Office Open XML

Document Interchange Specification

Ecma TC45

Working Draft 1.4

Part 3: Primer

Public Distribution

August 2006

Table of Contents

[Introduction xi](#_Toc143406696)

[1. Scope 1](#_Toc143406697)

[2. Introduction to WordprocessingML 2](#_Toc143406698)

[2.1 Stories 2](#_Toc143406699)

[2.2 Basic Document Structure 2](#_Toc143406700)

[2.3 Main Document 3](#_Toc143406701)

[2.3.1 Document Backgrounds 3](#_Toc143406702)

[2.4 Paragraphs and Rich Formatting 4](#_Toc143406703)

[2.4.1 Paragraphs 4](#_Toc143406704)

[2.4.2 Runs 5](#_Toc143406705)

[2.4.3 Run Content 7](#_Toc143406706)

[2.4.4 Formatting Property Values 7](#_Toc143406707)

[2.5 Tables 8](#_Toc143406708)

[2.5.1 Introduction 8](#_Toc143406709)

[2.5.2 Table Properties 9](#_Toc143406710)

[2.5.3 Table Grid 10](#_Toc143406711)

[2.5.4 Table Rows and Cells 12](#_Toc143406712)

[2.5.5 Table Layout 14](#_Toc143406713)

[2.5.6 Fixed Width Tables 15](#_Toc143406714)

[2.5.7 AutoFit Tables 15](#_Toc143406715)

[2.5.8 Complex Table Example 16](#_Toc143406716)

[2.5.9 Vertically Merged Cells 17](#_Toc143406717)

[2.6 Sections 19](#_Toc143406718)

[2.6.1 Section Properties 20](#_Toc143406719)

[2.6.2 Section Breaks 21](#_Toc143406720)

[2.7 Custom Markup 21](#_Toc143406721)

[2.7.1 Smart Tags 22](#_Toc143406722)

[2.7.2 Custom XML Markup 23](#_Toc143406723)

[2.7.3 Structured Document Tags 25](#_Toc143406724)

[2.8 Styles 28](#_Toc143406725)

[2.8.1 Styles Part 28](#_Toc143406726)

[2.8.2 Style Definitions 28](#_Toc143406727)

[2.8.3 Paragraph Styles 30](#_Toc143406728)

[2.8.4 Character Styles 31](#_Toc143406729)

[2.8.5 Linked Styles 32](#_Toc143406730)

[2.8.6 Numbering Styles 35](#_Toc143406731)

[2.8.7 Table Styles 35](#_Toc143406732)

[2.8.8 Default Document Paragraph and Character Properties 38](#_Toc143406733)

[2.8.9 Style Inheritance 38](#_Toc143406734)

[2.8.10 Style Application 40](#_Toc143406735)

[2.8.11 Latent Styles 40](#_Toc143406736)

[2.9 Numbering 41](#_Toc143406737)

[2.9.1 Numbering Part 42](#_Toc143406738)

[2.9.2 Numbering Definitions 42](#_Toc143406739)

[2.9.3 Abstract Numbering Definitions 42](#_Toc143406740)

[2.9.4 Numbering Definition Instances 45](#_Toc143406741)

[2.9.5 Applying Numbering to Paragraphs 46](#_Toc143406742)

[2.9.6 The Complete Story 49](#_Toc143406743)

[2.9.7 Numbering Styles 50](#_Toc143406744)

[2.9.8 Referencing Numbering Styles 50](#_Toc143406745)

[2.10 Fonts 53](#_Toc143406746)

[2.10.1 Font References 53](#_Toc143406747)

[2.10.2 Font Reference Types 54](#_Toc143406748)

[2.10.3 Ambiguous Characters 54](#_Toc143406749)

[2.10.4 Font Table 55](#_Toc143406750)

[2.10.5 Font Substitution Data 55](#_Toc143406751)

[2.10.6 Font Embedding 56](#_Toc143406752)

[2.10.7 Theme Fonts 56](#_Toc143406753)

[2.11 Headers and Footers 57](#_Toc143406754)

[2.11.1 Header Part 57](#_Toc143406755)

[2.11.2 Footer Part 57](#_Toc143406756)

[2.11.3 Headers and Footers 57](#_Toc143406757)

[2.11.4 Multiple Sections 60](#_Toc143406758)

[2.11.5 Empty Header or Footer 61](#_Toc143406759)

[2.12 Footnotes and Endnotes 61](#_Toc143406760)

[2.12.1 Footnote Part 62](#_Toc143406761)

[2.12.2 Endnote Part 63](#_Toc143406762)

[2.12.3 Footnotes and Endnotes 63](#_Toc143406763)

[2.12.4 Footnote and Endnote Types 64](#_Toc143406764)

[2.12.5 Footnote and Endnote Reference 68](#_Toc143406765)

[2.13 Glossary Document 69](#_Toc143406766)

[2.14 Annotations 70](#_Toc143406767)

[2.14.1 Introduction 70](#_Toc143406768)

[2.14.2 Inline Annotations 71](#_Toc143406769)

[2.14.3 Cross-Structure Annotations 71](#_Toc143406770)

[2.14.4 Property Annotations 72](#_Toc143406771)

[2.14.5 Comments 73](#_Toc143406772)

[2.14.6 Comments Part 74](#_Toc143406773)

[2.14.7 Revisions 74](#_Toc143406774)

[2.14.8 Bookmarks 75](#_Toc143406775)

[2.14.9 Range Permissions 76](#_Toc143406776)

[2.14.10 Spelling and Grammar 77](#_Toc143406777)

[2.15 Settings 77](#_Toc143406778)

[2.15.1 Document Settings 78](#_Toc143406779)

[2.15.2 Compatibility Settings 78](#_Toc143406780)

[2.15.3 Web Settings 79](#_Toc143406781)

[2.16 Fields and Hyperlinks 79](#_Toc143406782)

[2.16.1 Fields 79](#_Toc143406783)

[2.16.2 Hyperlinks 80](#_Toc143406784)

[2.17 Mail Merge 80](#_Toc143406785)

[2.17.1 Mail Merge, WordprocessingML, and Hosting Applications 80](#_Toc143406786)

[2.17.2 Connecting Documents to an External Data Source 81](#_Toc143406787)

[2.17.3 Populating Merged Documents with External Data 82](#_Toc143406788)

[2.18 Miscellaneous Topics 83](#_Toc143406789)

[2.18.1 Text Boxes 83](#_Toc143406790)

[2.18.2 Subdocuments 84](#_Toc143406791)

[2.18.3 Importing External Content 85](#_Toc143406792)

[2.18.4 Roundtripping Alternate Content 86](#_Toc143406793)

[3. Introduction to SpreadsheetML 88](#_Toc143406794)

[3.1 Workbook 88](#_Toc143406795)

[3.1.1 Overview 88](#_Toc143406796)

[3.1.2 Minimum Workbook Scenario 88](#_Toc143406797)

[3.1.3 Example Workbook Properties 88](#_Toc143406798)

[3.1.4 fileVersion 89](#_Toc143406799)

[3.1.5 workbookView 91](#_Toc143406800)

[3.2 Sheets 92](#_Toc143406801)

[3.2.1 Minimum Worksheet Scenario 92](#_Toc143406802)

[3.2.2 Example Sheet 92](#_Toc143406803)

[3.2.3 Sheet Properties 93](#_Toc143406804)

[3.2.4 Sheet Data 94](#_Toc143406805)

[3.2.5 Supporting Features 97](#_Toc143406806)

[3.2.6 Sheet Properties 98](#_Toc143406807)

[3.2.7 sheetData Cell Table 98](#_Toc143406808)

[3.2.8 Row 99](#_Toc143406809)

[3.2.9 Cell 99](#_Toc143406810)

[3.2.10 Supporting Sheet Features 101](#_Toc143406811)

[3.2.11 Defined Names 101](#_Toc143406812)

[3.2.12 AutoFilter 101](#_Toc143406813)

[3.2.13 Merged Cells 102](#_Toc143406814)

[3.2.14 Conditional Formatting 102](#_Toc143406815)

[3.3 Comments 103](#_Toc143406816)

[3.4 Shared String Table 103](#_Toc143406817)

[3.4.1 Overview 103](#_Toc143406818)

[3.4.2 File Architecture 104](#_Toc143406819)

[3.4.3 Example: Plain Text 105](#_Toc143406820)

[3.4.4 Illustration 105](#_Toc143406821)

[3.4.5 The XML 105](#_Toc143406822)

[3.4.6 Shared String Table 111](#_Toc143406823)

[3.4.7 Cell Table 112](#_Toc143406824)

[3.4.8 Example: Rich Text 113](#_Toc143406825)

[3.4.9 Illustration 113](#_Toc143406826)

[3.4.10 Shared String Table 113](#_Toc143406827)

[3.5 Formulas 115](#_Toc143406828)

[3.6 Tables 115](#_Toc143406829)

[3.6.1 Overview 115](#_Toc143406830)

[3.6.2 File Architecture 116](#_Toc143406831)

[3.6.3 Example: Table 117](#_Toc143406832)

[3.6.4 Illustration 117](#_Toc143406833)

[3.6.5 The Sheet XML 117](#_Toc143406834)

[3.6.6 The Table XML 118](#_Toc143406835)

[3.7 AutoFilters and Sort State Information 118](#_Toc143406836)

[3.8 External Connection 118](#_Toc143406837)

[3.8.1 Overview 118](#_Toc143406838)

[3.8.2 OLAP Connection 120](#_Toc143406839)

[3.8.3 Pivot XML fragment 120](#_Toc143406840)

[3.8.4 Connection XML 120](#_Toc143406841)

[3.8.5 Web Query 121](#_Toc143406842)

[3.8.6 QueryTable XML 122](#_Toc143406843)

[3.8.7 Connection XML 122](#_Toc143406844)

[3.8.8 Unused Connection 123](#_Toc143406845)

[3.8.9 ODBC 123](#_Toc143406846)

[3.8.10 Connection XML 123](#_Toc143406847)

[3.8.11 SQL 124](#_Toc143406848)

[3.8.12 Connection XML 125](#_Toc143406849)

[3.8.13 Text Import 125](#_Toc143406850)

[3.8.14 Connection XML 126](#_Toc143406851)

[3.9 External Links 127](#_Toc143406852)

[3.9.1 Overview 127](#_Toc143406853)

[3.9.2 Formula Example 127](#_Toc143406854)

[3.9.3 Sheet XML 128](#_Toc143406855)

[3.9.4 Workbook Relationships 130](#_Toc143406856)

[3.9.5 Supporting Workbook Cache (Cell C2) 130](#_Toc143406857)

[3.9.6 External Link (Cell C2) 131](#_Toc143406858)

[3.9.7 Supporting Workbook Cache (Cell B2) 131](#_Toc143406859)

[3.9.8 External Link (Cell B2) 132](#_Toc143406860)

[3.9.9 Hyperlink Example 132](#_Toc143406861)

[3.9.10 Worksheet XML 133](#_Toc143406862)

[3.9.11 Relationship 133](#_Toc143406863)

[3.10 Pivot Table, Pivot Cache, and Common Types 133](#_Toc143406864)

[3.10.1 Feature Overview 133](#_Toc143406865)

[3.10.2 File Architecture 135](#_Toc143406866)

[3.10.3 Example - Native with Range Source 136](#_Toc143406867)

[3.11 Styles 151](#_Toc143406868)

[3.11.1 Overview 151](#_Toc143406869)

[3.11.2 File Architecture 152](#_Toc143406870)

[3.11.3 Organization in the Styles Part 153](#_Toc143406871)

[3.11.4 Number Format Expressions 153](#_Toc143406872)

[3.11.5 Font Definitions 153](#_Toc143406873)

[3.11.6 Fill Definitions 153](#_Toc143406874)

[3.11.7 Borders Definitions 154](#_Toc143406875)

[3.11.8 Master Records 154](#_Toc143406876)

[3.11.9 Named Cell Styles 154](#_Toc143406877)

[3.11.10 Differential Formatting Records 154](#_Toc143406878)

[3.11.11 Custom Table Style Definitions 155](#_Toc143406879)

[3.11.12 Example 155](#_Toc143406880)

[3.11.13 Illustration 155](#_Toc143406881)

[3.11.14 File Architecture 157](#_Toc143406882)

[3.11.15 The XML for this Example 157](#_Toc143406883)

[3.11.16 Custom Table Style 164](#_Toc143406884)

[3.12 Calculation Chain 166](#_Toc143406885)

[3.12.1 Overview 166](#_Toc143406886)

[3.12.2 Example 166](#_Toc143406887)

[3.13 Worksheet Metadata 170](#_Toc143406888)

[3.13.1 Overview 170](#_Toc143406889)

[3.13.2 File Architecture – Relationships 172](#_Toc143406890)

[3.13.3 Example 172](#_Toc143406891)

[3.14 Shared Workbook Revisions 183](#_Toc143406892)

[3.14.1 Overview 183](#_Toc143406893)

[3.14.2 How It Works 183](#_Toc143406894)

[3.14.3 Example 184](#_Toc143406895)

[3.15 Query Tables 192](#_Toc143406896)

[3.15.1 Overview 192](#_Toc143406897)

[3.15.2 Web Query Example 192](#_Toc143406898)

[3.15.3 Text Import Example 193](#_Toc143406899)

[3.15.4 Access Table Example 194](#_Toc143406900)

[3.16 Volatile Dependencies 195](#_Toc143406901)

[3.17 Custom XML Mappings 195](#_Toc143406902)

[3.17.1 Overview 195](#_Toc143406903)

[3.17.2 File Architecture: Relationships 195](#_Toc143406904)

[3.17.3 Conceptual Model 196](#_Toc143406905)

[3.17.4 Example 196](#_Toc143406906)

[4. Introduction to PresentationML 202](#_Toc143406907)

[4.1 Presentation 202](#_Toc143406908)

[4.1.1 Basic Utilities 202](#_Toc143406909)

[4.1.2 Empty Element 202](#_Toc143406910)

[4.1.3 Name 203](#_Toc143406911)

[4.1.4 Direction 203](#_Toc143406912)

[4.1.5 Index and Index Range 203](#_Toc143406913)

[4.1.6 Slide Show ID 203](#_Toc143406914)

[4.1.7 Slide List Choice 204](#_Toc143406915)

[4.1.8 Slide Relationship 204](#_Toc143406916)

[4.1.9 Customer Data 204](#_Toc143406917)

[4.1.10 Future Extensibility 204](#_Toc143406918)

[4.1.11 The Presentation Object 204](#_Toc143406919)

[4.1.12 Structural Information 205](#_Toc143406920)

[4.1.13 Presentation-Level Properties 206](#_Toc143406921)

[4.1.14 Presentation Properties 210](#_Toc143406922)

[4.1.15 HTML Publish Properties 210](#_Toc143406923)

[4.1.16 Print Options Properties 212](#_Toc143406924)

[4.1.17 Slide Show Properties 212](#_Toc143406925)

[4.1.18 View Properties 213](#_Toc143406926)

[4.2 Comments 214](#_Toc143406927)

[4.2.1 Introduction 214](#_Toc143406928)

[4.2.2 Functional Overview 214](#_Toc143406929)

[4.2.3 Comment Author List 214](#_Toc143406930)

[4.2.4 Comment List 215](#_Toc143406931)

[4.3 Slides, Masters, Layouts, and Placeholders 216](#_Toc143406932)

[4.4 Future Extensibility 216](#_Toc143406933)

[4.5 Slide Synchronization 216](#_Toc143406934)

[4.5.1 Introduction 216](#_Toc143406935)

[4.5.2 Slide Update Info 216](#_Toc143406936)

[4.6 Animation 218](#_Toc143406937)

[4.6.1 Introduction 218](#_Toc143406938)

[4.6.2 Slide Transitions 218](#_Toc143406939)

[4.6.3 Timeline Overview 219](#_Toc143406940)

[4.6.4 Timeline Construction 220](#_Toc143406941)

[4.6.5 Animation Behaviors 222](#_Toc143406942)

[4.6.6 Conditional Properties 224](#_Toc143406943)

[4.6.7 Build Animations 225](#_Toc143406944)

[5. Introduction to DrawingML 227](#_Toc143406945)

[5.1 Audio and Video 227](#_Toc143406946)

[5.1.1 Introduction 227](#_Toc143406947)

[5.1.2 Functional Overview 227](#_Toc143406948)

[5.1.3 DrawingML Syntax 227](#_Toc143406949)

[5.2 Text 229](#_Toc143406950)

[5.2.1 Introduction 229](#_Toc143406951)

[5.2.2 Overview 230](#_Toc143406952)

[5.2.3 Body Level Properties 232](#_Toc143406953)

[5.2.4 Coordinate Systems and Transformations 239](#_Toc143406954)

[5.2.5 Introduction 239](#_Toc143406955)

[5.2.6 Base Representations 239](#_Toc143406956)

[5.2.7 Scalings and Rotations on Base Representations 240](#_Toc143406957)

[5.2.8 Flips 242](#_Toc143406958)

[5.2.9 Group Representations 242](#_Toc143406959)

[5.2.10 Translation (Moving of Groups) 243](#_Toc143406960)

[5.2.11 Group Rotation 244](#_Toc143406961)

[5.2.12 Group Scaling 245](#_Toc143406962)

[5.2.13 Translation, Scaling, and Rotation of Group Members 246](#_Toc143406963)

[5.2.14 Transformation Computations 248](#_Toc143406964)

[5.2.15 Apply Shape Transforms 248](#_Toc143406965)

[5.2.16 Apply Group Transforms 249](#_Toc143406966)

[5.3 Styles 250](#_Toc143406967)

[5.3.1 Introduction 250](#_Toc143406968)

[5.3.2 Shared Style Sheet 250](#_Toc143406969)

[5.4 Future Extensibility 263](#_Toc143406970)

[5.4.1 Terminology 263](#_Toc143406971)

[5.4.2 What is Future Extensibility? 264](#_Toc143406972)

[5.4.3 Future Extensibility Requirements 264](#_Toc143406973)

[5.4.4 Future Extensibility Constructs 265](#_Toc143406974)

[5.4.5 Shape Definitions and Attributes 271](#_Toc143406975)

[5.4.6 Introduction 271](#_Toc143406976)

[5.4.7 The Coordinate Systems 272](#_Toc143406977)

[5.4.8 Specifying a Preset Shape 273](#_Toc143406978)

[5.4.9 Defining a Preset Geometry 273](#_Toc143406979)

[5.4.10 Adjusting a Preset Geometry 273](#_Toc143406980)

[5.4.11 Specifying a Custom Shape 275](#_Toc143406981)

[5.5 Diagrams 279](#_Toc143406982)

[5.5.1 Introduction 279](#_Toc143406983)

[5.5.2 Element Property Set 280](#_Toc143406984)

[5.5.3 Color Transforms 286](#_Toc143406985)

[5.5.4 Style Definition 291](#_Toc143406986)

[5.5.5 Layout 293](#_Toc143406987)

[5.5.6 Drawing Anchors 318](#_Toc143406988)

[5.6 3-D 318](#_Toc143406989)

[5.6.1 Introduction 318](#_Toc143406990)

[5.6.2 3-D 318](#_Toc143406991)

[5.6.3 Styles 324](#_Toc143406992)

[5.7 Tables 328](#_Toc143406993)

[5.8 Shape Properties and Effects 328](#_Toc143406994)

[5.9 Chart 328](#_Toc143406995)

[5.10 Pictures 328](#_Toc143406996)

[5.10.1 Introduction 328](#_Toc143406997)

[5.10.2 Specifying a Basic Picture 328](#_Toc143406998)

[5.10.3 Attaching Properties to this Picture 329](#_Toc143406999)

[5.10.4 Transforming this Picture 331](#_Toc143407000)

[6. Introduction to VML 333](#_Toc143407001)

[7. Introduction to Shared MLs 334](#_Toc143407002)

[7.1 Metadata 334](#_Toc143407003)

[7.1.1 Metadata Properties 334](#_Toc143407004)

[7.1.2 Core Properties 335](#_Toc143407005)

[7.1.3 Extended Properties 335](#_Toc143407006)

[7.1.4 Custom Properties 335](#_Toc143407007)

[7.1.5 Variant Types 335](#_Toc143407008)

[7.2 Math 335](#_Toc143407009)

[7.2.1 Accent Object 336](#_Toc143407010)

[7.2.2 Bar Object 336](#_Toc143407011)

[7.2.3 Border Box Object 336](#_Toc143407012)

[7.2.4 Box Object 337](#_Toc143407013)

[7.2.5 Delimiters 337](#_Toc143407014)

[7.2.6 Equation Array Object 337](#_Toc143407015)

[7.2.7 Fraction Object 338](#_Toc143407016)

[7.2.8 Function Apply Object 338](#_Toc143407017)

[7.2.9 Group Character Object 339](#_Toc143407018)

[7.2.10 Upper and Lower Limits 339](#_Toc143407019)

[7.2.11 Matrix Object 339](#_Toc143407020)

[7.2.12 N-ary Object 340](#_Toc143407021)

[7.2.13 Phantom Object 340](#_Toc143407022)

[7.2.14 Radical Object 341](#_Toc143407023)

[7.2.15 Scripts (Superscript, Subscript, SubSuperscript, PreSubSuperscript) 342](#_Toc143407024)

[7.3 Bibliography 342](#_Toc143407025)

[7.4 Custom XML Data 342](#_Toc143407026)

Introduction

This Standard describes a family of XML schemas, collectively called Office Open XML, which define the XML vocabularies for word-processing, spreadsheet, and presentation documents, as well as the packaging of documents that conform to these schemas.

The goal is to enable the implementation of the Office Open XML formats by the widest set of tools and platforms, fostering interoperability across office productivity applications and line-of-business systems, as well as to support and strengthen document archival and preservation, all in a way that is fully compatible with the large existing investments in Microsoft Office documents.

This Standard is Part 3 of a multi-part standard covering Open XML-related technology.

* Part 1: "Fundamentals"
* Part 2: "Open Packaging Conventions"
* Part 3: "Primer" (this document)
* Part 4: "Markup Language Reference"
* Part 5: "Markup Compatibility"

# Scope

This Standard defines Office Open XML's vocabularies and document representation and packaging. It also specifies requirements for consumers and producers of Office Open XML.

# Introduction to WordprocessingML

This clause is informative.

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

## Stories

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

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

## Basic Document Structure

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

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

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

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

<?xml version="1.0"?>  
<w:document xmlns:w="…">  
 <w:body>  
 <w:p>  
 <w:r>  
 <w:t>Hello, world.</w:t>  
 </w:r>  
 </w:p>  
 </w:body>  
</w:document>

## Main Document

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
* Section properties
* Custom markup (custom XML, structured document tags)
* Tables
* Context-free chunks/alternate format chunks
* Annotations
* Revision markers
* Range permission markers
* Future extensibility containers

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

### 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:

<w:background w:bgColor="5C83B4">  
 <v:background id="\_x0000\_s1025" o:bwmode="white" fillcolor="#5c83b4  
 [3204] o:targetscreensize="800,600">  
 <v:fill color2="fill darken(118) method="linear sigma" focus="100%"  
 type="gradient"/>  
 </v:background>  
</w:bgPict>

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

## Paragraphs and Rich Formatting

### Paragraphs

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

Consider the paragraph fragment "The quick brown fox jumped …" which is centered on a paragraph. As all the text in the paragraph is emphasized using italics, in the XML, the contents of the paragraph will have that property as well as the justify-center property, for example:

<w:p>  
 <w:pPr>  
 <w:jc w:val="center"/>  
 <w:rPr>  
 <w:i/>  
 </w:rPr>  
 </w:pPr>

<w:r>  
 <w:rPr>  
 <w:i/>  
 </w:rPr>  
 <w:t>The quick brown fox jumped…</w:t>  
 </w:r>  
</w:p>

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

A paragraph's properties are specified via the pPr element. Some examples of paragraph properties are alignment, border, hyphenation override, indentation, line spacing, shading, text direction, and widow/orphan control.

It should also be noted that a pPr element may contain a set of run properties within a rPr element – these properties are applied to the run which contains the glyph which represents the paragraph mark and not the entire paragraph.

### Runs

The next level of the document hierarchy is the run, which defines a region of text with a common set of properties, represented by the r element. An r element allows the producer to combine breaks, styles, or formatting properties, applying the same information to all the parts of the run.

Just as a paragraph can have properties, so too can a run. All of the elements inside an r element have their properties controlled by a corresponding optional rPr run properties element, which must be the first child of the r element. In turn, the rPr element is a container for a set of property elements that are applied to the rest of the children of the r element. The elements inside the rPr container element allow the consumer to control whether the text in the following t elements is bold, underlined, or visible, for example. Some examples of run properties are bold, border, character style, color, font, font size, italic, kerning, disable spelling/grammar check, shading, small caps, strikethrough, text direction, and underline.

Consider the following run within a WordprocessingML document:

<w:r>  
 <w:rPr>  
 <w:b/>  
 <w:i/>  
 </w:rPr>  
 <w:t>quick</w:t>  
</w:r>

The run specifies two formatting properties in its run contents: bold and italic. These properties are therefore applied to all content within this run.

A producer can break a run into an arbitrary number of smaller runs, provided each smaller run uses the same set of properties, without changing the content of the document.

