              Example

17    Using PML 2007 and PML 2009, let's assume that PML 2007 understands some current schema version while  
 18    PML 2009 will understand some future extended version of the current schema.

19    PML 2007 knows how to handle connectors:



20

21

22    Now let's suppose that we desire to add a notion of labels on connectors:



23

24    Two alternate representations are required: PML 2009's schema has been extended to natively understand  
 25    how to represent a label on a connector; however, PML 2007 does not. With PML 2007, we approximate this  
 26    feature by representing the connector and label independently. The label is represented using a textbox. To  
 27    keep the two elements together, for convenience sake, they are grouped.

28    Looking at the corresponding XML, PML 2007's markup looks as follows:

**Note use of "p" to denote  
2007 DrawingML  
namespace**

```

<Choice Requires="p"> ←
  <p:nvGrpSpPr>...
  <p:grpSpPr>...
  <p:grpSp>
    <p:nvGrpSpPr>...
    <p:grpSpPr>
      <a:xfrm>...
    </p:grpSpPr>
  </p:grpSp>
  <p:cxnSp>
    <p:nvCxnSpPr>
      <p:cNvPr id="31" name="Straight Connector 31" descr="" />
      <p:cNvCxnSpPr />
      <p:nvPr />
    </p:nvCxnSpPr>
    <p:spPr>...
    <p:style>...
  </p:cxnSp>
  <p:sp>
    <p:nvSpPr>...
    <p:spPr>
      <a:xfrm>...
      <a:prstGeom prst="rect">...
      <a:noFill />
    </p:spPr>
    <p:txBody>
      <a:bodyPr wrap="none">...
      <a:lstStyle />
      <a:p>
        <a:r>
          <a:rPr lang="en-US" dirty="0" smtClean="0" />
          <a:t>Here is some label</a:t>
        </a:r>
        <a:endParaRPr smtClean="0" />
      </a:p>
    </p:txBody>
  </p:sp>
</p:grpSp>
</Choice>

```

CONNECTOR

TEXTBOX

- PML 2009 has been extended such that we may represent a label natively:

"p14" denotes post-2007 DrawingML namespace

```

<Choice Requires="p14">
  <p:cxnSp>
    <p:nvCxnSpPr>
      <p:cNvPr id="23" name="Straight Connector 23" descr="" />

    <p14:cntrLblPr>
      <p:txBody>
        <a:bodyPr anchor="ctr"/>
        <a:lstStyle/>
        <a:p>
          <a:pPr align="ctr"/>
          <a:t>Here is some label</a:t>
          <a:endParaRPr/>
        </a:p>
      </p:txBody>
    </p14:cntrLblPr>

    <p:cNvCxnSpPr>
      <a:stCxn id="3" idx="0"/>
    </p:cNvCxnSpPr>
  </p:nvCxnSpPr>
  <p:spPr>
    <a:xfrm flipV="1">...
    <a:prstGeom prst="line">...
    <a:noFill/>
    <a:ln w="12700">...
    <a:effectLst>...
  </p:spPr>
  <p:style>...
</p:cxnSp>
</Choice>

```

New "cntrLblPr" (Connector Label Property) introduced under connector Shape

LABEL EXTENSION

1

- 2 The final markup putting these choices together is as follows:

```

<p:sld xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/3/main"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:p="http://schemas.openxmlformats.org/presentationml/2006/3/main"
    xmlns:p14="http://schemas.openxmlformats.org/presentationml/2008/3/main">
    <p:cSld name="">
        <p:spTree>
            <p:nvGrpSpPr>...
            <p:grpSpPr>...
            AlternateContent Block
            <AlternateContent>
                <Choice Requires="p14">
                    <!-- Uplevel Choice goes here -->
                </Choice>
                <Choice Requires="p">
                    <!-- Downlevel Choice goes here -->
                </Choice>
            </AlternateContent>
        </p:spTree>
    </p:cSld>
    <p:clrMapOvr>...
    <p:timing>...
</p:sld>

```

1

## 2 8.3.4.2.3 AlternateContent Round-Trip Behavior

3 AlternateContent maintains multiple representations for the same content. Consider an extreme case. Using  
 4 the example above, let's suppose one edited the label using PML 2007. As PML 2007 wouldn't understand  
 5 future representations, there is no possibility that it could keep PML 2009's markup consistent with the edit  
 6 performed. Considering a simple case, let's suppose one deleted the label in its entirety. PML 2007 would only  
 7 know how to delete the corresponding textbox, and would not know how to update the corresponding  
 8 cntrLblPr.

9 If this textbox contained sensitive information, one might consider this a security leak. The user's belief is that  
 10 the information in the textbox was deleted, yet it persists in an alternate representation.

11 To solve this problem, all AlternateContent choices are discarded when an edit is performed at the location  
 12 the AlternateContent is placed. It is the consuming client's responsibility to replace the discarded  
 13 AlternateContent with a new representation.

14 If an edit to the label occurred in PML 2007, the PML 2009 version is discarded.

15 If an edit occurs in PML 2009, since PML 2009 understands both PML 2007 and PML 2009 schemas, it is  
 16 possible for PML 2009 to write an updated AlternateContent Block encompassing an update to both choices.

17 **End of informative text.**