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

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

<w:r>  
 <w:rPr>  
 <w:i/>  
 <w:rPr>  
 <w:t>emphasized</w:t>  
</w:r>

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

<w:r>  
 <w:t>only one</w:t>  
</w:r>

<w:r>  
 <w:t xml:space="preserve">word is </w:t>  
</w:r>  
<w:r>  
 <w:rPr>  
 <w:i/>  
 <w:rPr>  
 <w:t>empha</w:t>  
</w:r>

<w:r>  
 <w:rPr>  
 <w:i/>  
 <w:rPr>  
 <w:t>sized</w:t>  
</w:r>

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

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

The following run contains two sentences:

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

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

<w:r>  
 <w:rPr>  
 <w:b/>  
 </w:rPr>  
 <w:t xml:space="preserve">Hello, world. </w:t>  
</w:r>

<w:r>  
 <w:t>How are you, today?</w:t>  
</w:r>

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

### Run Content

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

* Text
* Deleted text
* Soft line breaks
* Field codes
* Deleted field codes
* Footnote/endnote reference marks
* Simple fields
* Page numbers
* Tabs
* Ruby text
* DrawingML content
* Embedded objects
* Pictures

#### Text

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

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

* delText - Deleted text
* instrText - Field codes
* delInstrText - Deleted field codes

These four types of text are defined using unique elements in WordprocessingML so that simple consumers can determine the text of the document simply by grabbing the contents of the t node, without needing to check where revisions start and end, etc. to determine the state of the text contents.

It is also notable that these are the only elements in a WordprocessingML document's main document part that can contain a XML text node.

### Formatting Property Values

Most of the children of an rPr or pPr element have a single val attribute that is limited to a specific set of values. For example, the b (bold) element causes the text that follows it to be bold when the b element has a val attribute with value on. If the val attribute isn't present for the b element, it defaults to "on". Therefore, <w:b/> is equivalent to <w:b w:val="on"/>.

Aside from the default values, which are documented with each element, this is particularly important when specifying the difference between omitting a formatting property and explicitly turning it off.

For example, consider the following run:

<w:r>  
 <w:rPr>  
 <w:b w:val="off"/>  
 </w:rPr>  
 <w:t xml:space="preserve">Hello, world. </w:t>  
</w:r>

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

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

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

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

## Tables

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

### Introduction

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

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

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

|  |
| --- |
|  |

This table is represented by the following WordprocessingML:

<w:tbl>  
 <w:tblPr>  
 <w:tblW w:w="5000" w:type="pct"/>  
 <w:tblBorders>  
 <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:left w:val="single" w:sz="4 w:space="0" w:color="auto"/>  
 <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 </w:tblBorders>  
 </w:tblPr>  
 <w:tblGrid>  
 <w:gridCol w:w="10296"/>  
 </w:tblGrid>  
 <w:tr>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="0" w:type="auto"/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
 </w:tr>  
</w:tbl>

This table specifies table-wide properties of 100% of page width (tblW) and the set of table borders (tblBorders), the table grid which defines a set of shared vertical edges within the table (discussed later), and a single row. end example]

### Table Properties

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

[Example: Consider the following simple WordprocessingML table:

|  |  |
| --- | --- |
|  |  |

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

<w:tbl>  
 <w:tblPr>  
 <w:tblW w:w="0" w:type="auto"/>  
 <w:tblBorders>  
 <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:left w:val="single" w:sz="4 w:space="0" w:color="auto"/>  
 <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:insideH w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:insideV w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 </w:tblBorders>  
 </w:tblPr>  
 <w:tblGrid>  
 ...  
 </w:tblGrid>  
 <w:tr>  
 ...  
 </w:tr>  
</w:tbl>

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

### Table Grid

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

[Example: Returning to the earlier 'one-cell empty table' example, the table has one column with a width of 10,296 twips.

<w:tblGrid>  
 <w:gridCol w:w="11204"/>  
</w:tblGrid>

end example]

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

|  |  |  |
| --- | --- | --- |
|  | |  |
|  |  | |

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

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

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

<w:tblGrid>  
 <w:gridCol w:w="2952"/>  
 <w:gridCol w:w="4416"/>  
 <w:gridCol w:w="1488"/>  
</w:tblGrid>  
<w:tr>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="7368" w:type="dxa"/>  
 <w:gridSpan w:val=”2”/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="1488" w:type="dxa"/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
</w:tr>  
<w:tr>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="2952" w:type="dxa"/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="5904" w:type="dxa"/>  
 <w:gridSpan w:val=”2”/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
</w:tr>

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

### Table Rows and Cells

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

A tr element has one formatting child element, trPr, which defines the row properties (such as the row’s width) and whether it can split across a page. Each property is defined by an individual child element under the trPr element. The complete set of table row properties can be found on the definition for the trPr element. As well, a table row can contain two types of content: custom markup (custom XML or structured document tags), and table cells.

The cells in a row contain the table’s content and are defined by tc elements, which are analogous to HTML <td> tags.

A tc element has one formatting child element, tcPr, which defines the properties for the cell. Each unique property is specified by a child element of this element. The complete set of table cell properties can be found on the definition for the tcPr element. As well, a table cell can contain any valid block-level content, which allows for the nesting of paragraphs and tables within table cells.

[Example: In the example below, the tcW element defines the width of the cell, where the attribute w is the value in twips. Here the width of the cell is 8,856 units, where units are defined by the attribute type. In this case, dxa represents twips.

<w:tr>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="8856" w:type="dxa"/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
</w:tr>

The tc element contains the cell's content, which, in this case, is an empty p element. end example]

[Example: Consider a table having one cell, which contains the text “Hello, world”:

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

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

<w:tr>  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="1770" w:type="dxa"/>  
 </w:tcPr>  
 <w:p>  
 <w:r>  
 <w:t>Hello, World</w:t>  
 </w:r>  
 </w:p>  
 </w:tc>  
</w:tr>

end example]

At both the row and cell levels, the properties must also specify how the rows and cells will be placed on the table grid.

The trPr element contains information about the number of grid units which should be omitted ('skipped') before and after the row is complete using the gridBefore and gridAfter elements, allowing rows to start at different columns on the grid, as well as a preferred width for that leading/trailing space using the wBefore and wAfter elements. The tcPr element also contains grid information pertaining to how many grids a cell spans using the gridSpan element, which determines how many grid units are consumed by the current cell, as well as a preferred width for that cell using the tcW element.

[Example: In the earlier complex table having two rows of two differently sized cells, a consumer shall represent that table containing three grid columns (one per distinct vertical line). Consider the following XML for the first row of that table:

<w:tr>  
 …  
 <w:tc>  
 <w:tcPr>  
 <w:tcW w:w="5145" w:type="dxa" />  
 <w:gridSpan w:val="2" />  
 </w:tcPr>  
 <w:p />  
 </w:tc>

<w:tc>  
 <w:tcPr>  
 <w:tcW w:w="2145" w:type="dxa" />  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
</w:tr>

Again, the gridSpan element is the number of grid columns that cell spans when being laid out on the table grid. In this example, the first cell of the first row contains two grid columns. As well, the cell specifies its preferred width using the tcW element, which tells the consumer the width desired by that cell at layout time. end example]

It is important to note that every width in a table is a preferred width - because the table must satisfy the grid at all times, conflicting table properties must be resolved by overriding preferred widths in a specific manner, shown below.

### Table Layout

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

* Table-level properties (e.g. preferred width)
* Table column grid
* Row-level properties (e.g. grid units before/after row start/end)
* Cell-level properties (e.g. number of grid units spanned)

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

### Fixed Width Tables

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

* The table grid is used to create the set of shared columns in the table and their initial widths as defined in the tblGrid element
* 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.
* 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.
* 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.
* Each additional cell is placed on the grid.
* 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.
* 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.
* 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.

### AutoFit Tables

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

* Perform the steps above to lay out the fixed width version of the table.
* 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).
* 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.
* 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

### Complex Table Example

The properties above are best illustrated by example:

[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):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | |  | |
|  | |  | |
|  |

1 2 3 4 5 6 7

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 with 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 stand 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:

<w:tr>  
 <w:trPr>  
 <w:gridBefore w:val="3"/>  
 <w:wBefore w:w="2748" w:type="dxa"/>  
 </w:trPr>  
 …  
</w:tr>

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | |  | |
|  | |  | |
| longtextstringwithnobreakingcharacters |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | |  | |
|  | |  | |
| longtextstringwithnobreakingcharacters |

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

### Vertically Merged Cells

Although the previous examples may have implied that tables have strict definition of rows, table cells can also be merged vertically. The tcPr element may containt the vmerge element that defines the extent of vertically merged grid columns within a table. A vmerge element with its val attribute set to restart marks the start of a vertically merged cell range. A vmerge element with the val attribute set to continue (the default value) marks the continuation of a vertically merged grid column. Cells between the first and last merged cell that are part of the vertical merge each must have a vmerge element to continue the vertical merge.

[Example: 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 |

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

|  |  |
| --- | --- |
| First cell, first row | Last cell, first row Last cell, second row |
| First cell, second row |

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

<w:tr>  
 <w:tc>  
 <w:p>  
 <w:r>  
 <w:t>First cell, first row</w:t>  
 </w:r>  
 </w:p>  
 </w:tc>

<w:tc>  
 <w:tcPr>  
 <w:vmerge w:val="restart"/>  
 </w:tcPr>  
 <w:p>  
 <w:r>  
 <w:t>Last cell, first row</w:t>  
 </w:r>  
 </w:p>

<w:p>  
 <w:r>  
 <w:t>Last cell, second row</w:t>  
 </w:r>  
 </w:p>  
 </w:tc>  
</w:tr>

<w:tr>  
 <w:tc>  
 <w:p>  
 <w:r>  
 <w:t>First cell, second row</w:t>  
 </w:r>  
 </w:p>  
 </w:tc>

<w:tc>  
 <w:tcPr>  
 <w:vmerge/>  
 </w:tcPr>  
 <w:p/>  
 </w:tc>  
</w:tr>

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

## Sections

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

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

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

<w:body>  
 <w:p>  
 <w:pPr>  
 <w:sectPr>  
 <w:pgSz w:w="12240" w:h="15840"/>  
 <w:pgMar w:top="1440" w:right="1800" w:bottom="1440"  
 w:left="1800" w:header="720" w:footer="720" w:gutter="0"/>  
 <w:cols w:space="720"/>  
 <w:docGrid w:linePitch="360"/>  
 </w:sectPr>  
 </w:pPr>

<w:r>  
 <w:t>This is sentence one.</w:t>  
 </w:r>  
 </w:p>

<w:p>  
 <w:r>  
 <w:t>This is sentence two.</w:t>  
 </w:r>  
 </w:p>

<w:sectPr>  
 <w:pgSz w:w="15840" w:h="12240" w:orient="landscape"/>  
 <w:pgMar w:top="1800" w:right="1440" w:bottom="1800"  
 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>

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

### Section Properties

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

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

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

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

<w:body>  
 <w:p  
 <w:pPr>  
 <w:sectPr>  
 <w:pgSz w:w="12240" w:h="15840"/>  
 <w:pgMar w:top="1440" w:right="1800" w:bottom="1440"  
 w:left="1800" w:header="720" w:footer="720" w:gutter="0"/>  
 <w:cols w:space="720"/>  
 <w:docGrid w:linePitch="360"/>  
 </w:sectPr>  
 </w:pPr>  
 <w:r>  
 <w:t>This is sentence one.</w:t>  
 </w:r>  
 </w:p>

<w:p>  
 <w:r>  
 <w:t>This is sentence two.</w:t>  
 </w:r>  
 </w:p>

<w:sectPr>  
 <w:pgSz w:w="15840" w:h="12240" w:orient="landscape"/>  
 <w:pgMar w:top="1800" w:right="1440" w:bottom="1800"  
 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>

### Section Breaks

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

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

## Custom Markup

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

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

* Smart tags
* Custom XML markup
* Structured document tags (content controls)

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

### Smart Tags

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

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



This text would translate to the following WordprocessingML markup:

<w:p w:rsidR="00672474" w:rsidRDefault="00672474">  
 <w:r>  
 <w:t xml:space="preserve">This is a stock symbol: </w:t>  
 </w:r>  
 <w:smartTag w:uri="urn:schemas-microsoft-com:office:smarttags" w:element="stockticker">  
 <w:r>  
 <w:t>MSFT</w:t>  
 </w:r>  
 </w:smartTag>  
</w:p>

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

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

The first of these is the namespace for this smart tag (contained in the uri attribute). This allows the smart tag to specify a URI which should be round-tripped with this smart tag and be available to a consumer. It is intended to be used to specify a family of smart tags to which this one belongs – for example, in the sample above, the smart tag belongs to the urn:schemas-microsoft-com:office:smarttags namespace.

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

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

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

<w:smartTag w:uri="urn:schemas-microsoft-com:office:smarttags" w:element="stockticker">  
 <w:smartTagPr>  
 <w:attr w:name="fullCompanyName" w:val="Microsoft Corporation"/>  
 </w:smartTagPr>  
 <w:r>  
 <w:t>MSFT</w:t>  
 </w:r>  
</w:smartTag>

The resulting XML, as seen above, simply adds an attr element which specifies the property and value for the property bag. end example]

A producer can embed a smart tag around any run-level content in a WordprocessingML document in order to embed additional information about the family and type of the data contained within. This allows ‘tagging’ of specific regions of a document with these semantics without need to provide context beyond the information provided in the uri and element attributes.

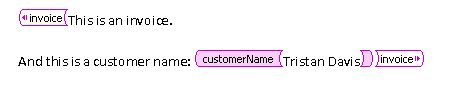
A consumer can read this smart tag data and provide additional functionality around these namespace/element pairs, which may or may not be specific to that smart tag type in the document. Examples of this functionality include: the ability to add/remove this markup via a user interface, ability to provide actions to operating in the context of this data type, etc.

### Custom XML Markup

The next example of customer-defined semantics which can be embedded in a WordprocessingML document is custom XML markup. Custom XML markup allows the application of the XML elements defined in any valid XML Schema file to be applied to the contents of a WordprocessingML document in one of two locations: around a paragraph or set of paragraphs (at the block level); or around an arbitrary run or set of runs within a document (at the inline level) to provide semantics to that content within the context and structures defined by the associated XML Schema definition file.

The distinction between custom XML markup and smart tags is based on the fact that custom XML markup corresponds with the contents of a custom XML schema; which means that as shown below, custom XML markup can be used at the block-level to mark up the contents of a document on levels beyond that of one or more runs as well as on the inline (run) level. It can also be validated against a custom XML schema by a producer at run time.

[Example: Consider a simple XML Schema which defines two elements: a root element of invoice, and a child element of customerName - the first defining that this file's contents are an invoice, and the second specifying that the enclosed text as a customer's name:



This output would translate to the following WordprocessingML markup:

<w:customXml w:uri="http://www.contoso.com/2006/invoice" w:element="invoice">  
 <w:p>  
 <w:r>  
 <w:t>This is an invoice.</w:t>  
 </w:r>  
 </w:p>  
 <w:p>  
 <w:r>  
 <w:t xml:space="preserve">And this is a customer name: </w:t>  
 </w:r>  
 <w:customXml w:uri="http://www.contoso.com/2006/invoice" w:element="customerName">  
 <w:r>  
 <w:t>Tristan Davis</w:t>  
 </w:r>  
 </w:customXml>  
 </w:p>  
</w:customXml>

end example]

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

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

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.

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.

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.

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

<w:customXml w:uri="http://www.contoso.com/2006/invoice" w:element="customerName">  
 <w:customXmlPr>  
 <w:attr w:uri="http://www.contoso.com/2006/invoice" w:name="type" w:val="individual"/>  
 </w:customXmlPr>  
 <w:r>  
 <w:t>Tristan Davis</w:t>  
 </w:r>  
</w:customXml>

The resulting XML, as seen above, simply adds an attr element which specifies the attribute for the custom XML element.end example]

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

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

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

### Structured Document Tags

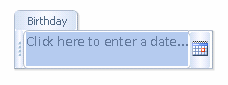
The final example of customer-defined semantics which can be embedded in a WordprocessingML document is the structured document tag (SDT).

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

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

This means that the customer can define the semantics and context of the tag, but can then use a rich set of pre-defined properties to define its behavior and appearance within the WordprocessingML document's presentation.

[Example: Consider a region which should be tagged with the semantic of "birthday", for the user to enter their date or birth into the document. Ideally, this region would also utilize a date picker to allow the user to enter the date from a calendar::



This content would translate to the following WordprocessingML markup:

<w:sdt>  
 <w:sdtPr>  
 <w:alias w:val="Birthday"/>  
 <w:id w:val="8775518"/>  
 <w:placeholder>  
 <w:docPart w:val="DefaultPlaceholder\_22479095"/>  
 </w:placeholder>  
 <w:showingPlcHdr/>  
 <w:date>  
 <w:dateFormat w:val="M/d/yyyy"/>  
 <w:lid w:val="EN-US"/>  
 </w:date>  
 </w:sdtPr>

<w:sdtContent>  
 <w:p>  
 <w:r>  
 <w:rPr>  
 <w:rStyle w:val="PlaceholderText"/>  
 </w:rPr>  
 <w:t>Click here to enter a date...</w:t>  
 </w:r>  
 </w:p>  
 </w:sdtContent>  
</w:sdt>

end example]

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

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

#### Structured Document Tag Properties

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

* Shared properties
* Locking properties
* Structured document tag type
* Type-specific properties

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

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

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

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

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

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

<w:sdtPr>

...

<w:date>

<w:dateFormat w:val="M/d/yyyy"/>

<w:lid w:val="EN-US"/>

</w:date>

</w:sdtPr>

end example]

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

#### Structured Document Tag Content

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

#### XML Mapping

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

## Styles

After looking at the primary elements of block-level content in a WordprocessingML file, it is now necessary to look at the information stored in the document that affects how this content is displayed.

The first such group of information is styles. Within a WordprocessingML file, styles are predefined sets of paragraph and/or character properties which can be applied to text within the document. This allows the formatting properties to be stored and managed independently from the content, allowing the look of document content to be changed in a single location (e.g. the look of all first-level headings is changed by changing the style with styleId Heading1 rather than looking for and changing each paragraph in the document).

[Example: The Normal paragraph style in a word processing document can have any number of formatting properties, e.g., font face = Times New Roman; font size = 12pt; paragraph justification = left). All paragraphs that reference this paragraph style would automatically inherit these properties. end example]

### Styles Part

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

The styles part stores two types of style information for the document:

* Style definitions
* Latent style information

### Style Definitions

Each style defined within a WordprocessingML document requires a style definition. The style definition contains all of the information needed by a consumer to store and display that style within a WordprocessingML document, and is defined using the style element. The style definition for any style in WordprocessingML can be divided into three segments (Note: the complete definition of style properties can be found on the reference for the style element):

* Common style properties
* Style ‘types’
* Type specific properties

Common style properties refer to the set of properties which can be used regardless of the type of style; for example, the style name, additional aliases for the style, a style ID (used by the document content to refer to the style), if style is hidden, if style is locked, etc.

[Example: Consider a style called Heading 1 in a document as follows:

<w:style w:type="paragraph" w:styleId="Heading1">  
 <w:name w:val="heading 1"/>  
 <w:basedOn w:val="Normal"/>  
 <w:next w:val="Normal"/>  
 <w:link w:val="Heading1Char"/>  
 <w:priority w:val="1"/>  
 <w:qformat/>  
 <w:rsid w:val="00F303CE"/>  
 …  
</w:style>

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

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

* Paragraph styles
* Character styles
* Linked styles (paragraph + character)
* Table styles
* Numbering styles
* Default paragraph + character properties

[Example: Consider a style called Heading 1 in a document as follows:

<w:style w:type="paragraph" w:styleId="Heading1">  
 <w:name w:val="heading 1"/>  
 <w:basedOn w:val="Normal"/>  
 <w:next w:val="Normal"/>  
 <w:link w:val="Heading1Char"/>  
 <w:priority w:val="1"/>  
 <w:qformat/>  
 <w:rsid w:val="00F303CE"/>  
 …  
</w:style>

The type attribute has a value of paragraph, which indicates that the following style definition is a paragraph style. end example]

### Paragraph Styles

The first type of style definition, paragraph styles are styles that apply to the contents of an entire paragraph as well as the paragraph mark. This definition implies that the style can define both character properties (properties that apply to text within the document) as well as paragraph properties (properties which apply to the positioning and appearance of the paragraph). Paragraph styles cannot be referenced by runs within a document, they must be referenced by the pStyle element within a paragraph’s paragraph properties (pPr) element.

A paragraph style has three defining type-specific characteristics:

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

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

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

<w:style w:type="paragraph" w:styleId="TestParagraphStyle">  
 <w:name w:val="Test Paragraph Style"/>  
 <w:qformat/>  
 <w:rsid w:val="00F85845"/>

<w:pPr>  
 <w:spacing w:line="480" w:lineRule="auto"/>  
 <w:ind w:firstLine="1440"/>  
 </w:pPr>

<w:rPr>  
 <w:rFonts w:ascii="Algerian" w:hAnsi="Algerian"/>  
 <w:color w:val="ED1C24"/>  
 <w:sz w:val="40"/>  
 </w:rPr>  
</w:style>

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

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

<w:p>  
 <w:pPr>  
 <w:pStyle w:val="TestParagraphStyle"/>  
 </w:pPr>  
 <w:r>  
 <w:t xml:space="preserve">Here is some fancy Text</w:t>  
 </w:r>  
</w:p>

The pStyle element links the paragraph with the style definition. end example]

### Character Styles

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

A character style has two defining type-specific characteristics:

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

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

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

<w:style w:type="character" w:styleId="TestCharacterStyle">  
 <w:name w:val="Test Character Style"/>  
 <w:priority w:val="99"/>  
 <w:qformat/>  
 <w:rsid w:val="00E77BF0"/>

<w:rPr>  
 <w:rFonts w:ascii="Courier New" w:hAnsi="Courier New"/>  
 <w:color w:val="FFF200"/>  
 <w:u w:val="single"/>  
 </w:rPr>  
</w:style>

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

<w:p>  
 <w:r>  
 <w:t xml:space="preserve">The following text is in the </w:t>  
 </w:r>

<w:r>  
 <w:rPr>  
 <w:rStyle w:val="TestCharacterStyle"/>  
 </w:rPr>  
 <w:t>character style</w:t>  
 </w:r>

<w:r>  
 <w:t>.</w:t>  
 </w:r>  
</w:p>

The rStyle element in the second run links that run with the style definition, inheriting the formatting properties for that run. end example]

### Linked Styles

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

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

[Note: A typical example of the use of a linked style is a quote style - if the style is applied to a paragraph, the quoted text should be indented additionally to create a block quote effect, but if the style is applied to text in a paragraph, only the character level effects should be applied. end note]

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

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

<w:pPr>  
 <w:spacing w:line="480" w:lineRule="auto"/>  
 <w:ind w:left="1440"/>  
 </w:pPr>

<w:rPr>  
 <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>  
 <w:color w:val="22B14C"/>  
 </w:rPr>  
</w:style>

<w:style w:type="character" w:styleId="TestLinkedStyleChar">  
 <w:name w:val="Test Linked Style Char"/>  
 <w:link w:val="TestLinkedStyle"/>  
 <w:rsid w:val="009C1646"/>  
 <w:rPr>  
 <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>  
 <w:color w:val="22B14C"/>  
 </w:rPr>  
</w:style>

The link element in the paragraph style specifies TestLinkedStyleChar, the styleId of the paired character style, and the link element in the character style specifies TestLinkedStyle, the styleId of the paired paragraph style, creating a linked style combination. end example]

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

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

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

<w:pPr>  
 <w:pStyle w:val="TestLinkedStyle"/>  
 <w:spacing w:line="480" w:lineRule="auto"/>  
 <w:ind w:left="1440"/>  
 </w:pPr>

<w:rPr>  
 <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>  
 <w:color w:val="22B14C"/>  
 </w:rPr>  
</w:style>

<w:style w:type="character" w:styleId="TestLinkedStyleChar">  
 <w:name w:val="Test Linked Style Char"/>  
 <w:link w:val="TestLinkedStyle"/>  
 <w:rsid w:val="009C1646"/>  
 <w:rPr>  
 <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>  
 <w:color w:val="22B14C"/>  
 </w:rPr>  
</w:style>

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

<w:p>  
 <w:pPr>  
 <w:pStyle w:val="TestLinkedStyle"/>  
 </w:pPr>  
 <w:r>  
 <w:t xml:space="preserve">A para version of Test Linked Style.</w:t>  
 </w:r>  
</w:p>

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

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

<w:p>  
 <w:r>  
 <w:t xml:space="preserve">Next run is character version of </w:t>  
 </w:r>

<w:r>  
 <w:rPr>  
 <w:rStyle w:val="TestLinkedStyleChar"/>  
 </w:rPr>  
 <w:t>Test Linked Style</w:t>  
 </w:r>

<w:r>  
 <w:t>.</w:t>  
 </w:r>  
</w:p>

The rStyle element in the second run’s properties links the run with the character version of the linked style definition. end example]

### Numbering Styles

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

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

[Example: Consider a numbering style “Test Numbering Style”:

<w:style w:type="numbering" w:styleId="TestNumberingStyle">

<w:name w:val="Test Numbering Style" />

<w:priority w:val="99" />

<w:rsid w:val="0045009F" />

<w:pPr>

<w:numPr>

<w:numId w:val="1" />

</w:numPr>

</w:pPr>

</w:style>

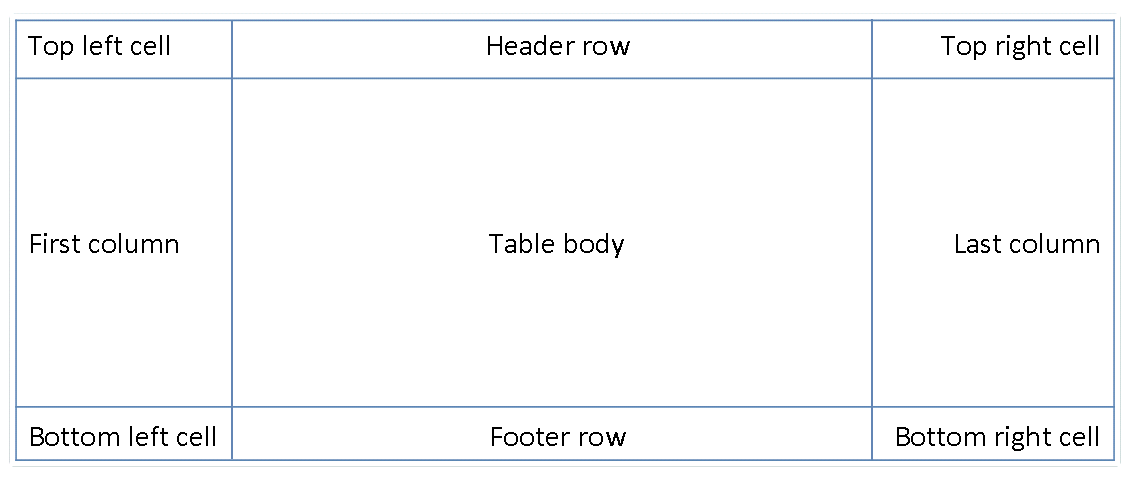
The only information specified in the numbering style definition is a reference to the numbering definition for the numbering information which is defined by this numbering style. end example]

### Table Styles

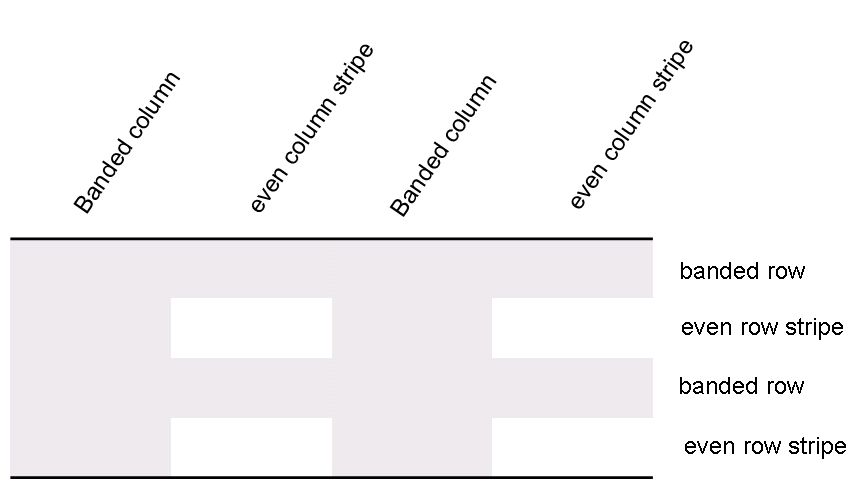
The last conventional type of style definition, table styles are styles which apply to the contents of zero or more tables within a document. This definition implies that the style can only define table properties (properties which apply to the table and its constituent rows and cells), however a table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

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.

These table conditional formats are applied to different regions of the table as follows:



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



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

* Whole table
* Banded columns, even column banding
* Banded rows, even row banding
* First row, last row
* First column, last column
* Top left, top right, bottom left, bottom right

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

<w:style w:type="table" w:styleId="TestTableStyle">  
 <w:name w:val="Test Table Style"/>  
 <w:basedOn w:val="TableNormal"/>  
 <w:priority w:val="99"/>  
 <w:rsid w:val="00340CC4"/>

<w:tblPr>  
 <w:tblBorders>  
 <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:left w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:insideH w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 <w:insideV w:val="single" w:sz="4" w:space="0" w:color="auto"/>  
 </w:tblBorders>

<w:tblCellMar>  
 <w:top w:w="0" w:type="dxa"/>  
 <w:left w:w="108" w:type="dxa"/>  
 <w:bottom w:w="0" w:type="dxa"/>  
 <w:right w:w="108" w:type="dxa"/>  
 </w:tblCellMar>  
 </w:tblPr>

<w:tblStylePr w:type="firstRow">  
 <w:rPr>  
 <w:b/>  
 </w:rPr>  
 <w:tcPr>  
 <w:shd w:val="clear" w:color="auto" w:fill="ED1C24"/>  
 </w:tcPr>  
 </w:tblStylePr>  
</w:style>

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

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

* First row
* Last row
* First column
* Last column
* Row banding
* Column banding

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

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

<w:tbl>  
 <w:tblPr>  
 <w:tblStyle w:val="Style2"/>  
 <w:tblW w:w="0" w:type="auto"/>  
 <w:tblLook w:val="0660"/>  
 </w:tblPr>  
 …  
</w:tbl>

…

<w:tbl>  
 <w:tblPr>  
 <w:tblStyle w:val="Style2"/>  
 <w:tblW w:w="0" w:type="auto"/>  
 <w:tblLook w:val="0460"/>  
 </w:tblPr>  
 …  
</w:tbl>

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

### Default Document Paragraph and Character Properties

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

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

### Style Inheritance

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

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

[Example: The “Tristan Test” paragraph style can inherit properties from the “Heading 1” paragraph style, which itself can inherit properties from the “Normal” paragraph style. end example]

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

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

<w:style w:type="character" w:styleId="Green">  
 <w:name w:val="Green" />  
 <w:basedOn w:val="Base" />  
 <w:rPr>  
 <w:color w:val="22B14C" />  
 </w:rPr>  
</w:style>

…

[../Local Settings/Temp/styles.xml](../Local%20Settings/Temp/styles.xml)<w:style w:type="character" w:styleId="Base">  
 <w:name w:val="Base" />   
 <w:rPr>  
 <w:rFonts w:ascii="Arial" w:hAnsi="Arial" />   
 <w:b />   
 </w:rPr>  
</w:style>

The definition of the Green character style has a basedOn element which specifies the Base style. This means that any use of the Green style is defined as bold, green, Arial text. end example]

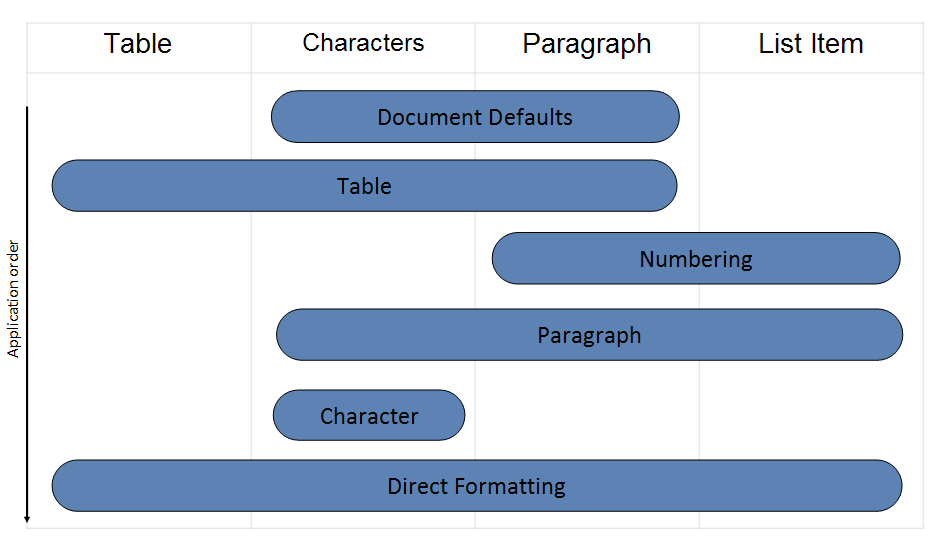
Conversely, a producer should not output any property on a style which has already been set by a previous level of the style hierarchy, as well as those which match the document defaults. This means that if the document defaults or any previous level in a style’s hierarchy specify a property which is unchanged at this level, that property should not be part of the style definition in the resulting WordprocessingML. Adding a property at multiple levels in the style hierarchy is not invalid, but unnecessarily duplicative as the setting is already applied to the text, resulting in an unnecessary increase to file size.

[Example: If the document default font is Bauhaus 93 and the Heading 1 style also specifies the Bauhaus 93 font, then a producer should not output any rFonts element for the Heading 1 style definition, because that formatting is inherited from the document defaults. end example]

### Style Application

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

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



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

### Latent Styles

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

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

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

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

[Example: Consider the following latent style information stored in a document’s styles part:

<w:latentStyles w:defLockedState="off" w:defPriority="99"  
 w:defSemiHidden="on" w:defUnhideWhenUsed="on" w:defQFormat="off"  
 w:count="180">

<w:lsdException w:name="Normal" w:unhideWhenUsed="off"  
 w:qformat="on"/>  
 <w:lsdException w:name="heading 1" w:semiHidden="off" w:priority="1"/>  
 <w:lsdException w:name="heading 2" w:priority="1"   
 w:unhideWhenUsed="on"/>  
 <w:lsdException w:name="heading 3" w:semiHidden="off/>  
 <w:lsdException w:name="heading 4" w:priority="1" w:qformat="on"/>  
 <w:lsdException w:name="heading 5" w:priority="1" w:qformat="on"/>

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

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

## Numbering

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

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

1. This is a paragraph with numbering information.

* This is also a paragraph with numbering information.

### 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.

### 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.

### 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>

<w:lvl w:ilvl="1" w:tplc="04090019">  
 …  
 </w:lvl>

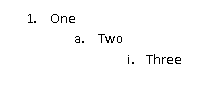
<w:lvl w:ilvl="2" w:tplc="04090019">  
 …  
 </w:lvl>

<w:lvl w:ilvl="3" w:tplc="0409000F">  
 …  
 </w:lvl>  
 …  
</w:abstractNum>

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

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

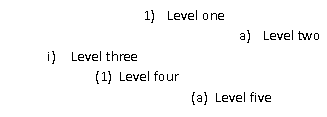
Consider three numbered paragraphs that reference the same numbering definition, but each, in turn, reference a different level within that list:



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

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

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



In this example, the properties of each level of the numbering definition is such that the paragraphs for each level are indented arbitrarily. However, this is still a completely valid numbering definition, and the paragraphs each represent subsequent levels of the same numbering definition.

Within an abstract numbering level definition, each numbering level is represented by an lvl element that defines a single level of numbering information. Numbering levels specify the following properties: starting number value, a number format presentation code (e.g. 1 vs. the string literal One), an associated paragraph style, which previous level should cause this numbering level to restart, the numbering text, number justification, a paragraph properties indentation for this level, etc. The complete definition of all numbering level properties can be found on the reference for the lvl element.

Consider the following numbering level definition in WordprocessingML:

<w:lvl w:ilvl="1">  
 <w:start w:val="4"/>  
 <w:nfc w:val="3"/>  
 <w:pStyle w:val="Heading1"/>  
 <w:lvlText w:val="BEFORE %2 AFTER %1 END"/>  
 <w:lvlJc w:val="left"/>

<w:pPr>  
 <w:tabs>  
 <w:tab w:val="num" w:pos="2880"/>  
 </w:tabs>  
 <w:ind w:left="288" w:firstLine="1152"/>  
 </w:pPr>  
</w:lvl>

This particular numbering level defines the following information:

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

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

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

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

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

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

1  
 BEFORE 1 AFTER 1 END  
 BEFORE 2 AFTER 1 END  
 BEFORE 3 AFTER 1 END

2  
 BEFORE 1 AFTER 2 END  
 BEFORE 2 AFTER 2 END  
 …

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

### Numbering Definition Instances

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

* An abstract numbering reference
* (Optional) level overrides

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

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

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

<w:numbering>  
 ...  
 <w:num w:numId="2">  
 <w:abstractNumId w:val="0" />  
 </w:num>

<w:num w:numId="3">  
 <w:abstractNumId w:val="1" />  
 </w:num>

<w:num w:numId="4">  
 <w:abstractNumId w:val="4" />  
 </w:num>

<w:num w:numId="5">  
 <w:abstractNumId w:val="4" />  
 </w:num>  
</w:numbering>

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

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

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

<w:num w:numId="6">  
 <w:abstractNumId w:val="4" />  
 <w:lvlOverride w:ilvl="0">  
 <w:lvl w:ilvl="0">  
 <w:start w:val="4" />  
 <w:lvlText w:val="%1)" />  
 <w:lvlJc w:val="left" />  
 <w:pPr>  
 <w:ind w:left="360" w:hanging="360" />  
 </w:pPr>  
 </w:lvl>  
 </w:lvlOverride>  
</w:num>

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

### Applying Numbering to Paragraphs

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

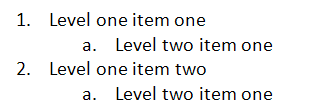
To accomplish this, numbered paragraphs are identified by the numPr element within the paragraph's properties element (the pPr element). The numbering properties within a paragraph are specified using two specific elements that specify the numbering definition information to use:

* A numbering definition instance reference
* A numbering level reference

The numbering definition instance reference is specified using the numId element. This element contains a reference to the numId attribute in a specific numbering definition instance within the numbering part, which links this paragraph to that numbering definition instance.

The numbering level reference is specified using the ilvl element. This element contains a reference to the ilvl attribute in the specified numbering definition instance's level information, which specifies the numbering level within the referenced numbering definition instance to be used by this numbered paragraph.

Consider the following numbered paragraphs in a WordprocessingML document:



These four numbered paragraphs, all referencing the same numbering definition, produce the following WordprocessingML:

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="0" />  
 <w:numId w:val="5" />  
 </w:numPr>  
 </w:pPr>  
 <w:r>  
 <w:t>Level one item one</w:t>  
 </w:r>  
</w:p>

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="1" />  
 <w:numId w:val="5" />  
 </w:numPr>  
 </w:pPr>  
 <w:r>  
 <w:t>Level two item one</w:t>  
 </w:r>  
</w:p>

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="0" />  
 <w:numId w:val="5" />  
 </w:numPr>  
 </w:pPr>

<w:r>  
 <w:t>Level one item two</w:t>  
 </w:r>  
</w:p>

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="1" />  
 <w:numId w:val="5" />  
 </w:numPr>  
 </w:pPr>

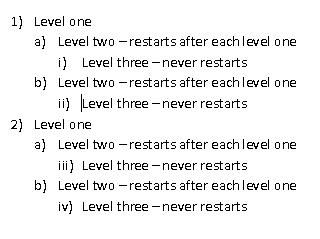
<w:r>  
 <w:t>Level two item one</w:t>  
 </w:r>  
</w:p>

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

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

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

* Level 1 is set to restart after each level 0 (lvlRestart of 1)
* Level 2 is set to never restart (lvlRestart of 0)



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

### The Complete Story

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

Consider the following WordprocessingML for a numbered paragraph:

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="0" />  
 <w:numId w:val="5" />  
 </w:numPr>  
 </w:pPr>

<w:r>  
 <w:t>Numbered paragraph</w:t>  
 </w:r>  
</w:p>

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

<w:numbering>  
 ...  
 <w:num w:numId="5">  
 <w:abstractNumId w:val="4" />   
 </w:num>  
</w:numbering>

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

<w:numbering>  
 <w:abstractNum w:abstractNumId="4">  
 <w:nsid w:val="FFFFFF7F" />  
 <w:multiLevelType w:val="singleLevel" />  
 <w:lvl w:ilvl="0">  
 <w:start w:val="1" />  
 <w:lvlText w:val="%1." />  
 <w:lvlJc w:val="left" />

<w:pPr>  
 <w:tabs>  
 <w:tab w:val="num" w:pos="720" />  
 </w:tabs>  
 <w:ind w:left="720" w:hanging="360" />  
 </w:pPr>  
 </w:lvl>  
 </w:abstractNum>  
 …  
</w:numbering>

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

### Numbering Styles

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

Unlike paragraph and character styles, numbering styles are never directly referenced by content in the document—instead, an abstract numbering definition specifies that it contains the underlying numbering information for a numbering style, and one or more numbering definition instances reference that abstract numbering definition.

### Referencing Numbering Styles

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

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

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

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

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

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="0" />  
 <w:numId w:val="6" />  
 </w:numPr>  
 </w:pPr>  
 <w:r>  
 <w:t>This paragraph references a numbering style via numStyleLink.</w:t>  
 </w:r>  
</w:p>

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

<w:num w:numId="6">  
 <w:abstractNumId w:val="0" />  
</w:num>

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

<w:abstractNum w:abstractNumId="0">  
 <w:nsid w:val="38901FA4" />  
 <w:multiLevelType w:val="multilevel" />  
 <w:numStyleLink w:val="TestNumberingStyle" />  
</w:abstractNum>

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

<w:style w:type="numbering" w:styleId="TestNumberingStyle">  
 <w:name w:val="Test Numbering Style" />  
 <w:uiPriority w:val="99" />  
 <w:rsid w:val="00DB3C4B" />  
 <w:pPr>  
 <w:numPr>  
 <w:numId w:val="4" />  
 </w:numPr>  
 </w:pPr>  
</w:style>

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

<w:num w:numId="4">  
 <w:abstractNumId w:val="2" />  
</w:num>

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

<w:abstractNum w:abstractNumId="2">  
 <w:nsid w:val="46364EB7" />  
 <w:multiLevelType w:val="multilevel" />  
 <w:styleLink w:val="TestNumberingStyle" />  
 <w:lvl w:ilvl="0">  
 <w:lvlText w:val="%1 %1 %1" />  
 <w:lvlJc w:val="left" />  
 <w:pPr>  
 <w:tabs>  
 <w:tab w:val="num" w:pos="360" />  
 </w:tabs>  
 <w:ind w:left="0" w:firstLine="0" />  
 </w:pPr>  
 </w:lvl>  
 …  
</w:abstractNum>

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

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

<w:p>  
 <w:pPr>  
 <w:numPr>  
 <w:ilvl w:val="0" />  
 <w:numId w:val="4" />  
 </w:numPr>  
 </w:pPr>

<w:r>  
 <w:t>This paragraph references a numbering style via styleLink.</w:t>  
 </w:r>  
</w:p>

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

<w:num w:numId="4">  
 <w:abstractNumId w:val="2" />  
</w:num>

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

<w:abstractNum w:abstractNumId="2">  
 <w:nsid w:val="46364EB7" />  
 <w:multiLevelType w:val="multilevel" />  
 <w:styleLink w:val="TestNumberingStyle" />  
 <w:lvl w:ilvl="0">  
 <w:lvlText w:val="%1 %1 %1" />  
 <w:lvlJc w:val="left" />

<w:pPr>  
 <w:tabs>  
 <w:tab w:val="num" w:pos="360" />  
 </w:tabs>  
 <w:ind w:left="0" w:firstLine="0" />  
 </w:pPr>  
 </w:lvl>  
 …  
</w:abstractNum>

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

## Fonts

### Font References

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

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

<w:r>  
 <w:rPr>  
 <w:rFonts w:ascii="Arial Black" w:hAnsi="Arial Black" />  
 </w:rPr>  
 <w:t>This run of text uses the Arial Black font face.</w:t>  
</w:r>

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

### Font Reference Types

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

* ASCII font
* High ANSI font
* East Asian font
* Complex Script font

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

The ASCII font formats all characters in the ASCII range (character values 0–127). This font is specified using the ascii attribute on the rFonts element.

The East Asian font formats all characters that belong to Unicode sub ranges for East Asian languages. This font is specified using the eastAsia attribute on the rFonts element.

The complex script font formats all characters that belong to Unicode sub ranges for complex script languages. This font is specified using the cs attribute on the rFonts element.

The high ANSI font formats all characters that belong to Unicode sub ranges other than those explicitly included by one of the groups above. This font is specified using the hAnsi attribute on the rFonts element.

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

<w:r>  
 <w:rPr>  
 <w:rFonts w:ascii="Arial Black" w:hAnsi="Arial Black" w:cs="Arial"  
 w:eastAsia="SimSun"/>  
 </w:rPr>  
 …  
</w:r>

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

* Complex script characters used the Arial font
* East Asian characters used the SimSun font
* All other characters used the Arial Black font

### Ambiguous Characters

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

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

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

### Font Table

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

* Applications to perform substitution with the most appropriate possible font when the desired font face is not available on the system. Since some fonts are commercially distributed, it is possible for a document to be formatted with one or more fonts that are not available depending on the machine opening the current system. This information allows the application that cannot locate the desired font to perform the most appropriate possible match.
* Embedding of fonts in the document to prevent the need for font substitution

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

### Font Substitution Data

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

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

<w:font w:name="Arial Black">  
 <w:panose1 w:val="020B0A04020102020204" />  
 <w:charset w:val="00" />  
 <w:family w:val="swiss" />  
 <w:pitch w:val="variable" />  
 <w:sig w:usb0="00000287" w:usb1="00000000" w:usb2="00000000"  
 w:usb3="00000000" w:csb0="0000009F" w:csb1="00000000" />  
</w:font>

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

* The font's Panose-1 number
* The character set of the font
* The font's family
* The font's pitch
* The code pages and Unicode sub ranges supported by the font

### Font Embedding

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

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

* Regular
* Bold
* Italic
* Bold + Italic

Each form is obfuscated using the mechanism described in the reference material on this subject.

### Theme Fonts

As well as storing standard font face information, run properties may store an abstraction for font face information known as theme fonts. Theme fonts are values that specify that the font face information for a run is not stored in the attribute value using the appropriate font face name, but is rather a reference into the document's theme part, allowing font face information to be stored and managed centrally as part of the 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 reference to the formatting that is stored centrally in another part.

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

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

<w:r>  
 <w:rPr>  
 <w:rFonts w:asciiTheme="minorHAnsi" w:hAnsiTheme="minorHAnsi" />  
 </w:rPr>  
 …  
</w:r>

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

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

## Headers and Footers

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

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



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

### Header Part

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

### Footer Part

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

### Headers and Footers

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

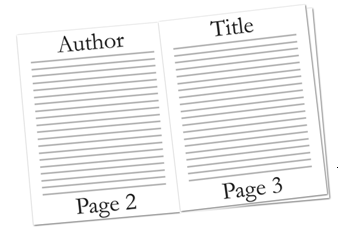
The hdr element defines a single header for the document, while the ftr element defines a single footer for the document. Headers and footers are just another document story in WordprocessingML. Within the root element of the header or footer, the content of the element is similar to the content of the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

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

* First page header/footer
* Odd page header/footer
* Even page header/footer

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

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



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

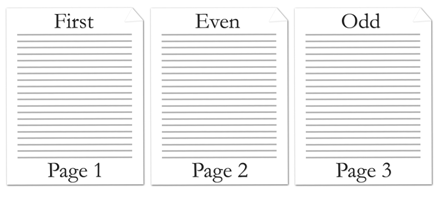


This document defines one header with the text Header. The header's content is stored in a unique Header part. The resulting header is represented by the following WordprocessingML:

<w:hdr>  
 <w:p>  
 <w:r>  
 <w:t>Header</w:t>  
 </w:r>  
 </w:p>  
</w:hdr>

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

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



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

First page header part:

<w:hdr>  
 <w:p>  
 <w:r>  
 <w:t>First</w:t>  
 </w:r>  
 </w:p>  
</w:hdr>

Even page header part:

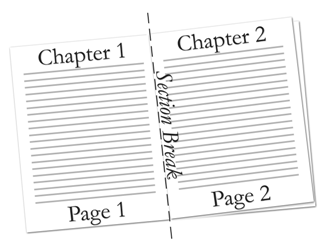
<w:hdr>  
 <w:p>  
 <w:r>  
 <w:t>Even</w:t>  
 </w:r>  
 </w:p>  
</w:hdr>

Odd page header part:

<w:hdr>  
 <w:p>  
 <w:r>  
 <w:t>Odd</w:t>  
 </w:r>  
 </w:p>  
</w:hdr>

### 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:

<w:body>  
 …  
 <w:p>  
 <w:pPr>  
 <w:sectPr>  
 <w:headerReference r:id="rId6" />  
 …  
 </w:sectPr>  
 </w:pPr>  
 …  
 </w:p>

…

<w:sectPr>  
 …  
 </w:sectPr>  
</w:body>

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

### Empty Header or Footer

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

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

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

## Footnotes and Endnotes

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

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

Both footnotes and endnotes consist of two parts:

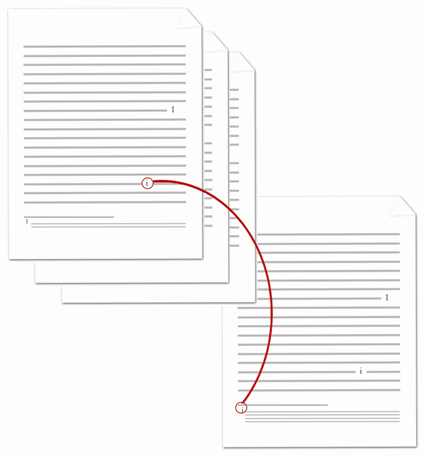
* A note reference mark in the body text to indicate that additional information is in a footnote or endnote, with a numbering system used for each to tell readers whether to look for the note at the end of the page or the end of the document or section.
* The actual footnote or endnote story content.

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



The note reference mark follows the noted text and specifies that there is associated footnote information; the footnote itself is at the bottom of the current page.

Consider the following example of an endnote applied to text in a document:



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

### Footnote Part

Footnote information in a WordprocessingML document is stored in the footnotes 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/footnotes and has a content type of vnd-openxmlformats.officedocument.wordprocessingml-footnotes+xml.

### Endnote Part

Endnote information in a WordprocessingML document is stored in the Endnotes 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/endnotes and has a content type of vnd-openxmlformats.officedocument.wordprocessingml-endnotes+xml.

### Footnotes and Endnotes

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

Consider three different types of footnotes, each identified by a footnote element, defined in the Footnotes part:

<w:footnotes ...>  
 <w:footnote w:type="separator" w:id="0">  
 …  
 </w:footnote>

<w:footnote w:type="continuationSeparator" w:id="1">  
 …  
 </w:footnote>

<w:footnote w:id="2">  
 …  
 </w:footnote>  
</w:footnotes>

Similarly consider three different types of endnotes, each identified by an endnote element, defined in the Endnotes part:

<w:endnotes ...>  
 <w:endnote w:type="separator" w:id="0">  
 …  
 </w:endnote>

<w:endnote w:type="continuationSeparator" w:id="1">  
 …  
 </w:endnote>

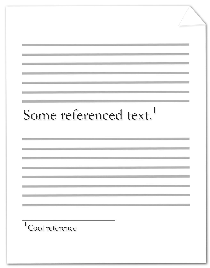
<w:endnote w:id="2">  
 …  
 </w:endnote>  
</w:endnotes>

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

### Footnote and Endnote Types

There are four different types of footnotes and endnotes: normal, separator, continuation separator, and continuation notice. Normal footnotes (or endnotes) contain the text of any footnote (or endnote) in the document. Separator footnotes (or endnotes) define the separator used to separate the footnote (or endnote) from the document text. Continuation separator footnotes (or endnotes) 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 footnotes (or endnotes) 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.



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:

<w:footnote w:id="2">  
 <w:p>  
 <w:pPr>  
 <w:pStyle w:val="FootnoteText" />  
 </w:pPr>

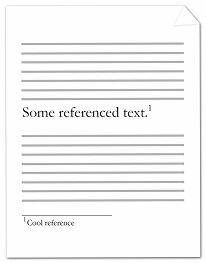
<w:r>  
 <w:rPr>  
 <w:rStyle w:val="FootnoteReference" />  
 </w:rPr>  
 <w:footnoteRef />  
 </w:r>

<w:r>  
 <w:t>Cool reference</w:t>  
 </w:r>  
 </w:p>  
</w:footnote>

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

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

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



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

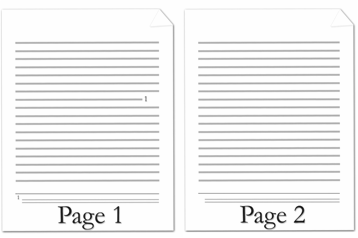
<w:footnote w:type="separator" w:id="0">  
 <w:p>  
 <w:pPr>  
 <w:spacing w:after="0" w:line="240" w:lineRule="auto" />  
 </w:pPr>

<w:r>  
 <w:separator />  
 </w:r>  
 </w:p>  
</w:footnote>

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

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

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



The line separating the document text from the footnote that is continued on the next page is represented by the following WordprocessingML:

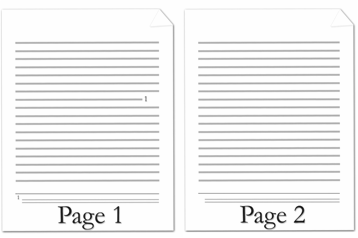
<w:footnote w:type="continuationSeparator" w:id="1">  
 <w:p >  
 <w:pPr>  
 <w:spacing w:after="0" w:line="240" w:lineRule="auto" />  
 </w:pPr>

<w:r>  
 <w:continuationSeparator />  
 </w:r>  
 </w:p>  
</w:footnote>

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

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

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



The continuation notice text is at the bottom of the footnote indicating that the footnote is continued to the next page and is represented by the following WordprocessingML:

<w:footnote w:type="continuationNotice" w:id="3">  
 <w:p >  
 <w:pPr>  
 <w:spacing w:after="0" w:line="240" w:lineRule="auto" />  
 </w:pPr>

<w:r>  
 <w:t>Continued</w:t>  
 </w:r>  
 </w:p>

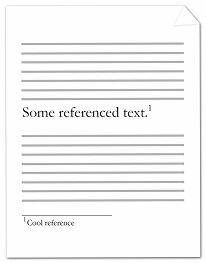
<w:p>  
 <w:pPr>  
 <w:spacing w:after="0" w:line="240" w:lineRule="auto" />  
 </w:pPr>  
 </w:p>  
</w:footnote>

In this example, the footnote has a unique ID of 3. The text that shows up after the footnote text is Continued.

### Footnote and Endnote Reference

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

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



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

<w:p>  
 <w:r>  
 <w:t>Some referenced text</w:t>  
 </w:r>

<w:r>  
 <w:rPr>  
 <w:rStyle w:val="FootnoteReference" />  
 </w:rPr>  
 <w:footnoteReference w:id="2" />  
 </w:r>

<w:r>  
 <w:t>Cool reference</w:t>  
 </w:r>  
</w:p>

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

## Glossary Document

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

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

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

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

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

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

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

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

[Example: Consider the following definition for the contents of a glossary document part within a WordprocessingML document:

<w:glossaryDocument>  
 <w:docParts>  
 <w:docPart>  
 <w:docPartPr>  
 …  
 </w:docPartPr>

<w:docPartBody>  
 <w:p>  
 <w:r>  
 <w:t>Sample entry.</w:t>  
 </w:r>  
 </w:p>  
 </w:docPartBody>  
 </w:docPart>

<w:docPart>  
 …  
 </w:docPart>  
 </w:docParts>  
</w:glossaryDocument>

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

Each glossary document entry consists of two components:

* The entry's properties, specified using the docPartPr element
* The entry's contents, specified using the docPartBody element

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

## Annotations

### Introduction

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

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

* Inline
* Cross-Structure
* Properties

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

### Inline Annotations

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

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

<w:p>  
 <w:r>  
 <w:t xml:space="preserve">The quick brown fox jumps over the </w:t>  
 </w:r>

<w:del … >  
 <w:r>  
 <w:delText>lazy</w:delText>  
 </w:r>  
 </w:del>

<w:ins … >  
 <w:r>  
 <w:t>jet lagged</w:t>  
 </w:r>  
 </w:ins>

<w:r>  
 <w:t xml:space="preserve"> dog.</w:t>  
 </w:r>  
</w:p>

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

### Cross-Structure Annotations

A cross-structure annotation is an annotation that can span portions of WordprocessingML markup. (Cross-structure annotations may span parts of multiple paragraphs, one half of a custom XML markup element's contents, and so on.) In these cases, the annotation's region is delimited by two elements, a start element and an end element, which mark the start and end points of the annotated content, respectively, but do not contain it. Matching start and end markers have the same id attribute value.

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:

<w:p>  
 <w:r>  
 <w:t>Example</w:t>  
 </w:r>  
 <w:bookmarkStart w:id="0" w:name="sampleBookmark" />  
 <w:r>  
 <w:t xml:space="preserve"> text.</w:t>  
 </w:r>  
</w:p>

<w:p>  
 <w:r>  
 <w:t>Example</w:t>  
 </w:r>  
 <w:bookmarkEnd w:id="0" />  
 <w:r>  
 <w:t xml:space="preserve"> text.</w:t>  
 </w:r>  
</w:p>

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

### Property Annotations

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

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

<w:p>  
 <w:r>  
 <w:rPr>  
 <w:b/>  
 <w:rPrChange … >  
 <w:rPr/>  
 </w:rPrChange>  
 </w:rPr>  
 <w:t>Example</w:t>  
 </w:r>

<w:r>  
 <w:t xml:space="preserve"> text.</w:t>  
 </w:r>  
</w:p>

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

### Comments

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

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

* The comment anchor (the text to which the comment applies)
* The comment content (the contents of the comment)

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

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



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

<w:p>  
 <w:r>  
 <w:t xml:space="preserve">Some </w:t>  
 </w:r>

<w:commentRangeStart w:id="0" />  
 <w:r>  
 <w:t>text.</w:t>  
 </w:r>  
 <w:commentRangeEnd w:id="0" />

<w:r>  
 <w:commentReference w:id="0" />  
 </w:r>  
</w:p>

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

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

<w:comment w:id="0" w:author="Joe Smith"  
 w:date="2006-04-06T13:50:00Z" w:initials="User">

<w:p>

<w:pPr>

<w:pStyle w:val="CommentText" />

</w:pPr>

<w:r>

<w:rPr>

<w:rStyle w:val="CommentReference" />

</w:rPr>

<w:annotationRef />

</w:r>

<w:r>

<w:t>comment</w:t>

</w:r>

</w:p>

</w:comment>

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

### Comments Part

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

### Revisions

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

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

A revision consists of two required pieces of information:

* The revision type (specified via the name of the revision element)
* A unique revision identifier (used to uniquely identify revisions)

As well as optional information:

* The author of the revision
* The date and time of the revision

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



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

<w:p>

<w:r>

<w:t>Some</w:t>

</w:r>

<w:ins w:id="0" w:author="Joe Smith" w:date="2006-03-31T12:50:00Z">

<w:r>

<w:t>text</w:t>

</w:r>

</w:ins>

</w:p>

The ins element contains both the required information: all of the content which shall be treated as revision marked as inserted (the word text); a unique revision identifier of 0.

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

Within a WordprocessingML document, the following types of revisions can be used to track the changes to a document:

* Insertions
* Deletions
* Moves
* Changes to run/paragraph/table/numbering/section properties
* Changes to custom XML markup

### 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:

<w:p>  
 <w:r>  
 <w:t>Example</w:t>  
 </w:r>  
 <w:bookmarkStart w:id="0" w:name="sampleBookmark" />  
 <w:r>  
 <w:t xml:space="preserve"> text.</w:t>  
 </w:r>  
</w:p>

<w:p>  
 <w:r>  
 <w:t>Example</w:t>  
 </w:r>  
 <w:bookmarkEnd w:id="0" />  
 <w:r>  
 <w:t xml:space="preserve"> text.</w:t>  
 </w:r>  
</w:p>

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

### Range Permissions

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

Like bookmarks, range permissions are a legacy word-processing function that predates the concepts of XML and well-formedness, so 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 a single paragraph, where a range permission has been added spanning the words range permission:

<w:p>  
 <w:r>  
 <w:t xml:space="preserve">This is a </w:t>  
 </w:r>

<w:permStart w:id="0" w:edGrp="everyone"/>  
 <w:r>  
 <w:t>range permission</w:t>  
 </w:r>  
 <w:permEnd w:id="0"/>

<w:r>  
 <w:t>.</w:t>  
 </w:r>  
</w:p>

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

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

### Spelling and Grammar

A spelling and grammar error is an annotation used to specify the locations of an existing spelling and/or grammatical error within the contents of a document.

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

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

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

<w:p>  
 <w:proofErr w:val="spellStart"/>  
 <w:r>  
 <w:t>erqwt</w:t>  
 </w:r>  
 <w:proofErr w:val="spellEnd"/>  
 <w:r>  
 <w:t xml:space="preserve"> werewr</w:t>  
 </w:r>  
</w:p>

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

## 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.

### 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:



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



### Compatibility Settings

A compatibility setting is an optional setting used to mimic behavior of documents created in earlier word-processing applications. It is recommended that new WordprocessingML documents contain no compatibility settings. If compatibility settings are needed, they are stored in the Document Settings part (§2.15.1).

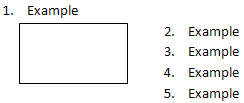
Consider a document in which the compatibility setting ww11IndentRules is applied. As a compatibility setting, this element specifies an indentation behavior to be applied throughout the given document to preserve visual fidelity with an earlier word processing application. Specifically, if the indentation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

<w:compat>

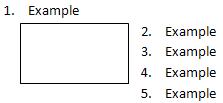
<w:ww11IndentRules />

</w:compat>

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



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



### Web Settings

A web setting is a setting used to specify a document-level property that is applicable when saving a web page as a WordprocessingML document, or when saving a WordprocessingML document as a webpage. Thus, if a given WordprocessingML document was not created from a web page, and will never become a web page, no web settings are needed within the document. If they are needed, web settings are stored in the Web Settings part.

Consider a document in which the web setting allowPNG is applied. As a web setting this element specifies if the PNG graphics format will be used for persisting images when saving the document as a web page. Specifically, if the PNG graphics format will be used when saving a document as a web page, this requirement would be specified by adding the following WordprocessingML to the settings part:

<w:webSettings>

<w:allowPNG />

</w:webSettings>

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

## Fields and Hyperlinks

### Fields

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

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

A field is a set of codes that instructs a WordprocessingML consumer to insert text, graphics, page numbers, and other material into a document automatically. [Example: The DATE field causes the current date to be inserted. end example] The text or graphics inserted into a document when a consumer carries out a field's codes is referred to as the field 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 any field is updated is outside the scope of this standard.

### Hyperlinks

Yet to be supplied

## Mail Merge

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

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

### Mail Merge, WordprocessingML, and Hosting Applications

The two key parts of the mail merge process are:

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

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

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

* creating a new document for each merged document
* generating and sending emails containing merged document
* printing merged documents

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

### Connecting Documents to an External Data Source

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

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

Consider a document containing static WordprocessingML constructs such as paragraphs in addition to two 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, |

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

<w:mailMerge>  
 …  
 <w:dataType w:val="database" />  
 <w:query w:val="SELECT \* FROM Table1" />  
 <w:dataSource r:id="rId1" />  
 …  
</w:mailMerge>

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

### Populating Merged Documents with External Data

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

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

<w:fieldMapData>  
 <w:type w:val="dbColumn" />   
 <w:name w:val="Customer Title" />   
 <w:mappedName w:val="Courtesy Title" />   
 <w:column w:val="9" />   
</w:fieldMapData>

<w:fieldMapData>  
 <w:type w:val="dbColumn" />   
 <w:name w:val="Customer Last Name" />   
 <w:mappedName w:val="Last Name" />   
 <w:column w:val="10" />   
</w:fieldMapData>

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

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

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

The mail merge process will then run through the specified external database and populate the mail merge fields with the data from the Customer Title and Customer Last Name columns in the specified database, and generate two of merged documents containing the specified external data as well as the static 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 |
| 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.  Sincerely, | Dear Mr. Doe:  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, | Dear Ms. Smith:  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, |

## Miscellaneous Topics

### Text Boxes

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

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

### Subdocuments

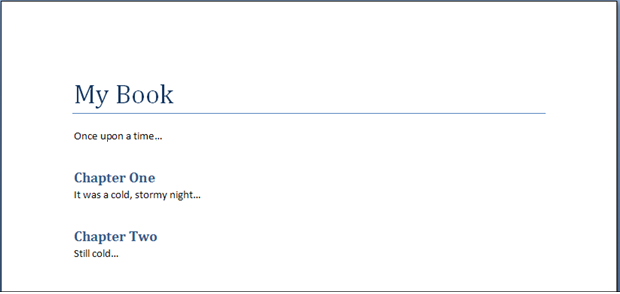
Within a WordprocessingML document, it is sometimes necessary to break a large document into two or more separate WordprocessingML document files, allowing each of these files to be distributed, edited, and handled independently.

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

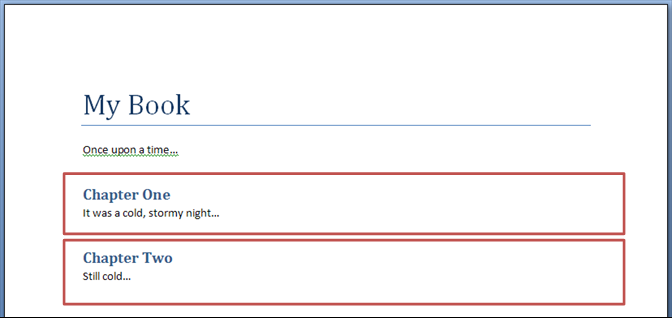
When a WordprocessingML document is comprised of other WordprocessingML documents in this way, the resulting documents are a master document and its subdocuments.

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

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



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



The result is three WordprocessingML documents:

* 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)

### Importing External Content

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 included because it was stored in a format other than the WordprocessingML format defined by this Standard.

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

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

<html … >  
 <body style="margin-left:200px;margin-top:50px">  
 <p>Paragraph one.</p>  
 <blockquote style="border:5px solid #00FFFF">Paragraph in a blockquote.</blockquote>  
 <p>Paragraph two.</p>  
 </body>  
</html>

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

### Roundtripping Alternate Content

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

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

<ve:AlternateContent xmlns:ve="…">  
 <ve:Choice Requires="colors" xmlns:colors="urn:randomTextColors">  
 <w:p>  
 <w:pPr>  
 <colors:random colors:val="true" />  
 </w:pPr>  
 <w:r>  
 <w:t>Random colors!</w:t>  
 </w:r>  
 </w:p>  
 </ve:Choice>  
 <ve:Fallback>  
 <w:p>  
 <w:r>  
 <w:rPr>  
 <w:color w:val="FF0000" />  
 </w:rPr>  
 <w:t>R</w:t>  
 </w:r>  
 <w:r>  
 <w:rPr>  
 <w:color w:val="00FF00" />  
 </w:rPr>  
 <w:t>a</w:t>  
 </w:r>  
 …  
 </w:p>  
 </ve:Fallback>  
</ve:AlternateContent>

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

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

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

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

<w:fsb>  
 <ve:AlternateContent xmlns:ve="…">  
 <ve:Choice Requires="colors" xmlns:colors="urn:randomTextColors">  
 …  
 </ve:Choice>  
 <ve:Fallback>  
 …  
 </ve:Fallback>  
 </ve:AlternateContent>  
</w:fsb>

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

<w:p>  
 <w:r>  
 <w:rPr>  
 <w:color w:val="FF0000" />  
 </w:rPr>  
 <w:t>R</w:t>  
 </w:r>  
 <w:r>  
 <w:rPr>  
 <w:color w:val="00FF00" />  
 </w:rPr>  
 <w:t>a</w:t>  
 </w:r>  
 …  
</w:p>

End of informative text.

# Introduction to SpreadsheetML

This clause is informative.

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

## Workbook

### Overview

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

### Minimum Workbook Scenario

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

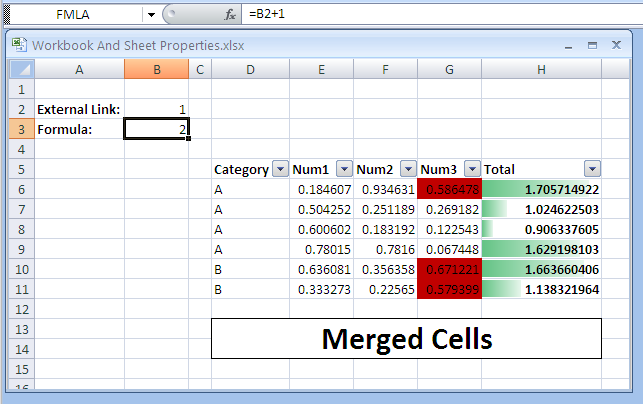
* A single sheet
* A sheet ID
* A relationship Id that points to the location of the sheet definition

For example:

<workbook>  
 <sheets>  
 <sheet name="Sheet1" sheetId="1" r:id="rId1"/>  
 </sheets>  
</workbook>

### Example Workbook Properties

Consider the following graphical representation of a workbook:



The above example will have the following workbook properties definition:

<workbook>  
 <fileVersion lastEdited="4" lowestEdited="4" rupBuild="3814"/>  
 <workbookPr backupFile="1" saveExternalLinkValues="0" updateLinks="never"/>  
 <calcPr calcId="122211" calcMode="manual" iterate="1"/>  
 <bookViews>  
 <workbookView showHorizontalScroll="0" showVerticalScroll="0"  
 showSheetTabs="0" xWindow="45" yWindow="15" windowWidth="9420"  
 windowHeight="5460" tabRatio="701"/>  
 </bookViews>

<sheets>  
 <sheet name="Sheet1" sheetId="1" sh:id="rId1"/>  
 <sheet name="Sheet2" sheetId="2" sh:id="rId2"/>  
 <sheet name="Sheet3" sheetId="3" sh:id="rId3"/>  
 </sheets>  
</workbook>

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

### fileVersion

This contains file versioning properties.

#### lastEdited

This represents the version of the application that last saved the file.

#### lowestEdited

This represents 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

This represents an incremental public release of the application (e.g., RTM version or SP1 version).

#### workbookPr

This is a group of various workbook properties.

#### backupFile

This flag indicates whether the application should create a backup of the file in question during a save operation.

#### saveExternalLinkValues

This flag 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

This flag 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

This specifies 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

This flag indicates when the application should calculate formulas:

* Manual means to wait for the user to initate 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.

### workbookView

A single view definition.

#### showHorizontalScroll

This flag controls visibility of the horizontal scroll bar of the application. In the example above, it is set to not being visible.

#### showVerticalScroll

This flag controls visibility of the vertical scroll bar of the application. In the example above, it is set to not being visible.

#### showSheetTabs

This flag controls visibility of the worksheet tabs in the application. In the example above, they are set to not being visible.

#### xWindow

This value specifies the x coordinate (in twips) of the upper right corner of the workbook window.

#### yWindow

This value specifies the y coordinate (in twips) of the upper right corner of the workbook window.

#### windowWidth

This value specifies the width of the workbook window.

#### windowHeight

This value specifies the height of the workbook window.

#### tabRatio

This value specifies the ratio between the workbook tabs bar and the horizontal scroll bar.

#### sheets

A collection of worksheets in the workbook.

#### sheet

A single sheet definition (book-level).

#### name

The name of the worksheet. These must be unique within the workbook.

#### sheetId

The internal Id of the sheet. These must be unique within the workbook.

#### id

The relationship Id that points to the sheet part definition.

## Sheets

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

Other types of sheets include chart sheets and dialog sheets.

### Minimum Worksheet Scenario

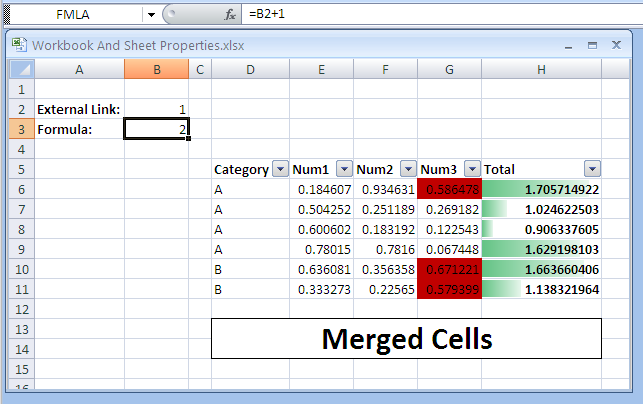
The smallest possible (blank) sheet is as follows:

<worksheet>  
 <sheetData/>  
</worksheet>

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

### Example Sheet

Consider the following graphical representation of a worksheet:



=‘C:\[ExternalBook.xlsx]Sheet1’!$A$1

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

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

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

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

### Sheet Properties

<worksheet>  
 <sheetPr filterMode="1"/>  
 <dimension ref="A2:H14"/>

<sheetViews>  
 <sheetView tabSelected="1" workbookViewId="0">  
 <selection activeCell="B3" sqref="B3"/>  
 </sheetView>  
 </sheetViews>

<sheetFormatPr defaultRowHeight="15"/>

<cols>  
 <col min="1" max="1" width="12.85546875" bestFit="1" customWidth="1"/>  
 <col min="3" max="3" width="3.28515625" customWidth="1"/>  
 <col min="4" max="4" width="11.140625" bestFit="1" customWidth="1"/>  
 <col min="8" max="8" width="17.140625" style="1" customWidth="1"/>  
 </cols>

### Sheet Data

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

<sheetData>  
 <row r="2" spans="1:2" customFormat="1">  
 <c r="A2" s="1" t="s">  
 <v>0</v>  
 </c>

<c r="B2">  
 <f>[1]Sheet1!$A$1</f>  
 <v>1</v>  
 </c>  
 </row>

<row r="3" spans="1:8" customFormat="1">  
 <c r="A3" s="1" t="s">  
 <v>1</v>  
 </c>

<c r="B3">  
 <f>B2+1</f>  
 <v>2</v>  
 </c>

<c r="H3" s="1"/>  
 </row>

<row r="4" spans="1:8">  
 <c r="H4"/>  
 </row>

<row r="5" spans="4:8">  
 <c r="D5" s="1" t="s">  
 <v>4</v>  
 </c>

<c r="E5" s="1" t="s">  
 <v>5</v>  
 </c>

<c r="F5" s="1" t="s">  
 <v>6</v>  
 </c>

<c r="G5" s="1" t="s">  
 <v>7</v>  
 </c>

<c r="H5" s="1" t="s">  
 <v>8</v>  
 </c>  
 </row>

<row r="6" spans="4:8">  
 <c r="D6" t="s">  
 <v>2</v>  
 </c>

<c r="E6">  
 <v>0.18460660235998017</v>  
 </c>

<c r="F6">  
 <v>0.93463071023892952</v>  
 </c>

<c r="G6">  
 <v>0.58647760893211043</v>  
 </c>

<c r="H6" s="1">  
 <f ce="1">SUM(E6:G6)</f>  
 <v>1.7057149215310201</v>  
 </c>  
 </row>

<row r="7" spans="4:8">  
 <c r="D7" t="s">  
 <v>2</v>  
 </c>

<c r="E7">  
 <v>0.50425224796279555</v>  
 </c>

<c r="F7">  
 <v>0.25118866081991786</v>  
 </c>

<c r="G7">  
 <v>0.26918159410869791</v>  
 </c>

<c r="H7" s="1">  
 <f t="shared" ref="H7:H11" ce="1" si="0">SUM(E7:G7)</f>  
 <v>1.0246225028914113</v>  
 </c>  
 </row>

<row r="8" spans="4:8">  
 <c r="D8" t="s">  
 <v>2</v>  
 </c>

<c r="E8">  
 <v>0.6006019062877066</v>  
 </c>

<c r="F8">  
 <v>0.18319235857964333</v>  
 </c>

<c r="G8">  
 <v>0.12254334000604317</v>  
 </c>

<c r="H8" s="1">  
 <f t="shared" ce="1" si="0">SUM(E8:G8)</f>  
 <v>0.9063376048733931</v>  
 </c>  
 </row>

<row r="9" spans="4:8" hidden="1">  
 <c r="D9" t="s">  
 <v>2</v>  
 </c>

<c r="E9">  
 <v>0.78015011938458589</v>  
 </c>

<c r="F9">  
 <v>0.78159963723670689</v>  
 </c>

<c r="G9">  
 <v>6.7448346870105036E-2</v>  
 </c>

<c r="H9" s="1">  
 <f t="shared" ce="1" si="0">SUM(E9:G9)</f>  
 <v>1.6291981034913978</v>  
 </c>  
 </row>

<row r="10" spans="4:8" hidden="1">  
 <c r="D10" t="s">  
 <v>3</v>  
 </c>

<c r="E10">  
 <v>0.63608141933645479</v>  
 </c>

<c r="F10">  
 <v>0.35635845012920608</v>  
 </c>

<c r="G10">  
 <v>0.67122053637107193</v>  
 </c>

<c r="H10" s="1">  
 <f t="shared" ce="1" si="0">SUM(E10:G10)</f>  
 <v>1.6636604058367328</v>  
 </c>  
 </row>

<row r="11" spans="4:8" hidden="1">  
 <c r="D11" t="s">  
 <v>3</v>  
 </c>

<c r="E11">  
 <v>0.33327331908137214</v>  
 </c>

<c r="F11">  
 <v>0.2256497329592122</v>  
 </c>

<c r="G11">  
 <v>0.5793989116090501</v>  
 </c>

<c r="H11" s="1">  
 <f t="shared" ce="1" si="0">SUM(E11:G11)</f>  
 <v>1.1383219636496344</v>  
 </c>  
 </row>

<row r="13" spans="4:8">  
 <c r="D13" s="2" t="s">  
 <v>9</v>  
 </c>

<c r="E13" s="3"/>  
 <c r="F13" s="3"/>  
 <c r="G13" s="3"/>  
 <c r="H13" s="4"/>  
 </row>

<row r="14" spans="4:8">  
 <c r="D14" s="5"/>  
 <c r="E14" s="6"/>  
 <c r="F14" s="6"/>  
 <c r="G14" s="6"/>  
 <c r="H14" s="7"/>  
 </row>  
</sheetData>

### Supporting Features

The supporting feature definitions follow the cell table data:

<sheetProtection objects="0" scenarios="0"/>

<autoFilter ref="D5:H11">  
 <filterColumn colId="0">  
 <filters>  
 <filter val="A"/>  
 </filters>  
 </filterColumn>

<filterColumn colId="1">  
 <customFilters and="1">  
 <customFilter operator="greaterThan" val="0"/>  
 <customFilter operator="lessThan" val="0.7"/>  
 </customFilters>  
 </filterColumn>  
</autoFilter>

<mergeCells>  
 <mergeCell ref="D13:H14"/>  
</mergeCells>

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

<dataBar minLength="10" maxLength="90" showValue="1">  
 <cfvo type="min" val="0"/>  
 <cfvo type="max" val="0"/>  
 <color rgb="FF63C384"/>  
 </dataBar>  
 </cfRule>  
</conditionalFormatting>

<conditionalFormatting sqref="G6:G11">  
 <cfRule type="cellIs" dxfId="0" priority="1" stopIfTrue="0"  
 operator="greaterThan">  
 <formula>0.5</formula>  
 </cfRule>  
</conditionalFormatting>

<printOptions/>  
<pageMargins left="0.7" right="0.7" top="0.75" bottom="0.75"  
 header="0.3" footer="0.3"/>  
<pageSetup orientation="portrait" horizontalDpi="300" verticalDpi="300"/>  
<headerFooter/>

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

### Sheet Properties

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

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

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

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

### sheetData Cell Table

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

### Row

<row r="2" spans="1:2" customFormat="1">  
 <c r="A2" s="1" t="s">  
 <v>0</v>  
 </c>

<c r="B2">  
 <f>[1]Sheet1!$A$1</f>  
 <v>1</v>  
 </c>  
</row>

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

### Cell

<c r="B3">  
 <f>B2+1</f>  
 <v>2</v>  
</c>

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

#### Cell Values

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

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

#### Formulas

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

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

##### Shared Formulas

<row r="7" spans="4:8">  
 <c r="H7" s="1">  
 <f t="shared" ref="H7:H11" ce="1" si="0">SUM(E7:G7)</f>  
 <v>1.0246225028914113</v>  
 </c>  
</row>

<row r="8" spans="4:8">  
 <c r="H8" s="1">  
 <f t="shared" ce="1" si="0">SUM(E8:G8)</f>  
 <v>0.9063376048733931</v>  
 </c>  
</row>

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

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

##### External Referencing Formulas

<c r="B2">  
 <f>[1]Sheet1!$A$1</f>  
 <v>1</v>  
</c>

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

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

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

### Supporting Sheet Features

### Defined Names

<definedNames>  
 <definedName name="FMLA">Sheet1!$B$3</definedName>  
 <definedName name="SheetLevelName" comment="This name is scoped to Sheet1"  
 localSheetId="0">Sheet1!$B$3</definedName>  
</definedNames>

Defined names can be used in place of cell references in formulas. For example, instead of using B3+1 to add 1 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 perform the calculation.

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

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

### AutoFilter

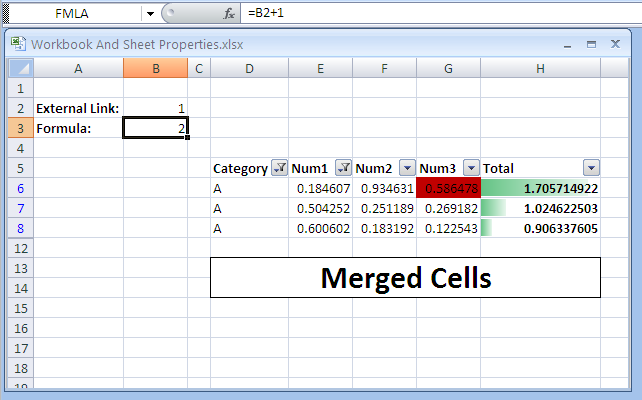
<autoFilter ref="D5:H11">  
 <filterColumn colId="0">  
 <filters>  
 <filter val="A"/>  
 </filters>  
 </filterColumn>

<filterColumn colId="1">  
 <customFilters and="1">  
 <customFilter operator="greaterThan" val="0"/>  
 <customFilter operator="lessThan" val="0.7"/>  
 </customFilters>  
 </filterColumn>  
</autoFilter>

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

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.

The resulting grid could be rendered like this:



### Merged Cells

<mergeCells>  
 <mergeCell ref="D13:H14"/>  
</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.

### Conditional Formatting

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

<dataBar minLength="10" maxLength="90" showValue="1">  
 <cfvo type="min" val="0"/>  
 <cfvo type="max" val="0"/>  
 <color rgb="FF63C384"/>  
 </dataBar>  
 </cfRule>  
</conditionalFormatting>

<conditionalFormatting sqref="G6:G11">  
 <cfRule type="cellIs" dxfId="0" priority="1" stopIfTrue="0" operator="greaterThan">  
 <formula>0.5</formula>  
 </cfRule>  
</conditionalFormatting>

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

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

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

## Comments

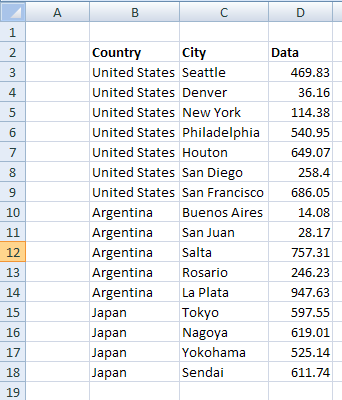
Yet to be supplied

## Shared String Table

### Overview

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

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



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

### File Architecture

There is a single shared strings part for all the strings in a workbook. This part is related to the workbook. Each cell (in sheet1.xml, for example) containing a string value refers by index to a string expressed in the shared strings part. The solid arrows represent relationships among the parts and the dotted arrows represent references by index to a string in the shared strings part.

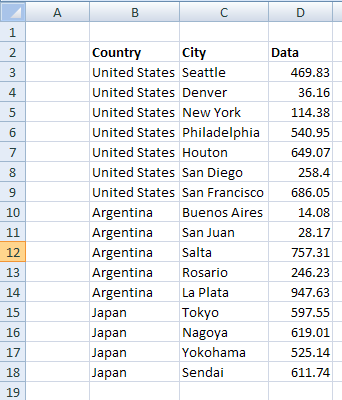
### Example: Plain Text

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

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

### Illustration

Consider the example in the introduction:



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

### The XML

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

<sst … count="35" uniqueCount="22">  
 <si>  
 <t>United States</t>  
 </si>

<si>  
 <t>Seattle</t>  
 </si>

<si>  
 <t>Denver</t>  
 </si>

<si>  
 <t>New York</t>  
 </si>

<si>  
 <t>Philadelphia</t>  
 </si>

<si>  
 <t>Houton</t>  
 </si>

<si>  
 <t>San Diego</t>  
 </si>

<si>  
 <t>San Francisco</t>  
 </si>

<si>  
 <t>Argentina</t>  
 </si>

<si>  
 <t>Buenos Aires</t>  
 </si>

<si>  
 <t>San Juan</t>  
 </si>

<si>  
 <t>Salta</t>  
 </si>

<si>  
 <t>Rosario</t>  
 </si>

<si>  
 <t>La Plata</t>  
 </si>

<si>  
 <t>Japan</t>  
 </si>

<si>  
 <t>Tokyo</t>  
 </si>

<si>  
 <t>Nagoya</t>  
 </si>

<si>  
 <t>Yokohama</t>  
 </si>

<si>  
 <t>Sendai</t>  
 </si>

<si>  
 <t>Country</t>  
 </si>

<si>  
 <t>City</t>  
 </si>

<si>  
 <t>Data</t>  
 </si>  
</sst>

The cell table for this example looks like this:

<sheetData>  
 <row r="2" spans="2:4" customFormat="1">  
 <c r="B2" s="1" t="s">  
 <v>19</v>  
 </c>

<c r="C2" s="1" t="s">  
 <v>20</v>  
 </c>

<c r="D2" s="1" t="s">  
 <v>21</v>  
 </c>  
 </row>

<row r="3" spans="2:4" customFormat="1">  
 <c r="B3" t="s">  
 <v>0</v>  
 </c>

<c r="C3" t="s">  
 <v>1</v>  
 </c>

<c r="D3">  
 <f t="shared" ref="D3:D18" ca="1" si="0">ROUND(RAND()\*1000,2)</f>  
 <v>374.9</v>  
 </c>  
 </row>

<row r="4" spans="2:4" customFormat="1">  
 <c r="B4" t="s">  
 <v>0</v>  
 </c>

<c r="C4" t="s">  
 <v>2</v>  
 </c>

<c r="D4">  
 <f t="shared" ca="1" si="0"/>  
 <v>452.82</v>  
 </c>  
 </row>

<row r="5" spans="2:4" customFormat="1">  
 <c r="B5" t="s">  
 <v>0</v>  
 </c>

<c r="C5" t="s">  
 <v>3</v>  
 </c>

<c r="D5">  
 <f t="shared" ca="1" si="0"/>  
 <v>632.1</v>  
 </c>  
 </row>

<row r="6" spans="2:4" customFormat="1">  
 <c r="B6" t="s">  
 <v>0</v>  
 </c>

<c r="C6" t="s">  
 <v>4</v>  
 </c>

<c r="D6">  
 <f t="shared" ca="1" si="0"/>  
 <v>886.37</v>  
 </c>  
 </row>

<row r="7" spans="2:4" customFormat="1">  
 <c r="B7" t="s">  
 <v>0</v>  
 </c>

<c r="C7" t="s">  
 <v>5</v>  
 </c>

<c r="D7">  
 <f t="shared" ca="1" si="0"/>  
 <v>291.14</v>  
 </c>  
 </row>

<row r="8" spans="2:4" customFormat="1">  
 <c r="B8" t="s">  
 <v>0</v>  
 </c>

<c r="C8" t="s">  
 <v>6</v>  
 </c>

<c r="D8">  
 <f t="shared" ca="1" si="0"/>  
 <v>114.97</v>  
 </c>  
 </row>

<row r="9" spans="2:4" customFormat="1">  
 <c r="B9" t="s">  
 <v>0</v>  
 </c>

<c r="C9" t="s">  
 <v>7</v>  
 </c>

<c r="D9">  
 <f t="shared" ca="1" si="0"/>  
 <v>291.99</v>  
 </c>  
 </row>

<row r="10" spans="2:4" customFormat="1">  
 <c r="B10" t="s">  
 <v>8</v>  
 </c>

<c r="C10" t="s">  
 <v>9</v>  
 </c>

<c r="D10">  
 <f t="shared" ca="1" si="0"/>  
 <v>335.42</v>  
 </c>  
 </row>

<row r="11" spans="2:4" customFormat="1">  
 <c r="B11" t="s">  
 <v>8</v>  
 </c>

<c r="C11" t="s">  
 <v>10</v>  
 </c>

<c r="D11">  
 <f t="shared" ca="1" si="0"/>  
 <v>664.72</v>  
 </c>  
 </row>

<row r="12" spans="2:4" customFormat="1">  
 <c r="B12" t="s">  
 <v>8</v>  
 </c>

<c r="C12" t="s">  
 <v>11</v>  
 </c>

<c r="D12">  
 <f t="shared" ca="1" si="0"/>  
 <v>992.62</v>  
 </c>  
 </row>

<row r="13" spans="2:4" customFormat="1">  
 <c r="B13" t="s">  
 <v>8</v>  
 </c>

<c r="C13" t="s">  
 <v>12</v>  
 </c>

<c r="D13">  
 <f t="shared" ca="1" si="0"/>  
 <v>148.5</v>  
 </c>  
 </row>

<row r="14" spans="2:4" customFormat="1">  
 <c r="B14" t="s">  
 <v>8</v>  
 </c>

<c r="C14" t="s">  
 <v>13</v>  
 </c>

<c r="D14">  
 <f t="shared" ca="1" si="0"/>  
 <v>193.53</v>  
 </c>  
 </row>

<row r="15" spans="2:4" customFormat="1">  
 <c r="B15" t="s">  
 <v>14</v>  
 </c>

<c r="C15" t="s">  
 <v>15</v>  
 </c>

<c r="D15">  
 <f t="shared" ca="1" si="0"/>  
 <v>849.36</v>  
 </c>  
 </row>

<row r="16" spans="2:4" customFormat="1">  
 <c r="B16" t="s">  
 <v>14</v>  
 </c>

<c r="C16" t="s">  
 <v>16</v>  
 </c>

<c r="D16">  
 <f t="shared" ca="1" si="0"/>  
 <v>765.46</v>  
 </c>  
 </row>

<row r="17" spans="2:4" customFormat="1">  
 <c r="B17" t="s">  
 <v>14</v>  
 </c>

<c r="C17" t="s">  
 <v>17</v>  
 </c>

<c r="D17">  
 <f t="shared" ca="1" si="0"/>  
 <v>350.26</v>  
 </c>  
 </row>

<row r="18" spans="2:4" customFormat="1">  
 <c r="B18" t="s">  
 <v>14</v>  
 </c>

<c r="C18" t="s">  
 <v>18</v>  
 </c>

<c r="D18">  
 <f t="shared" ca="1" si="0"/>  
 <v>979.22</v>  
 </c>  
 </row>  
</sheetData>

### Shared String Table

<sst … count="35" uniqueCount="22">  
 <si>  
 <t>United States</t>  
 </si>

<si>  
 <t>Seattle</t>  
 </si>

<si>  
 <t>Denver</t>  
 </si>

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

### Cell Table

<row r="2" spans="2:4" customFormat="1">  
 <c r="B2" s="1" t="s">  
 <v>19</v>  
 </c>

<c r="C2" s="1" t="s">  
 <v>20</v>  
 </c>

<c r="D2" s="1" t="s">  
 <v>21</v>  
 </c>  
</row>

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

<row r="3" spans="2:4" customFormat="1">  
 <c r="B3" t="s">  
 <v>0</v>  
 </c>

<c r="C3" t="s">  
 <v>1</v>  
 </c>

<c r="D3">  
 <f t="shared" ref="D3:D18" ca="1" si="0">ROUND(RAND()\*1000,2)</f>  
 <v>374.9</v>  
 </c>  
</row>

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

<row r="4" spans="2:4" customFormat="1">  
 <c r="B4" t="s">  
 <v>0</v>  
 </c>

<c r="C4" t="s">  
 <v>2</v>  
 </c>

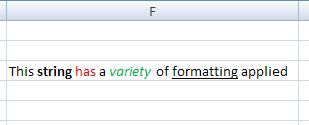
<c r="D4">  
 <f t="shared" ca="1" si="0"/>  
 <v>452.82</v>  
 </c>  
</row>

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

### Example: Rich Text

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

### Illustration



### Shared String Table

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

<si>  
 <r>  
 <t xml:space="preserve">This </t>  
 </r>

<r>  
 <rPr>  
 <b/>  
 <sz val="11"/>  
 <color theme="1"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t xml:space="preserve">string </t>  
 </r>

<r>  
 <rPr>  
 <sz val="11"/>  
 <color rgb="FFFF0000"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t>has</t>  
 </r>

<r>  
 <rPr>  
 <sz val="11"/>  
 <color theme="1"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t xml:space="preserve"> a </t>  
 </r>

<r>  
 <rPr>  
 <i/>  
 <sz val="11"/>  
 <color rgb="FF00B050"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t>variety</t>  
 </r>

<r>  
 <rPr>  
 <sz val="11"/>  
 <color theme="1"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t xml:space="preserve"> of </t>  
 </r>

<r>

<rPr>  
 <u/>  
 <sz val="11"/>  
 <color theme="1"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t>formatting</t>  
 </r>

<r>  
 <rPr>  
 <sz val="11"/>  
 <color theme="1"/>  
 <rFont val="Calibri"/>  
 <family val="2"/>  
 <scheme val="minor"/>  
 </rPr>  
 <t xml:space="preserve"> applied</t>  
 </r>  
</si>

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

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

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

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

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

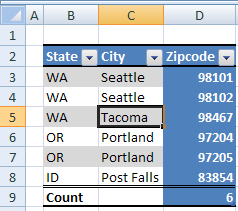
## Formulas

Yet to be supplied

## Tables

### Overview

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



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

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

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

### File Architecture

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

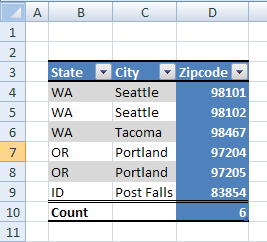
The sheet XML stores the numeric and textual data. The table XML records the various attributes for the particular table object.

### Example: Table

This example demonstrates a table created from data that was previously entered in the sheet. (See §3.17 for a discussion of Tables with XML data bindings.)

### Illustration

Consider the example provided in §3.6.1 above:



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

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

* The column heading area, with bold facing and top and bottom borders
* The data area, with banded striping
* The total row area, with a top double border and bold facing
* The last column area, with a solid background fill of blue.

### The Sheet XML

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

…  
 <tableParts count="1">  
 <tablePart r:id="rId1"/>  
 </tableParts>  
</worksheet>

### 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:

<table xmlns= … id="8" name="Table19" displayName="Table19" ref="B3:D10"  
 totalsRowCount="1">  
 <autoFilter ref="B3:D10"/>  
 <tableColumns count="3">  
 <tableColumn id="1" name="State" totalsRowLabel="Count"  
 totalsRowDxfId="0"/>  
 <tableColumn id="2" name="City"/>  
 <tableColumn id="3" name="Zipcode" totalsRowFunction="count"/>  
 </tableColumns>  
 <tableStyleInfo name="TableStyleMedium16" showFirstColumn="0"  
 showLastColumn="1"  
 showRowStripes="1" showColumnStripes="0"/>  
</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.

## AutoFilters and Sort State Information

Yet to be supplied

## External Connection

### Overview

Many spreadsheet users want to be able to access data from external sources: databases, text files, web pages, XML web services, OLAP cubes. Typically, a spreadsheet application will provide abilities for the user to locate, browse, connect to, and query external data sources. Once the data source has been located, connected to, and queried, the resulting data must be rendered in the spreadsheet application, and made available for further analysis.

Data sources such as databases are made available for browsing and consumption via data-provider technologies. Typically, the data provider provides a standard interface for accessing the data, and removes the complexity introduced due to each database application's providing non-standard data access APIs. In this way, OLEDB providers, for example, can be written for myriad database implementations, and a consumer can always use a single interface (defined by OLEDB) to access these disparate data sources.

A live connection to a data source is established by the application at runtime, and can only exist as a live connection while the application is running. There are two types of information about a particular connection:

* The information used to establish the connection.
* The information and properties about how the connection should be used and how the connection should behave in conjunction with the application.

Information about a connection can be supplied by the user as the connection is being established—for example, providing a password, picking a table, applying a filter, or setting behavioral properties such as whether to refresh the data when the workbook is opened and whether to store refreshed data in the worksheet when the workbook is saved.

Information about a connection can also be persisted in a connection file separately from the workbook file. In this way a directory or file share containing a variety of these connection files can be considered a library of data connections, for example.

Any time a connection is established, whether by using information from a connection file or by gathering the connection information directly from the user, a copy of the connection information is stored in the workbook.

Data providers and connection types discussed below are:

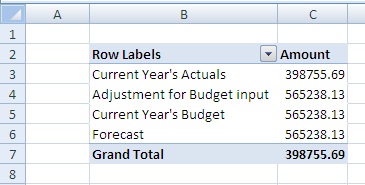
* ODBC
* OLEDB
* ADO
* DAO
* Text Import
* Web

The corresponding features in SpreadsheetML that render and analyze the data are:

* Query Table
* Table
* XML Map
* Pivot Table
* The CUBE\* Functions

### OLAP Connection

Below is a PivotTable that is rendering data from an OLAP source:



### 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.

<pivotCacheDefinition … saveData="0" refreshedBy="Chad Rothschiller"  
 refreshedDate="2006-04-13T16:02:14" backgroundQuery="1"  
 createdVersion="3"  
 refreshedVersion="3" minRefreshableVersion="3" recordCount="0">  
 <cacheSource type="external" connectionId="2"/>  
 <cacheFields count="2">  
 <cacheField name="[Category].[Category Description]"  
 caption="Category   
 Description" numFmtId="0" hierarchy="1" level="1">  
 <sharedItems count="4">  
 <s v="[Category].[All Category].[Current Year's Actuals]"  
 c="Current Year's Actuals"/>

### Connection XML

Looking at the XML in the workbook connections part that describes the connection whose id="2":

<connection id="2" odcFile="  
 c:\…\externalData.odc" keepAlive="1" name="externalData"  
 description="FoodMart 2000 - Budget Planing" type="5"  
 refreshedVersion="3" background="1">

<dbPr connection="Provider=MSOLAP.2;Integrated Security=SSPI;Persist  
 Security Info=True;Data Source=xlextdat8;Initial  
 Catalog=AdamTest;Client  
 Cache Size=25;Auto Synch Period=10000;MDX Compatibility=1"  
 command="Budget" commandType="1"/>  
 <olapPr sendLocale="1" rowDrillCount="1000"/>  
</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 whenever the connection is created so that the connection information can be reused if desired. The connection information is copied from the connection file into the spreadsheet. When a connection cannot be established, the spreadsheet application can check the external connection file to see if a newer definition of the connection is available.
* name indicates the friendly name of the connection. This name must be unique within a workbook.
* keepAlive indicates that the application should hold the connection open once established, instead of closing the connection after retrieving the data.
* type value of 5 indicates that this connection type is OLEDB.
* refreshedVersion indicates the version of the application which last refreshed this connection. 3=Excel 2007; 2=Excel 2003; 1=Excel 2002, 0=Excel 2000 or before.
* background value of 1 indicates that background refresh (asynchronous refresh) is enabled. Note that this is not a guarantee that the connection will be refreshed asynchronously. Certain objects may require a connection to be refreshed either synchronously or asynchronously regardless of this setting.

Attributes on the dbPr element express additional properties on the connection.

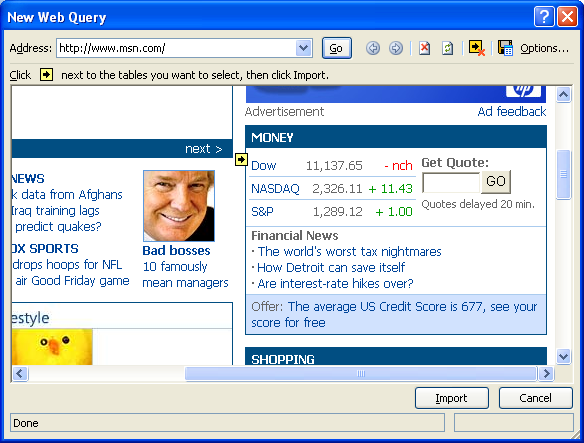
* connection expresses the connection string that is needed to establish a connection to the external data source.
* command can indicate a table name, a cube name, or an SQL expression requesting data.
* commandType indicates what kind of information is found in command: 1 means the value of command is the name of an OLAP cube.

Attributes on the olapPr element express properties that apply to connections to OLAP data sources.

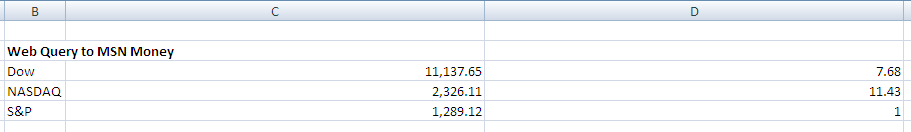
* sendLocale value of 1 indicates that the client application should send its user interface language locale to the OLAP data provider in order to receive back from the server localized OLAP cube member string values.
* rowDrillCount is number of rows to return on a drill through request.

### Web Query

A possible user interface for picking a web source is:



A possible rendering in the spreadsheet grid might be:



### QueryTable XML

The XML expressing the definition of the QueryTable indicates that it is using the connection whose Id value is 1 (connectionId):

<queryTable … name="msn" connectionId="1" autoFormatId="16"  
 applyNumberFormats="0" applyBorderFormats="0" applyFontFormats="1"  
 applyPatternFormats="1" applyAlignmentFormats="0" applyWidthHeightFormats="0"/>

### Connection XML

The workbook connection whose Id is 1 is expressed below.

<connection id="1" name="Connection" type="4" refreshedVersion="3"  
 background="1" saveData="1">  
 <webPr sourceData="1" parsePre="1" consecutive="1" xl2000="1"  
 url="http://www.msn.com" htmlTables="1">  
 <tables count="1">  
 <x v="2"/>  
 </tables>  
 </webPr>  
</connection>

Attributes and elements that have been previously discussed are not discussed here.

* type value of 4 indicates the connection is a web query connection.
* saveData value of 1 indicates that refreshed data will be kept in the sheet when saving the workbook. 0 indicates to remove the data from the workbook when saving the workbook.
* sourceData value of 1indicates to import and parse the XML data rather than consume the web page's HTML definition.
* parsePre value of 1 indicates that text in <PRE> tags is interpreted as tables.
* consecutive value of 1 means consecutive delimiters are treated as one delimiter
* xl2000 value of 1 means the query was refreshed using Excel 2000.
* url is the address indicating where to retrieve data for this query.
* htmlTables true means only import html tables

tables indicates which HTML table to import from the web page.

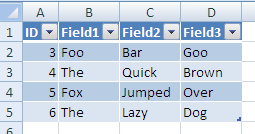
* v value of 2 indicates that the second table is the one to import.

### Unused Connection

A connection can be expressed in a workbook, but not currently used. It remains until deleted by the user explicitly. This simply means that there is no object or feature in the workbook; that is referencing the connection.

### ODBC

A database table imported to the grid, where the data provider is ODBC:



When a table object is used to render external data, it is associated with a QueryTable object to store the properties used when a range is associated with external data. Therefore, the Table object references the QueryTable name, which in turns references connectionId to identify the connection in the workbook connections part.

### Connection XML

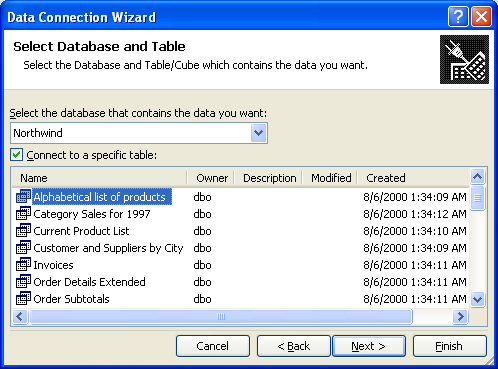
<connection id="4" name="Query from MS Access Database" type="1"  
 refreshedVersion="3" background="1" saveData="1">

<dbPr connection="DSN=MS Access  
 Database;DBQ=E:\…\Database1.accdb;DefaultDir=E:\Documents and   
 Settings\chadroth\Desktop;DriverId=25;FIL=MS   
 Access;MaxBufferSize=2048;PageTimeout=5;"  
 command="SELECT Table1.ID,   
 Table1.Field1, Table1.Field2,  
 Table1.Field3\_x000d\_\_x000a\_FROM   
 `E:\…\Database1.accdb`.Table1 Table1"/>  
</connection>

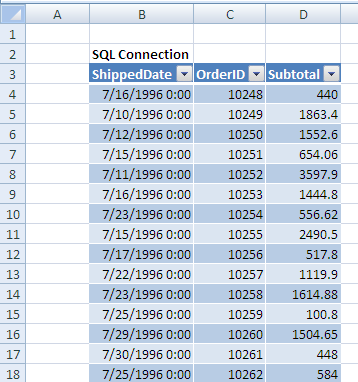
* type value of 1 indicates ODBC connection type.
* command contents are an SQL select statement.

### SQL

An implementation might use a data connection wizard to connect to a SQL table; for example:



The resulting data is rendered in the grid:



In this example, a table object is used to render external data, it is associated with a QueryTable object to store the properties used when a range is associated with external data. Therefore, the Table object references the QueryTable name, which in turns references connectionId to identify the connection in the workbook connections part.

### Connection XML

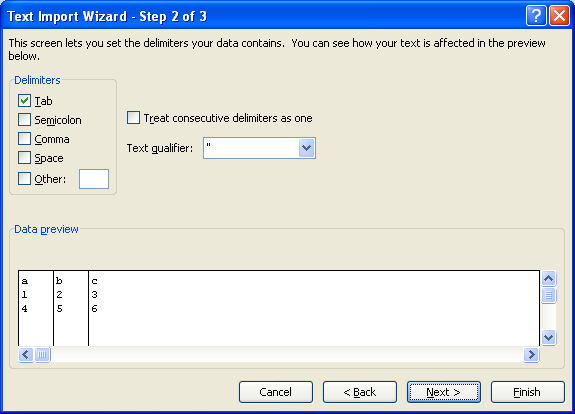
<connection id="6" odcFile="c:\…\xlextdat8 Northwind Summary of Sales by  
 Year.odc" keepAlive="1"  
 name="xlextdat8 Northwind Summary of Sales by Year"   
 type="5" refreshedVersion="3" background="1" saveData="1">

<dbPr connection="Provider=SQLOLEDB.1;Integrated Security=SSPI;  
 Persist Security Info=True;Initial Catalog=Northwind;Data  
 Source=xlextdat8;Use Procedure for Prepare=1;Auto   
 Translate=True;Packet Size=4096;Workstation ID=CHADROTHO12;Use  
 Encryption for Data=False;Tag with column collation when   
 possible=False" command="&quot;Northwind&quot;.&quot;dbo&quot;  
 .&quot;Summary of Sales by Year&quot;" commandType="3"/>  
</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.

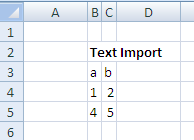
### Text Import

Text Import settings:



Note that there are additional settings not pictured here.

The resulting data in the grid:



The range is associated with a QueryTable object. This query table definition references the connectionId used to retrieve the data.

### Connection XML

<connection id="5" name="Text" type="6" refreshedVersion="3"  
 background="1" saveData="1">…  
 <textPr codePage="437" sourceFile="E:\ …\Text.txt">  
 <textFields count="3">  
 <textField type="text"/>  
 <textField position="5"/>  
 <textField type="skip" position="10"/>  
 </textFields>  
 </textPr>  
</connection>

connection defines the connection

* type value of 6 indicates that this is a text import type of connection.

textPr expresses properties which are specific to text import connections.

* codePage value of 437 indicates that the text file is using the IBM PC (OEM) code page 437 character set.
* sourceFile indicates where the file is located.

textFields expresses information about the particular fields in the text file.

* delimited value of 1 (default) indicates that the text is delimited (variable length). Since this example uses the default value, it is not saved as part of the connection information.
* type indicates the data type (user-specified) of the particular field.
* position indicates the starting position of the field for fixed-width fields.
* thousands specifies the thousands separator character (not in this example, but of enough interest to mention).
* tab, space, comma attributes with values of 1 would flag these characters as delimiters (not in this example, but of enough interest to mention).

## External Links

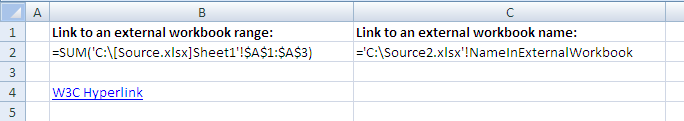
### Overview

An external link is used to link a workbook to other workbook or to external data. The most frequent occurrence for linking a workbook to other workbooks has to do with formulas. In this case, a formula references a range or name defined in another workbook. Hyperlinks on cells and other spreadsheet objects are also considered an external link. OLE links are yet another technology used to link the workbook to another object. Finally, Dynamic Data Exchange (DDE) servers can be used to access external data. DDE servers are accessed through formulas in the workbook.

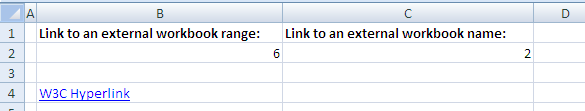
The goal of the way in which external links are saved is to always write the target source in a relationship file, so that external resources are easily discoverable in lightweight relationship XML rather than deep in the application's XML.

### Formula Example

Consider cells B2 and C2 in the following worksheet, Sheet1:



Here, the formulas themselves are displayed in the cells.



Here, the results of the formulas are displayed in the cells.

The formula is expressed in Sheet1's XML, as shown in the following subclause.

### Sheet XML

The corresponding content from Sheet1.xml is:

<worksheet …>  
 <dimension ref="B1:C4"/>  
 <sheetViews>  
 <sheetView tabSelected="1" workbookViewId="0">  
 <selection activeCell="B2" sqref="B2"/>  
 </sheetView>  
 </sheetViews>

<sheetFormatPr defaultRowHeight="15"/>

<cols>  
 <col min="1" max="1" width="1.7109375" customWidth="1"/>  
 </cols>

<sheetData>  
 <row r="1" spans="2:3" customFormat="1" ht="9" customHeight="1"/>  
 <row r="2" spans="2:3" customFormat="1">  
 <c r="B2">  
 <f>SUM([1]Sheet1!$A$1:$A$3)</f>  
 <v>6</v>  
 </c>

<c r="C2">  
 <f>[2]!NameInExternalWorkbook</f>  
 <v>2</v>  
 </c>  
 </row>

<row r="4" spans="2:3" customFormat="1">  
 <c r="B4" s="1" t="s">  
 <v>0</v>  
 </c>  
 </row>  
 </sheetData>

<hyperlinks>  
 <hyperlink ref="B4" r:id="rId1"/>  
 </hyperlinks>

<printOptions/>  
 <pageMargins left="0.7" right="0.7" top="0.75" bottom="0.75"  
 header="0.3" footer="0.3"/>  
 <headerFooter/>  
</worksheet>

#### Cell B2

The formula expressed in cell B2 (cell B2 is the c element whose r="B2") is this:

SUM([1]Sheet1!$A$1:$A$3)

The external reference to another workbook in this case is tokenized to [1]. The value inside the brackets is a 1-based index to the externalReferences collection in the workbook part.

#### Cell C2

The formula expressed in cell C2 (cell C2 is the c element whose r is C2) is this:

[2]!NameInExternalWorkbook

The external reference to another workbook in this case is tokenized to [2]. The value inside the brackets is a 1-based index to the externalReferences collection in the workbook part.

#### Workbook XML

The corresponding content from workbook.xml is

<workbook …>  
 <fileVersion lastEdited="4" lowestEdited="4" rupBuild="4012"/>  
 <workbookPr defaultThemeVersion="123820"/>  
 <bookViews>  
 <workbookView xWindow="360" yWindow="270" windowWidth="18735"  
 windowHeight="11445"/>  
 </bookViews>

<sheets>  
 <sheet name="Sheet1" sheetId="1" r:id="rId1"/>  
 <sheet name="Sheet2" sheetId="2" r:id="rId2"/>  
 <sheet name="Sheet3" sheetId="3" r:id="rId3"/>  
 </sheets>

<externalReferences>  
 <externalReference r:id="rId4"/>  
 <externalReference r:id="rId5"/>  
 </externalReferences>

<calcPr calcId="122211"/>  
 <webPublishing codePage="1252"/>  
</workbook>

The workbook part's externalReferences collection indicates that there are two external workbook references in this workbook. The first supporting external workbook data cache, also stored in this workbook, can be found by following the relationship from the workbook whose Id value is rId4. The second supporting external workbook data cache, also stored in this workbook, can be found by following the relationship from the workbook whose Id value is rId5.

### Workbook Relationships

The corresponding content from workbook.xml.rels is:

<Relationships xmlns="http://…/package/2006/relationships">  
 <Relationship Id="rId8" Type="http://…/sharedStrings" Target="sharedStrings.xml"/>  
 <Relationship Id="rId3" Type="http://…/worksheet" Target="worksheets/sheet3.xml"/>  
 <Relationship Id="rId7" Type="http://…/styles" Target="styles.xml"/>  
 <Relationship Id="rId2" Type="http://…/worksheet" Target="worksheets/sheet2.xml"/>  
 <Relationship Id="rId1" Type="http://…/worksheet" Target="worksheets/sheet1.xml"/>

<Relationship Id="rId6" Type="http://…/theme" Target="theme/theme1.xml"/>  
 <Relationship Id="rId5" Type="http://…/externalLink"  
 Target="externalLinks/externalLink2.xml"/>  
 <Relationship Id="rId4" Type="http://…/externalLink"   
 Target="externalLinks/externalLink1.xml"/>  
 <Relationship Id="rId9" Type="http://…/calcChain" Target="calcChain.xml"/>  
</Relationships>

These relationship expressions indicate that cell B2 is supported by the external workbook data cache located at externalLinks/externalLink1.xml in the package. These relationship expressions also indicate that cell C2 is supported by the external workbook data cache located at externalLinks/externalLink2.xml in the package.

### Supporting Workbook Cache (Cell C2)

The corresponding content from externalLink2.xml is:

<externalLink …>  
 <externalBook xmlns:r="http://schemas.openxmlformats.org  
 /officeDocument/2006/relationships" r:id="rId1">  
 <sheetNames>  
 <sheetName val="Sheet1"/>  
 <sheetName val="Sheet2"/>  
 <sheetName val="Sheet3"/>  
 </sheetNames>

<definedNames>  
 <definedName name="NameInExternalWorkbook"  
 refersTo="='Sheet1'!$B$1"/>  
 </definedNames>

<sheetDataSet>  
 <sheetData sheetId="0">  
 <row r="1">  
 <cell r="B1">  
 <v>2</v>  
 </cell>  
 </row>  
 </sheetData>

<sheetData sheetId="1"/>  
 <sheetData sheetId="2"/>  
 </sheetDataSet>  
 </externalBook>  
</externalLink>

Supporting workbook data caches store the top-level structure of the workbook (sheet names, defined names, cell table). Only the cells referenced are cached. This supporting workbook data cache indicates that the workbook being referenced by C2 has three sheets, whose names are "Sheet1", "Sheet2", and "Sheet3", and has a defined name of "NameInExternalWorkbook". Additionally, the cell table shows that cell B1 in this workbook is the cell being referenced. A copy of the cell table is stored locally, inside the workbook containing the external link.

The r:id="rId1" on the top level externalLink element indicates the Id of the relationship from the externalLink2.xml part, which indicates the location of the actual external workbook.

### External Link (Cell C2)

The corresponding content from externalLink2.xml.rels is

<Relationships …>  
 <Relationship Id="rId1" Type="http://…/externalLinkPath"  
 Target="file:///C:\Source2.xlsx" TargetMode="External"/>  
</Relationships>

This relationship indicates that the supporting workbook that C2 references resides on the local drive, at c:\source2.xlsx.

### Supporting Workbook Cache (Cell B2)

The corresponding content from externalLink1.xml is:

<externalLink …>  
 <externalBook xmlns:r="http://…/relationships" r:id="rId1">  
 <sheetNames>  
 <sheetName val="Sheet1"/>  
 <sheetName val="Sheet2"/>  
 <sheetName val="Sheet3"/>  
 </sheetNames>

<sheetDataSet>  
 <sheetData sheetId="0">  
 <row r="1">  
 <cell r="A1">  
 <v>1</v>  
 </cell>  
 </row>

<row r="2">  
 <cell r="A2">  
 <v>2</v>  
 </cell>  
 </row>

<row r="3">  
 <cell r="A3">  
 <v>3</v>  
 </cell>  
 </row>  
 </sheetData>

<sheetData sheetId="1"/>  
 <sheetData sheetId="2"/>  
 </sheetDataSet>  
 </externalBook>  
</externalLink>

This supporting workbook data cache indicates that the workbook being referenced by B2 has three sheets, whose names are "Sheet1", "Sheet2", and "Sheet3". Additionally, the cell table shows that cells A1, A2, and A3, whose values are 1, 2, and 3, respectively, in this workbook are being referenced. A copy of the cell table is stored locally, inside the workbook containing the external link.

The r:id="rId1" on the top level externalLink element indicates the Id of the relationship from the externalLink1.xml part, which indicates the location of the actual external workbook.

### External Link (Cell B2)

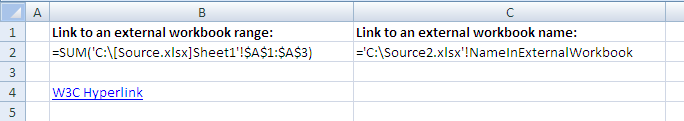
The corresponding content from externalLink1.xml.rels is

<Relationships …>  
 <Relationship Id="rId1" Type="http://…/externalLinkPath"   
 Target="file:///C:\Source.xlsx" TargetMode="External"/>  
</Relationships>

This relationship indicates that the supporting workbook that C2 references resides on the local drive, at c:\source.xlsx.

### Hyperlink Example

Consider the following worksheet:



Cell B4 contains a hyperlink, whose friendly name is "W3C Hyperlink", and whose target is "<http://www.w3.org/>".

### Worksheet XML

See §3.9.3 for the full XML. Here is the snippet expressing the hyperlink information, whose collection appears immediately after the sheetData collection in this example.

<hyperlinks>  
 <hyperlink ref="B4" r:id="rId1"/>  
</hyperlinks>

The hyperlink XML indicates that cell B4 of this sheet has a hyperlink, whose target can be found by following the relationship whose Id="rId1" from the current sheet. The 'friendly' name of the hyperlink is stored in the cell definition.

### Relationship

The corresponding content from sheet1.xml.rels is:

<Relationships …>  
 <Relationship Id="rId1" Type="http://…/hyperlink"  
 Target="<http://www.w3.org/>" TargetMode="External"/>  
</Relationships>

This hyperlink points external to the workbook (TargetMode="External"), and the URL is found in the value of Target to be "<http://www.w3.org/>".

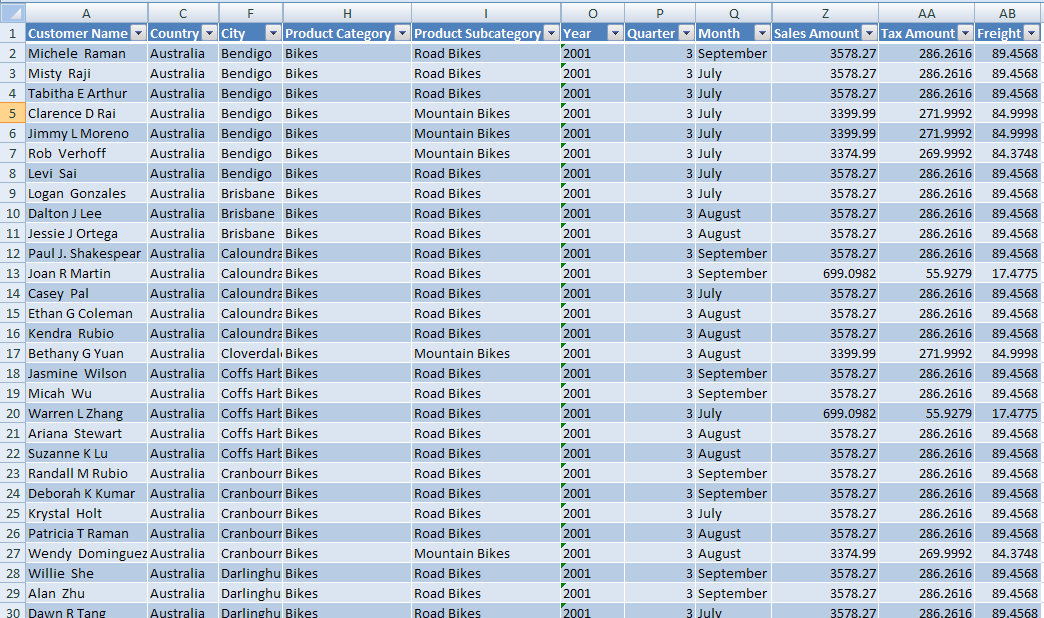
## Pivot Table, Pivot Cache, and Common Types

### Feature Overview

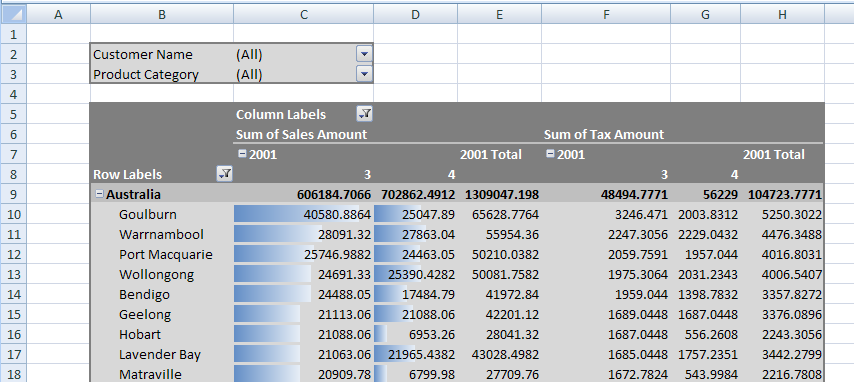
PivotTables display aggregated views of data easily and in an understandable layout. Hundreds or thousands of pieces of underlying information can be aggregated on row & column axes, revealing the meanings behind the data. PivotTable reports are used to organize and summarize your data in different ways. Creating a PivotTable report is about moving pieces of information around to see how they fit together. In a few gestures the pivot rows and columns can be moved into different arrangements and layouts.

A PivotTable object has a row axis area, a column axis area, a values area, and a report filter area. Additionally, PivotTables have a corresponding field list pane displaying all the fields of data which can be placed on one of the PivotTable areas.

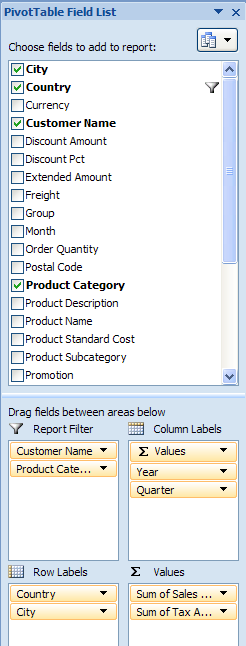
Consider this source data:



This data can be consolidated and summarized in a PivotTable. One way to organize the information would look like this:



Here is the corresponding PivotTable field list:



### File Architecture

The workbook points to (and owns the longevity of) the pivotCacheDefinition part, which in turn points to and owns the pivotCacheRecords part. The workbook also points to and owns the sheet part, which in turn points to and owns a pivotTable part definition, when a PivotTable is on the sheet (there can be multiple PivotTables on a sheet). The pivotTable part points to the appropriate pivotCacheDefinition which it is using. Since multiple PivotTables can use the same cache, the pivotTable part does not own the longevity of the pivotCacheDefinition.

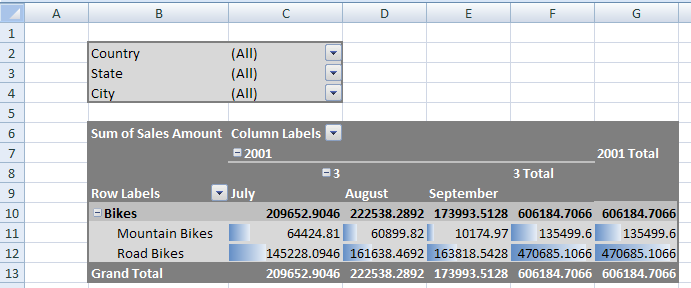
The pivotTable part describes the particulars of the layout of the PivotTable on the sheet. It indicates what fields are on the row axis, the column axis, report filter, and values areas of the PivotTable. It also indicates formatting information about the PivotTable. If conditional formatting has been applied to the PivotTable, that is also expressed in the pivotTable part.

The pivotCacheRecords part contains the underlying data to be aggregated. It is a cache of the source data. The pivotCacheDefinition part defines each field in the pivotCacheRecords part, including field name and information about the data contained in the field. The pivotCacheDefinition part also defines pivot items that are shared among the pivotTable and pivotRecords parts.

### Example - Native with Range Source

#### Illustration

Consider the source data pictured in the overview section. There are 28 fields of data in total (some aren't shown). A corresponding PivotTable summary of the data can look like this:



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.

Row Grand Totals are turned on, and column Subtotals are turned on for Quarter and Year (if there was more than 1 quarter in the source data the Year Subtotal would be more interesting).

#### XML - pivotCacheDefinition part

The pivotCacheDefinition part defines each field in the source data, including the name, the string resources of the instance data (for shared items), and information about the type of data appearing in the field. Note: some of the "Customer Name" and "City" values have been removed to improve readability and reduce length.

<pivotCacheDefinition xmlns:r="…" r:id="rId1" refreshedBy="AnonUser"  
 refreshedDate="2006-05-22T10:07:16" createdVersion="3"  
 refreshedVersion="3" minRefreshableVersion="3" recordCount="182">

<cacheSource type="worksheet">  
 <worksheetSource name="Table1"/>  
 </cacheSource>

<cacheFields count="28">  
 <cacheField name="Customer Name" numFmtId="0">  
 <sharedItems count="7">  
 <s v="Michele Raman"/>  
 <s v="Misty Raji"/>  
 <s v="Tabitha E Arthur"/>  
 <s v="Clarence D Rai"/>  
 <s v="Jimmy L Moreno"/>  
 <s v="Rob Verhoff"/>  
 <s v="Levi Sai"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Group" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Country" numFmtId="0">  
 <sharedItems count="1">  
 <s v="Australia"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Region" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="State" numFmtId="0">  
 <sharedItems count="5">  
 <s v="Victoria"/>  
 <s v="Queensland"/>  
 <s v="South Australia"/>  
 <s v="New South Wales"/>  
 <s v="Tasmania"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="City" numFmtId="0">  
 <sharedItems count="7">  
 <s v="Bendigo"/>  
 <s v="Brisbane"/>  
 <s v="Caloundra"/>  
 <s v="Cloverdale"/>  
 <s v="Coffs Harbour"/>  
 <s v="Cranbourne"/>  
 <s v="Darlinghurst"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Postal Code" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Product Category" numFmtId="0">  
 <sharedItems count="1">  
 <s v="Bikes"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Product Subcategory" numFmtId="0">  
 <sharedItems count="2">  
 <s v="Road Bikes"/>  
 <s v="Mountain Bikes"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Product Name" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Product Description" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Promotion Category" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Promotion" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Promotion Type" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Year" numFmtId="0">  
 <sharedItems count="1">  
 <s v="2001"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Quarter" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" containsInteger="1" minValue="3" maxValue="3"  
 count="1">  
 <n v="3"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Month" numFmtId="0">  
 <sharedItems count="3">  
 <s v="September"/>  
 <s v="July"/>  
 <s v="August"/>  
 </sharedItems>  
 </cacheField>

<cacheField name="Currency" numFmtId="0">  
 <sharedItems/>  
 </cacheField>

<cacheField name="Order Quantity" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" containsInteger="1" minValue="1"  
 maxValue="1"/>  
 </cacheField>

<cacheField name="Unit Price" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" minValue="699.09820000000002"  
 maxValue="3578.27"/>  
 </cacheField>

<cacheField name="Extended Amount" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" minValue="699.09820000000002"  
 maxValue="3578.27"/>  
 </cacheField>

<cacheField name="Discount Pct" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" containsInteger="1" minValue="0"  
 maxValue="0"/>  
 </cacheField>

<cacheField name="Discount Amount" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" containsInteger="1" minValue="0"  
 maxValue="0"/>  
 </cacheField>

<cacheField name="Product Standard Cost" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" minValue="413.1463"   
 maxValue="2171.2941999999998"/>  
 </cacheField>

<cacheField name="Total Product Cost" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" minValue="413.1463"  
 maxValue="2171.2941999999998"/>  
 </cacheField>

<cacheField name="Sales Amount" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" minValue="699.09820000000002"  
 maxValue="3578.27"/>  
 </cacheField>

<cacheField name="Tax Amount" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1"  
 minValue="55.927900000000001" maxValue="286.26159999999999"/>  
 </cacheField>

<cacheField name="Freight" numFmtId="0">  
 <sharedItems containsSemiMixedTypes="0" containsString="0"  
 containsNumber="1" minValue="17.477499999999999"  
 maxValue="89.456800000000001"/>  
 </cacheField>  
 </cacheFields>  
</pivotCacheDefinition>

In the context of pivotCacheDefinition:

* r:id indicates the relationship id pointing to the corresponding pivotCacheRecords part.
* refreshedBy indicates the username of whomever last refreshed the PivotCache.
* refreshedDate indicates when the PivotCache was last refreshed.
* createdVersion indicates the version of Excel which created the PivotCache.
* refreshedVersion indicates the version of Excel which last refreshed the PivotCache.
* minRefreshableVersion indicates the minimum version of Excel required to be able to refresh this PivotCache.

In the context of cacheSource:

* type indicates that data in a worksheet is the source for this PivotCache.
* worksheetSource identifies the particular location of the source data. In this case, it is a named range whose name is "Table1".

In the context of cacheFields, which is a collection of all the field definitions in the source data:

* cacheField indicates the name of the field and provides number format information.

In the context of cacheField:

* sharedItems indicates various flags about the data in this field. Child elements express the values of the shared items.

In the context of sharedItems:

* containsSemiMixedTypes "1" indicates that this field contains text values possibly mixed with other types of values, this can contain blanks. In this example the value is "0".
* containsString value of "1" indicates that this field contains a text value. In this example, the value is "0".
* containsNumber value of "1" indicates that this field contains numeric values.
* containsInteger indicates that this field contains integer values.
* minValue indicates that this field's minimum value is "3".
* maxValue indicates that this field's maximum value is "3".
* s indicates string content for this item value (expressed in v).
* n indicates the numeric content for this item value (expressed in v).

If there are no shared items expressed for a particular field, then the values are expressed directly in the pivotCacheRecords part.

Items in the PivotCacheDefinition can be shared, in order to reduce the redundancy of those values, since they're referenced in multiple places across all the PivotTable parts. For example, a value might be part of a filter, it might appear on a row or column axis, and will appear in the pivotCacheRecords definition as well. However, because of the performance cost of creating the optimized shared items, items are only shared if they are actually in use in the PivotTable. Therefore, depending on user actions on the PivotTable layout, the pivotCacheDefinition and underlying PivotCacheRecords part may be updated.

#### XML - pivotCacheRecords part

This part expresses the underlying source data that the PivotTable is aggregating. (Note that the data has been trimmed down to two records to increase readability.)

<pivotCacheRecords … xmlns:r="…" count="2">  
 <r>  
 <x v="0"/>  
 <s v="Pacific"/>

<x v="0"/>  
 <s v="Australia"/>

<x v="0"/>  
 <x v="0"/>  
 <s v="3550"/>

<x v="0"/>  
 <x v="0"/>  
 <s v="Road-150 Red, 62"/>  
 <s v="This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has a extremely light heat-treated aluminum frame, and steering that allows precision control."/>  
 <s v="No Discount"/>  
 <s v="No Discount"/>  
 <s v="No Discount"/>

<x v="0"/>  
 <x v="0"/>  
 <x v="0"/>  
 <s v="Australian Dollar"/>

<n v="1"/>  
 <n v="3578.27"/>  
 <n v="3578.27"/>  
 <n v="0"/>  
 <n v="0"/>  
 <n v="2171.2941999999998"/>  
 <n v="2171.2941999999998"/>  
 <n v="3578.27"/>  
 <n v="286.26159999999999"/>  
 <n v="89.456800000000001"/>  
 </r>

<r>  
 <x v="1"/>  
 <s v="Pacific"/>

<x v="0"/>  
 <s v="Australia"/>

<x v="0"/>  
 <x v="0"/>  
 <s v="3550"/>

<x v="0"/>  
 <x v="0"/>  
 <s v="Road-150 Red, 44"/>  
 <s v="This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has a extremely light heat-treated aluminum frame, and steering that allows precision control."/>

<s v="No Discount"/>  
 <s v="No Discount"/>  
 <s v="No Discount"/>

<x v="0"/>  
 <x v="0"/>  
 <x v="1"/>  
 <s v="Australian Dollar"/>

<n v="1"/>  
 <n v="3578.27"/>  
 <n v="3578.27"/>  
 <n v="0"/>  
 <n v="0"/>  
 <n v="2171.2941999999998"/>

<n v="2171.2941999999998"/>  
 <n v="3578.27"/>  
 <n v="286.26159999999999"/>  
 <n v="89.456800000000001"/>  
 </r>  
</pivotCacheRecords>

In the context of pivotCacheRecords:

* r contains one record.

In the context of r:

* x is an index value referencing an item for this field, as defined in the pivotCacheDefinition part.
* s indicates that a value is being expressed inline in this record, and it is a string value.
* n indicates that a value is being expressed inline in this record, and it is a numeric value.

#### XML - pivotTable part

The pivotTable part is organized into 11 sections.

* Top-level attributes
* Location information
* Collection of Fields
* Fields on the row axis
* Items on the row axis (specific values)
* Fields on the column axis
* Items on the column axis (specific values)
* Fields in the report filter area
* Fields in the values area
* Style information
* This is what the shell of that structure looks like:

<pivotTableDefinition>  
 <location/>  
 <pivotFields/>  
 <rowFields/>  
 <rowItems/>  
 <colFields/>  
 <colItems/>  
 <pageFields/>  
 <dataFields/>  
 </dataFields>  
 <conditionalFormats/>  
 <pivotTableStyleInfo/>  
</pivotTableDefinition>

Each collection will now be addressed section by section.

##### Attributes on pivotTableDefinition

<pivotTableDefinition xmlns:sh="…" name="PivotTable2" cacheId="5"  
 applyNumberFormats="0" applyBorderFormats="0" applyFontFormats="0"  
 applyPatternFormats="0" applyAlignmentFormats="0"  
 applyWidthHeightFormats="1"  
 dataCaption="Values" updatedVersion="3" minRefreshableVersion="3"  
 showCalcMbrs="0" useAutoFormatting="1" colGrandTotals="0"  
 itemPrintTitles="1"  
 createdVersion="3" indent="0" outline="1" outlineData="1"  
 multipleFieldFilters="0">

In the context of pivotTableDefinition:

* name indicates the name of the PivotTable.
* cacheId references by Id a particular pivotCache in the pivotCaches collection listed in workbook.xml.
* applyNumberFormats value of "1" means to apply legacy autoformat number format properties.
* applyBorderFormats value of "1" means to apply legacy autoformat border format properties.
* applyFontFormats value of "1" means to apply legacy autoformat Font format properties.
* applyPatternFormats value of "1" means to apply legacy autoformat pattern format properties.
* applyAlignmentFormats value of "1" means to apply legacy autoformat alignment format properties.
* applyWidthHeightFormats value of "1" means to apply legacy autoformat width and height format properties.
* 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.
* updatedVersion is the Pivot version that last updated the PivotTable.
* minRefreshableVersion is the minimum Pivot version required to update this PivotTable's Pivot Cache.
* showCalcMbrs indicates whether calculated members should be shown in the PivotTable. Only applies to PivotTables based on OLAP sources.
* useAutoFormatting indicates whether autoformatting has been applied to the PivotTable.
* colGrandTotals indicates whether column grand totals are on for this PivotTable.
* rowGrandTotals defaults to "1" and therefore is not written.
* itemPrintTitles flag indicating whether PivotItem names should be repeated at the top of each printed page.
* createdVersion The Pivot version that created the cache.
* indent indentation increment for compact row axis, which means the Report Layout is set to Compact Form.
* outline flag indicating whether new fields should have their outline form flag set to "1".
* outlineData flag indicating whether the values field in the PivotTable should be displayed in outline form.
* multipleFieldFilters flag indicating whether each field of a pivot table can have multiple filters set on it.

##### Location Information

Location provides details on where the PivotTable is located in the sheet.

<location ref="B6:G13" firstHeaderRow="1" firstDataRow="4"  
 firstDataCol="1" rowPageCount="3" colPageCount="1"/>

In the context of location:

* ref the location of the PivotTable area, not including the report filter area.
* firstHeaderRow the first row of the PivotTable header, relative to the top left cell in ref value.
* firstDataRow the first row of the PivotTable values area, relative to the top left cell in ref value.
* firstDataCol the first column of the PivotTable values area, relative to the top left cell in ref value.
* rowPageCount indicates how many rows the report filter area will occupy, as fields are added to it, before taking up another column (there can be multiple rows and columns of fields in the report filter area). By default there is a single column of report filter fields and the fields occupy as many rows as there are fields..
* colPageCount indicates how many columns the report filter region will occupy, as fields are added to it, before taking up another row (there can be multiple rows and columns of fields in the report filter region). By default, there is a single column of report filter fields and the fields occupy as many rows as there are fields.

##### PivotTable Fields

This collection expresses item order and field information for each field associated with the PivotTable, whether shown in the PivotTable report or not. (Note that items have been removed from the "Customer Name" and "City" fields (1st and 6th) to shorten the example.)

<pivotFields count="28">  
 <pivotField showAll="0" includeNewItemsInFilter="1">  
 <items count="8">  
 <item x="66"/>  
 <item x="133"/>  
 <item x="74"/>  
 <item x="27"/>  
 <item x="118"/>  
 <item x="63"/>  
 <item x="141"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField axis="axisPage" showAll="0" includeNewItemsInFilter="1">  
 <items count="2">  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField axis="axisPage" showAll="0" includeNewItemsInFilter="1">  
 <items count="6">  
 <item x="3"/>  
 <item x="1"/>  
 <item x="2"/>  
 <item x="4"/>  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField axis="axisPage" showAll="0" includeNewItemsInFilter="1">  
 <items count="8">  
 <item x="0"/>  
 <item x="1"/>  
 <item x="2"/>  
 <item x="3"/>  
 <item x="4"/>  
 <item x="5"/>  
 <item x="6"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField axis="axisRow" showAll="0" includeNewItemsInFilter="1">  
 <items count="2">  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField axis="axisRow" showAll="0" includeNewItemsInFilter="1">  
 <items count="3">  
 <item x="1"/>  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField axis="axisCol" showAll="0" includeNewItemsInFilter="1">  
 <items count="2">  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField axis="axisCol" showAll="0" includeNewItemsInFilter="1">  
 <items count="2">  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField axis="axisCol" showAll="0" includeNewItemsInFilter="1">  
 <items count="4">  
 <item x="1"/>  
 <item x="2"/>  
 <item x="0"/>  
 <item t="default"/>  
 </items>  
 </pivotField>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField dataField="1" showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>

<pivotField showAll="0" includeNewItemsInFilter="1"/>  
</pivotFields>

In the context of pivotField:

* showAll flag indicating whether to show all items for this field.
* includeNewItemsInFilter 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.
* axis indicates on which axis this field is shown on the PivotTable.
* dataField indicates that this field is in the values area of the PivotTable.

In the context of items, which is a listing of items (by index) in this field. The order in which the items are listed is the order they would appear on a particular axis (row or column, for example). In this example, the first field is "Customer Name" and the first item referenced here is <item x="66"/>,which references the value "Adam L Flores" in the pivotCacheDefinition. Therefore if one added "Customer Name" to the row axis, "Adam L Flores" would be the first row item listed.

In the context of item:

* t value of 'default' indicates the subtotal or total item.

##### Row Axis Fields

This collection indicates which fields are on the row axis of the PivotTable.

<rowFields count="2">  
 <field x="7"/>  
 <field x="8"/>  
</rowFields>

In the context of field within rowFields:

* x is a zero based index into the pivotFields collection.

For this example, this collection indicates that "Product Category" and "Product Subcategory" are on the row axis of the PivotTable, in that order.

##### Row Items

This collection is a listing of all the values on the row axis of the PivotTable. In the spreadsheet example, the item values are found in cells B10:B13. For example, "Bikes" is in B10, and corresponds to the first I element below.

<rowItems count="4">  
 <i>  
 <x/>  
 </i>

<i r="1">  
 <x/>  
 </i>

<i r="1">  
 <x v="1"/>  
 </i>

<i t="grand">  
 <x/>  
 </i>  
</rowItems>

In the context of rowItems:

* i expresses all the values (for all fields) in one row of the row axis. There will be an I element for every row in the PivotTable.

In the context of i:

* r indicates how many fields/item values to "fill down" from the previous row item.

Note that the first item has no r explicitly written. Since a default of "0" is specified in the schema, for any item whose r is missing, a default value of "0" is implied.

In the context of x:

* v is a zero-based index referencing a pivotField item value. There will be as many x elements as there are item values in any particular row. Note that these x  elements may not be explicitly written, but instead "inherited" from the previous row/i element, via the value of r. Note also that the pivotField items don't list values explicitly, but instead reference a shared item value in the pivotCacheDefinition part.

Note that the first instance of x has no attribute value v associated with it, so v's default value of "0" is implied.

Looking at the layout of the PivotTable in the spreadsheet for this example, "Bikes" is the first (and only) item value in the first row, in cell B10. In the XML defining the PivotTable row item values, the first I element corresponds to the first row. There is a single index element x. The first (and only) x element corresponds to the first field on the row axis, namely "Product Category", and an index value of "0" indicates that the 0th item in the items collection for that pivotField definition is how to obtain the item value. Note that "Bikes" isn't explicitly listed as a value here, but instead the 0th item is an index to this field's shared items collection in the pivotCacheDefinition part.

For the second row, there are two item values, one item value (Bikes) from the first field in that row (Product Category) and one item value (Mountain Bikes) from the second field in that row (Product Subcategory). In the PivotTable, the first item value "Bikes" is hidden from view. In the XML for this example, the second I element expresses both item values for this row. The first item value "Bikes" is expressed implicitly, because the value of r on the second i element is '1', indicating that the first item value from the previous row will be reused again as the first item value for the current row. The second item value is expressed explicitly via the x element under the second i element. The index of '0' indicates that the 0th item in the pivotField element for that field is how to obtain the item value. Note again that the 0th item is itself an index into this field's shared items collection in the pivotCacheDefinition part.

The item values for the third row can be discovered in a similar way, so will not be discussed in detail here.

In the context of item:

* t value of 'default' indicates a grand total as the last row item value.

##### Column Axis Fields

This collection indicates which fields are on the column axis of the PivotTable.

<colFields count="3">  
 <field x="14"/>  
 <field x="15"/>  
 <field x="16"/>  
</colFields>

In the context of field:

* x is a zero based index into the pivotFields collection defined in this part.

For this example, the collection indicates that "Year", "Quarter" and "Month" are on the column axis of the PivotTable, in that order.

##### Column Items

This collection is a listing of all the values on the column axis of the PivotTable. In this example, the item values are found in cells C6:H8. For example, "2001" / "3" / "July" values are in C7:C9. Those are the first column item values and are referenced by the first <i> element below.

<colItems count="5">  
 <i>  
 <x/>  
 <x/>  
 <x/>  
 </i>

<i r="2">  
 <x v="1"/>  
 </i>

<i r="2">  
 <x v="2"/>  
 </i>

<i t="default" r="1">  
 <x/>  
 </i>

<i t="default">  
 <x/>  
 </i>  
</colItems>

In the context of colItems:

* i expresses all the values (for all fields) in one column of the column axis. There will be an i element for every column in the PivotTable column area.

In the context of i:

* r indicates how many fields/item values to "fill right" from the previous column.

Note that the first item has no r explicitly written so the default value of "0" is implied.

In the context of x:

* v is a zero-based index referencing a pivotField item value. There will be as many x elements as there are item values in any particular column. Note that these x elements sometimes are not explicitly written, but instead "inherited" from the previous column/i element, via the value of r. Note also that the pivotField items don't list values explicitly, but instead reference a shared item value in the pivotCacheDefinition part.

Note that the first instance of x has no attribute value v associated with it, so v's default value of "0" is implied.

The first i collection represents all item values for the first column in the column axis area of the PivotTable. The first x in the first i corresponds to the first field in the columns area of the PivotTable, namely "Year". The implied index value of '0' on this x indicates that the item value for this first item in the column is the 0th item for this pivotField. The 0th item for this pivotField is itself an index to an item value into this field's shared items collection in the pivotCacheDefinition part, namely "2001".

The item values corresponding to the second and third x elements can be found in the same way, arriving at "3" for the second item value, and arriving at "July" for the third item value for this first column.

The second i collection expresses all three item values for the second column in the column axis area. The r value of '2' indicates that the first two item values from the previous column will be repeated here, which means that the first item value for this second column will be "2001" again and the second item value for this second column will be "3". The third item value is expressed by the only x element under this second i element, and without further explanation is understood to reference the item value "August".

##### Report Filter Area Fields

This collection describes which fields are found in the report filter area of the PivotTable.

<pageFields count="3">  
 <pageField fld="2" hier="0"/>  
 <pageField fld="4" hier="0"/>  
 <pageField fld="5" hier="0"/>  
</pageFields>

In the context of pageField:

* fld is a zero-based index indicating the field to be on the report filter area.
* hier is an index of the OLAP hierarchy to which this belongs.

##### Values Area Fields

This collection describes which fields are found in the values area of the PivotTable.

<dataFields count="1">  
 <dataField name="Sum of Sales Amount" fld="25" baseField="0" baseItem="0"/>  
</dataFields>

In the context of dataField:

* name is the name of the values field.
* fld is the index of the field being summarized.
* baseField is the index of the base field when showDataAs calculation is in use.
* baseItem is the index of the base item when showDataAs calculation is in use.

##### PivotTable Style Information

Styles information is discussed in the informative section on spreadsheetML styles. Therefore the XML is provided for completeness, but will not be discussed here.

<pivotTableStyleInfo name="PivotStyleDark8" showRowHeaders="1"  
 showColHeaders="1" showRowStripes="0" showColStripes="0"  
 showLastColumn="1"/>  
</pivotTableDefinition>

## Styles

### Overview

There are several ways to express formatting applied to objects in a worksheet. SpreadsheetML supports the concepts of styles, themes, and direct formatting applied to cell ranges, tables, PivotTables, charts, and shapes.

A style is a named collection of formatting elements. A cell style specifies number format, cell alignment, font information, cell border specifications, colors, and background/foreground fills. A table style specifies formatting elements for the regions of a table (e.g., make the header row and totals bold face, and apply light gray fill to alternating rows in the data portion of the table to achieve striped or banded rows). A PivotTable style specifies formatting elements for the regions of a PivotTable (e.g., first- and second-level subtotals, row axis, column axis, and page fields).

A style can specify color, fonts, and shape effects directly, or these elements can be referenced indirectly by referring to a theme definition. The use of styles allows for quicker application of formatting and more consistently stylized documents.

A theme defines a set of colors, font information, and effects on shapes (including charts). If a style or formatting element defines its color, font, or effect by referencing a theme, then picking a new theme switches all the colors, fonts, and effects for that formatting element.

Applying direct dormatting means that particular elements of formatting (e.g., a bold font face or a number format) have been applied, but the elements of formatting have been chosen individually instead of collectively by choosing a named style. When applying direct formatting, themes can still be referenced, causing those elements to change as the theme is changed.

### File Architecture

There is a single Styles part for all the formatting definitions in the workbook. Similarly, there is a single Theme part to define the theme used in the workbook. These parts are related to the workbook. Each of the formatted objects refers by index to a set of formatting definitions, which are expressed in the Styles part. If the formatting element is part of a themed set, the element will reference an index to a theme element defined in the Themes part. The solid arrows represent relationships among the parts, and the dotted arrows represent references by index to various elements in the target part.

### Organization in the Styles Part

The Styles part is organized into the collections described in §3.11.4 through §3.11.9, in the following order. These collections are siblings in the Styles part XML definition, whose parent, the root node of this part, is styleSheet.

### Number Format Expressions

This contains the cell number formats that are used in this workbook. These never reference a theme. Built-in formats are not written to disk, so the example below represents a custom format:

<numFmts count="1">  
 <numFmt numFmtId="164" formatCode="&quot;$&quot;#,##0.00"/>  
</numFmts>

A numFmt definition is referenced by index (numFmtId) from either a cellXf or cellStyleXf element.

### Font Definitions

This contains the font definitions used in this workbook. Elements of the font definition may reference theme definitions.

<fonts count="1">  
 <font>  
 <b/>  
 <sz val="11"/>  
 <color theme="1"/>  
 <name val="Calibri"/>  
 <family val="2"/>  
 </font>  
</fonts>

This font definition specifies bold face, font size of 11, font name Calibri, and references a font color specified in the theme part, specifically the color whose index is 1 in the clrScheme collection. A font definition is referenced by index (fontId) from either a cellXf or cellStyleXf element.

### Fill Definitions

This contains the fills used in this workbook.

<fills count="1">  
 <fill>  
 <patternFill patternType="solid">  
 <fgColor theme="4"/>  
 <bgColor theme="4"/>  
 </patternFill>  
 </fill>  
</fills>

This fill definition specifies a solid pattern fill, whose color uses a themed color, whose index is 4 in the clrScheme collection of the Themes part. A fill definition is referenced by index (fillId) from either a cellXf or cellStyleXf element.

### Borders Definitions

This contains the border formats.

<borders count="1">  
 <border>  
 <left/>  
 <right/>  
 <top/>  
 <bottom/>  
 <diagonal/>  
 </border>  
</borders>

This example specifies a cell with left, right, top, and bottom borders. A border definition is referenced by index (borderId) from either a cellXf or cellStyleXf element.

### Master Records

The master cell style record (xf) ties together all the formatting (e.g., number format, font information, and fill) for a named cell style or for a cell's direct formatting. An xf inside cellStyleXfs is referenced by index (xfId) from a cellStyle definition, which names a particular cell style.

<cellStyleXfs count="1">  
 <xf numFmtId="0" fontId="0" fillId="0" borderId="0"/>  
</cellStyleXfs>

For a cell's direct formatting, an xf inside cellXfs is referenced by index (s) from a cell definition (c) in one of the sheets.

<cellXfs count="1">  
 <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"/>  
</cellXfs>

### Named Cell Styles

This is a collection of named cell styles used in the workbook.

<cellStyles count="1">  
 <cellStyle name="Accent1" xfId="1" builtinId="29"/>  
</cellStyles>

### Differential Formatting Records

Differential formatting enables subsets of formatting to be specified, without overriding other elements of formatting. For example, if it is desired to express "add bold face to whatever formatting is already there", then a dxf definition can be used. dxf definitions are used to express additional (or differential) formatting that will be applied via table styles or PivotTable styles. dxf definitions are referenced by index (dxfId) from a tableStyleElement. The formatting elements used in a dxf definition are subsets of formatting collections described above.

<dxfs count="1">  
 <dxf>  
 <font>  
 <b/>  
 <color theme="0"/>  
 </font>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="5"/>  
 <bgColor theme="5"/>  
 </patternFill>  
 </fill>  
 </dxf>  
</dxfs>

### Custom Table Style Definitions

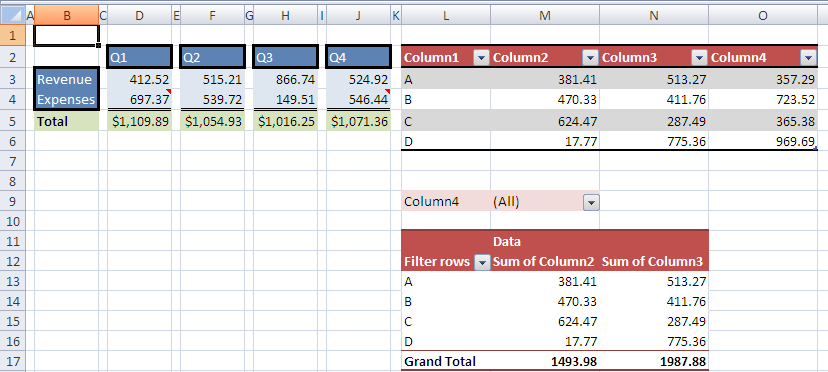
Built-in table and PivotTable styles are not saved out; only custom-defined styles are saved out. In this example, a custom table style defines formatting for an element of a table, the "whole table" region.

<tableStyles count="1" defaultTableStyle="TableStyleMedium9"  
 defaultPivotStyle="PivotStyleLight16">  
 <tableStyle name="TableStyleMedium10 - Custom" pivot="0" count="1">  
 <tableStyleElement type="wholeTable" dxfId="6"/>  
 </tableStyle>  
</tableStyles>

### Example

### Illustration

For this example, consider the following graphic representation of a worksheet:



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. This style 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.

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.

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.

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).

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.

### File Architecture

All of the cells illustrated are defined in the Sheet1 part in this example. The table is defined in the Table1 part and the PivotTable is defined in the part named PivotTable1. Each of the formatted objects refers to a set of formatting definitions that are expressed in the Styles part. If the formatting element is part of a themed set, the element will reference a theme element defined in the Themes part. The solid arrows represent relationships among the parts; the dotted arrows represent references by Id to various elements in the target part.

### The XML for this Example

<styleSheet>  
 <numFmts count="1">  
 <numFmt numFmtId="164" formatCode="&quot;$&quot;#,##0.00"/>  
 </numFmts>

<fonts count="5">  
 <font>  
 <sz val="11"/>  
 <color theme="1"/>  
 <name val="Calibri"/>  
 <scheme val="minor"/>  
 </font>

<font>  
 <b/>  
 <sz val="11"/>  
 <color theme="1"/>  
 <name val="Calibri"/>  
 <family val="2"/>  
 </font>

<font>  
 <b/>  
 <sz val="8"/>  
 <color indexed="81"/>  
 <name val="Calibri"/>  
 <charset val="1"/>  
 <scheme val="minor"/>  
 </font>

<font>  
 <sz val="8"/>  
 <color indexed="81"/>  
 <name val="Calibri"/>  
 <charset val="1"/>  
 <scheme val="minor"/>  
 </font>

<font>  
 <sz val="11"/>  
 <color theme="0"/>  
 <name val="Calibri"/>  
 <scheme val="minor"/>  
 </font>  
 </fonts>

<fills count="5">  
 <fill>  
 <patternFill patternType="none"/>  
 </fill>

<fill>  
 <patternFill patternType="gray125"/>  
 </fill>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="4"/>  
 <bgColor theme="4"/>  
 </patternFill>  
 </fill>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="6" tint="0.59999389629810485"/>  
 <bgColor indexed="65"/>  
 </patternFill>  
 </fill>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="4" tint="0.79998168889431442"/>"  
 <bgColor indexed="65"/>  
 </patternFill>  
 </fill>  
 </fills>

<borders count="5">  
 <border>  
 <left/>  
 <right/>  
 <top/>  
 <bottom/>  
 <diagonal/>  
 </border>

<border>  
 <left/>  
 <right/>  
 <top/>  
 <bottom style="double">  
 <color indexed="64"/>  
 </bottom>  
 <diagonal/>  
 </border>

<border>  
 <left style="thick">  
 <color auto="1"/>  
 </left>

<right style="thick">  
 <color auto="1"/>  
 </right>

<top style="thick">  
 <color auto="1"/>  
 </top>

<bottom style="thick">  
 <color auto="1"/>  
 </bottom>  
 <diagonal/>  
 </border>

<border>  
 <left style="thick">  
 <color auto="1"/>  
 </left>

<right style="thick">  
 <color auto="1"/>  
 </right>

<top style="thick">  
 <color auto="1"/>  
 </top>  
 <bottom/>  
 <diagonal/>  
 </border>

<border>  
 <left style="thick">  
 <color auto="1"/>  
 </left>

<right style="thick">  
 <color auto="1"/>  
 </right>

<top/>

<bottom style="thick">  
 <color auto="1"/>"  
 </bottom>  
 <diagonal/>  
 </border>  
 </borders>

<cellStyleXfs count="2">  
 <xf numFmtId="0" fontId="0" fillId="0" borderId="0"/>  
 <xf numFmtId="0" fontId="4" fillId="2" borderId="0"  
 applyNumberFormat="0" applyBorder="0" applyAlignment="0"  
 applyProtection="0">  
 <protection/>  
 </xf>  
 </cellStyleXfs>

<cellXfs count="14">  
 <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"/>  
 <xf numFmtId="0" fontId="1" fillId="0" borderId="0" xfId="0" applyFont="1"/>  
 <xf numFmtId="4" fontId="0" fillId="0" borderId="0" xfId="0"  
 applyNumberFormat="1" applyBorder="1"/>

<xf numFmtId="164" fontId="0" fillId="0" borderId="0" xfId="0"  
 applyNumberFormat="1" applyBorder="1"/>  
 <xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"   
 pivotButton="1"/>

<xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"   
 applyAlignment="1">  
 <alignment horizontal="left"/>  
 </xf>

<xf numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"  
 applyNumberFormat="1"/>  
 <xf numFmtId="0" fontId="4" fillId="2" borderId="2" xfId="1"   
 applyBorder="1"/>  
 <xf numFmtId="0" fontId="4" fillId="2" borderId="3" xfId="1"   
 applyBorder="1"/>

<xf numFmtId="0" fontId="4" fillId="2" borderId="4" xfId="1"   
 applyBorder="1"/>  
 <xf numFmtId="0" fontId="1" fillId="3" borderId="0" xfId="0"  
 applyFont="1" applyFill="1"/>  
 <xf numFmtId="164" fontId="0" fillId="3" borderId="0" xfId="0"   
 applyNumberFormat="1" applyFill="1" applyBorder="1"/>

<xf numFmtId="4" fontId="0" fillId="4" borderId="0" xfId="0"  
 applyNumberFormat="1" applyFill="1" applyBorder="1"/>  
 <xf numFmtId="4" fontId="0" fillId="4" borderId="1" xfId="0"  
 applyNumberFormat="1" applyFill="1" applyBorder="1"/>  
 </cellXfs>

<cellStyles count="2">

<cellStyle name="Accent1" xfId="1" builtinId="29"/>  
 <cellStyle name="Normal" xfId="0" builtinId="0"/>  
 </cellStyles>

<dxfs count="7">  
 <dxf>  
 <fill>  
 <patternFill patternType="solid">  
 <fgColor theme="0" tint="-0.14999847407452621"/>  
 <bgColor theme="0" tint="-0.14999847407452621"/>  
 </patternFill>  
 </fill>  
 </dxf>

<dxf>  
 <fill>  
 <patternFill patternType="solid">  
 <fgColor theme="0" tint="-0.14999847407452621"/>  
 <bgColor theme="0" tint="-0.14999847407452621"/>  
 </patternFill>  
 </fill>  
 </dxf>

<dxf>  
 <font>  
 <b/>  
 <color theme="0"/>  
 </font>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="5"/>  
 <bgColor theme="5"/>  
 </patternFill>  
 </fill>  
 </dxf>

<dxf>  
 <font>  
 <b/>  
 <color theme="0"/>  
 </font>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="5"/>  
 <bgColor theme="5"/>  
 </patternFill>  
 </fill>  
 </dxf>

<dxf>  
 <border>  
 <top style="double">  
 <color theme="1"/>  
 </top>  
 </border>  
 </dxf>

<dxf>  
 <font>  
 <b/>  
 <color theme="0"/>  
 </font>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="5"/>  
 <bgColor theme="5"/>  
 </patternFill>  
 </fill>

<border>  
 <bottom style="medium">  
 <color theme="1"/>  
 </bottom>  
 </border>  
 </dxf>

<dxf>  
 <font>  
 <color theme="1"/>  
 </font>

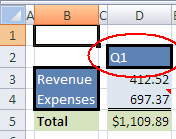
<border>  
 <top style="medium">  
 <color theme="1"/>  
 </top>

<bottom style="medium">  
 <color theme="1"/>  
 </bottom>  
 </border>  
 </dxf>  
 </dxfs>

<tableStyles count="1" defaultTableStyle="TableStyleMedium9"  
 defaultPivotStyle="PivotStyleLight16">

<tableStyle name="TableStyleMedium10 - Custom" pivot="0" count="7">  
 <tableStyleElement type="wholeTable" dxfId="6"/>  
 <tableStyleElement type="headerRow" dxfId="5"/>  
 <tableStyleElement type="totalRow" dxfId="4"/>  
 <tableStyleElement type="firstColumn" dxfId="3"/>  
 <tableStyleElement type="lastColumn" dxfId="2"/>  
 <tableStyleElement type="firstRowStripe" dxfId="1"/>  
 <tableStyleElement type="firstColumnStripe" dxfId="0"/>  
 </tableStyle>  
 </tableStyles>  
 <colors/>  
</styleSheet>

#### Cell D2 Formatting



Cell D2 contains the text "Q1", and is defined in the cell table of sheet1 as:

<c r="D2" s="7" t="s">  
 <v>0</v>  
</c>

On this cell, attribute s with value 7 indicates that the 7th xf definition of cellXfs holds the formatting information for the cell. The 7th xf of cellXfs is defined as follows:

<xf numFmtId="0" fontId="4" fillId="2" borderId="2" xfId="1" applyBorder="1"/>

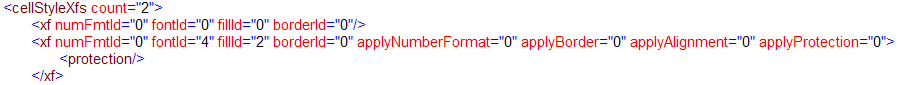
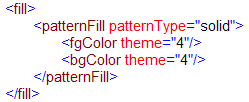
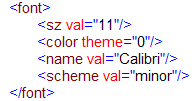
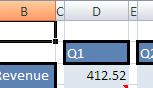
The number formatting information cannot be found in a numFmt definition because it is a built-in format. Instead, it is implicitly understood to be the 0th built-in number format; the font information can be found in the 4th font definition; the fill information can be found in the 2nd fill definition; the border information can be found in the 2nd border definition; this cell is using a cell style which is defined in the 1st cellStyleXf definition; Finally, borders specified in this xf should be applied.

Remember that these collections are zero-based.

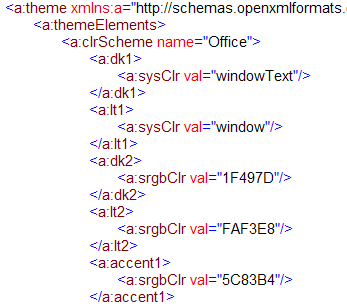
Additionally the fill definition for D2 references a themed color, whose index is 4th in the clrScheme definition of the theme part:

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="4"/>  
 <bgColor theme="4"/>  
 </patternFill>  
 </fill>

Graphically, the index references can be shown like this:

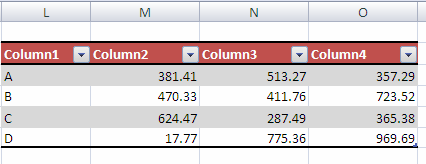


(built-in)



Start

### Custom Table Style



This range of cells is a table object with a custom table style applied. The table definition in table1 specifies which table style is applied, and which aspects of the table style definition are 'turned on' and should be applied:

<table id="2" name="Table11" displayName="Table11" ref="L20:O24"  
 totalsRowShown="0">  
 <tableStyleInfo name="TableStyleMedium10 - Custom" showFirstColumn="0"  
 showLastColumn="0" showRowStripes="1" showColumnStripes="0"/>  
</table>

The tableStyleInfo element indicates that this table uses the TableStyleMedium10 - custom style and that "first column", "last column", and "column stripes" formatting are OFF. It also indicates that "row stripes" formatting is ON.

Here is the TableStyleMedium10 - Custom definition in the Styles part:

<tableStyles count="1" defaultTableStyle="TableStyleMedium9"  
 defaultPivotStyle="PivotStyleLight16">  
 <tableStyle name="TableStyleMedium10 - Custom" pivot="0" count="7">  
 <tableStyleElement type="wholeTable" dxfId="6"/>  
 <tableStyleElement type="headerRow" dxfId="5"/>  
 <tableStyleElement type="totalRow" dxfId="4"/>  
 <tableStyleElement type="firstColumn" dxfId="3"/>  
 <tableStyleElement type="lastColumn" dxfId="2"/>  
 <tableStyleElement type="firstRowStripe" dxfId="1"/>  
 <tableStyleElement type="firstColumnStripe" dxfId="0"/>  
 </tableStyle>  
 </tableStyles>

<colors/>  
</styleSheet>

Note that even though column stripes are defined for this table style, they are not used for this instance of the table.

The header row formatting for this table is defined by the 5th dxf definition:

<dxf>  
 <font>  
 <b/>  
 <color theme="0"/>  
 </font>

<fill>  
 <patternFill patternType="solid">  
 <fgColor theme="5"/>  
 <bgColor theme="5"/>  
 </patternFill>  
 </fill>

<border>  
 <bottom style="medium">  
 <color theme="1"/>  
 </bottom>  
 </border>  
 </dxf>

This formatting indicates that for the header row of this table, the font is bold face and uses a themed color; the fill is solid and uses a themed color; and there is a bottom border on the cells.

## Calculation Chain

### Overview

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

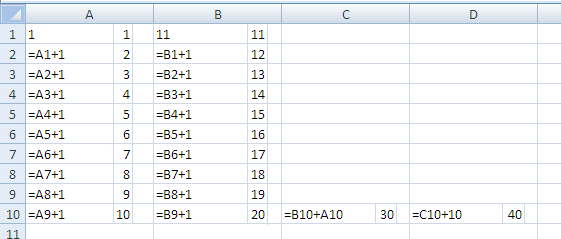
Any particular calculation event can cause the calculation chain order to be rearranged or altered. For example, adding more formulas to the workbook will add references in the Calculation Chain part.

Another example of how the calculation order can be updated involves the idea of partial calculation. Partial 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.

While calculation chain information can be loaded by a spreadsheet application, it is not required. A 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.

### Example

Consider the following set of formulas in a workbook:



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

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

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

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

#### Partial Calculation

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

<calcChain xmlns="…">  
 <c r="D10" i="1"/>  
 <c r="C10"/>  
 <c r="A3"/>  
 <c r="B3"/>

<c r="A4"/>  
 <c r="B4"/>  
 <c r="A5"/>  
 <c r="B5"/>  
 <c r="A6"/>

<c r="B6"/>  
 <c r="A7"/>  
 <c r="B7"/>  
 <c r="A8"/>  
 <c r="B8"/>

<c r="A9"/>  
 <c r="B9"/>  
 <c r="A10"/>  
 <c r="B10"/>  
 <c r="B2"/>  
 <c r="A2"/>  
</calcChain>

Every c element represents a cell containing a formula. The first cell calculated appears first (top-to-bottom), and so on. The reference attribute r indicates the cell's address in the sheet. The index attribute i indicates the index of the sheet with which that cell is associated. The sub-chain attribute s (not present in this first example) indicates that this cell can be treated as a sub chain of the preceding cell. Sub-chains can be useful when calculation can be multi-threaded or calculated concurrently. Whenever a cell does not contain an i or s attribute, it is understood to inherit these values from the previous cell.

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

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

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

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

#### First Full Calculation

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

<calcChain xmlns="…">  
 <c r="B2" i="1"/>  
 <c r="B3" s="1"/>  
 <c r="B4" s="1"/>  
 <c r="B5" s="1"/>

<c r="B6" s="1"/>  
 <c r="B7" s="1"/>  
 <c r="B8" s="1"/>  
 <c r="B9" s="1"/>  
 <c r="B10" s="1"/>

<c r="C10" s="1"/>  
 <c r="D10" s="1"/>  
 <c r="A2"/>  
 <c r="A3" s="1"/>  
 <c r="A4" s="1"/>

<c r="A5" s="1"/>  
 <c r="A6" s="1"/>  
 <c r="A7" s="1"/>  
 <c r="A8" s="1"/>  
 <c r="A9" s="1"/>  
 <c r="A10" s="1"/>  
</calcChain>

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

Additionally, the application has discovered that the formulas in column B can be calculated in parallel with the formulas in column A (i.e., they don't depend on each other). This is evidenced by the presence of s="1" on the cell element for cells B2:B10, indicating that B2:10 are part of a "child-chain" starting with B2. Note also that C10 and B10 are included in that child chain, even though these formulas do, in fact, depend on calculated values from column A. This is due to the multi-threaded nature of the calculation engine. Currently the chain on which C10 and D10 reside can be calculated concurrently with the chain on which A2:A10 reside because by the time C10 and D10 need to be calculated (e.g., by CPU #1), A10 has already been calculated (e.g., by CPU #2). In some future calculation, the timing may be different, and at that time, the application will need to resort to moving C10 and D10 to a new calculation level (see §3.12.2.3).

#### Twentieth 20th Full Calculation

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

<calcChain xmlns="…">  
 <c r="B2" i="1"/>  
 <c r="B3" s="1"/>  
 <c r="B4" s="1"/>  
 <c r="B5" s="1"/>

<c r="B6" s="1"/>  
 <c r="B7" s="1"/>  
 <c r="B8" s="1"/>  
 <c r="B9" s="1"/>  
 <c r="B10" s="1"/>

<c r="A2"/>  
 <c r="A3" s="1"/>  
 <c r="A4" s="1"/>  
 <c r="A5" s="1"/>  
 <c r="A6" s="1"/>

<c r="A7" s="1"/>  
 <c r="A8" s="1"/>  
 <c r="A9" s="1"/>  
 <c r="A10" s="1"/>  
 <c r="C10" l="1"/>  
 <c r="D10" s="1"/>  
</calcChain>

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

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

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

## Worksheet Metadata

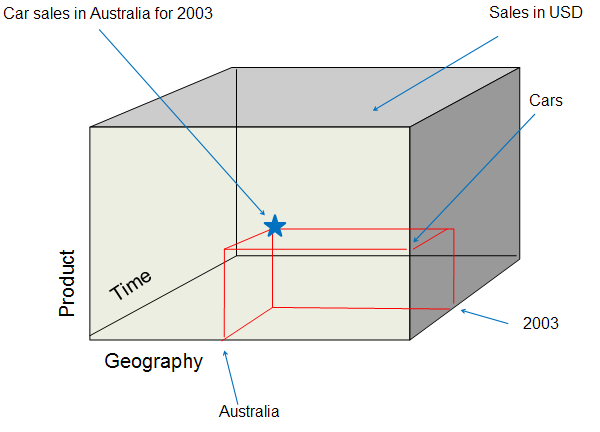
### Overview

Value and cell metadata are additional properties that can be associated with a particular cell or value. Cell metadata properties can be carried along with the cell as it moves (e.g., via insert, shift, copy/paste, merge, or unmerge) and value metadata properties can be propagated along with the value as it is referenced in formulas.

All of this metadata is stored separately in the metadata.xml part in the workbook.

While the architecture of this feature allows for future extensions, only MDX metadata—metadata that is associated with a particular cube function and it's results—is currently defined. For example, if a CUBEMEMBER function call is used to identify a particular member in an OLAP cube, then the metadata would express the OLAP connection name, the mdx expression identifying that member, and various operational attributes of that metadata (e.g., whether it propagates through formula assignment, shifts with the cell when the cell moves locations in the grid, and so on).

#### OLAP Cube Review



Consider the 3-dimensional OLAP cube above. The three dimensions of the cube are "Product", "Geography", and "Time". "Sales Amount" is the measure being summarized, and is often considered an additional dimension of the cube. OLAP cubes can be N-dimensional, while this one has three dimensions.

Within each dimension are hierarchies, or ways of organizing the dimension into various levels of granularity. For example, within the Time dimension there can exist the Calendar Year / Quarter / Month / Day hierarchy. Likewise, the Time dimension can also have the Fiscal Year / Quarter / Month / Day hierarchy. Each of Year / Quarter / Month / Day represents levels in the hierarchy. '2003' is considered a member of the 'Year' level within the hierarchy. Likewise, the Product dimension can have multiple hierarchies. A hierarchy could be constructed based on the type of product while another hierarchy could be constructed based on the color of the product.

In the example above, picking a member from one dimension would be visualized as a slice through the cube. For example, picking 'Australia' from the Geography dimension could be a relatively thick slice of the cube, if there were many levels underneath 'Country', like 'State', 'City', and 'PostalCode'. Picking a member from Geography that is more granular than 'Australia" results in a thinner slice of the cube in the Geography dimension, because now some of Australia will have been omitted from the data.

A tuple is the intersection of two or more members from distinct dimensions. In the example above, three members from three dimensions are expressed:

1. From Geography we have [Geography].[Country].&[Australia]
2. From Product we have [Product].[Cars].children
3. From Time we have [Time].[Calendar].[Calendar Year].&[2003]

#### OLAP Function Summary

There are seven recognized “CUBE” functions that can be used in cell formulas. These functions enable formulas in any cell to fetch data from Analysis Services (2000 & 2005). Specifically, the functions can fetch any member, set, aggregated value, property, or KPI from the OLAP cube. Because it is formula-driven, the layout of this data is as flexible as cells and formulas. The OLAP data can be placed anywhere on the spreadsheet, intermingled with other local calculations or within other formulas.

The function names are: CUBEKPIMEMBER, CUBEMEMBER, CUBEMEMBERPROPERTY, CUBERANKEDMEMBER, CUBESET, CUBESETCOUNT, and CUBEVALUE.

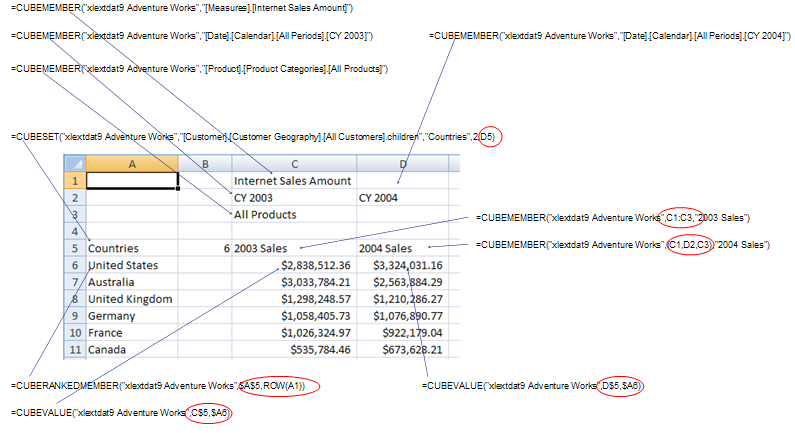
### File Architecture – Relationships

The workbook holds the relationship to the metadata part, and cells within a sheet reference the items in the metadata part.

### Example

In the following example, Cube functions are used to build up a report of internet sales by country, for all products, for the calendar years 2003 and 2004.

#### Illustration



##### Function Summary

CUBEMEMBER(connection, member-expression, caption) returns a member in the cube (e.g., Bicycles, Cars, All Products, CY 2004) (member can be a tuple)

CUBESET(connection, set-expression, caption, sort-order, sort-by) returns a set of members (e.g., all Countries)

CUBERANKEDMEMBER(connection, set-expression, rank, caption) returns one member of the referenced set (e.g., Australia)

CUBEVALUE(connection, member-expression-1, member-expression-2, …) returns the aggregate summarized value for the intersection of members specified.

##### Walk Through

C1 contains a CUBEMEMBER function call specifying the "Internet Sales Amount" member from the Measures dimension. C2 contains a CUBEMEMBER function call specifying "CY 2003" from the Date dimension. D2 contains a similar function specifying "CY2004" from the Date dimension. C3 contains a CUBEMEMBER function call specifying "All Products" from the Product dimension. Each of these cells contain simple string values (e.g., "All Products" in C3), and each of these cells is associated with mdx metadata which specifies the mdx expression identifying a particular member of a particular dimension (e.g., [Measures].[Internet Sales Amount]).

A5 contains a CUBESET function call specifying a set of members. Additionally, the CUBESET function call allows for specifying a caption for the cell ("Countries"), a sort order for sorting the set (in this case, "2" corresponds to descending), and a sort by field (in this case the set will be sorted by the member as expressed in the mdx associated with cell D5, labeled "2004 Sales"). Finally, it should be noted that CUBESET returns a set of members, not just a single member.

Cells A6:A11 use the CUBERANKEDMEMBER function to return the individual members, by rank, returned from the CUBESET function call in A5. For example, A6 uses the "xlextdat9 Adventure Works" connection to connect to the OLAP cube, and addresses the first member (because "ROW(A1)" resolves to "1") in the set returned in A5.

Cell C6 uses the CUBEVALUE function to return measure data identified by intersecting the mdx expression found in A6 with the mdx expression found in C5 ("CY 2003 Internet Sales for All Products in the United States"). C7:C11 use similar CUBEVALUE function calls. D6:D11 involves similar functions as well, but using "CY 2004" instead.

The power of metadata in this example is that anytime a CUBE function argument referenced another cell, and that argument requires a set or member mdx expression, the mdx metadata for that referenced cell is returned to the calling function instead of the simple string value. For example, A6 contains a CUBERANKEDMEMBER function call, whose second argument is expecting a set of members. The reference for that argument is $A$5. Instead of using the A5's string value of "Countries" (which would result in a error), A5's mdx expression metadata is used instead, which returns a set. Similarly any of the CUBEVALUE function calls rely on cell references, where those cells contain mdx metadata used to pinpoint the measure data requested. Furthermore, each of the cells referenced by C6:D11, in turn reference other cells' mdx metadata. In this way, the mdx metadata is able to propagate through the formula calculation chain.

#### Worksheet Metadata XML

##### General Organization

<metadata>  
 <metadataTypes/>  
 <metadataStrings/>  
 <mdxMetadata/>  
 <valueMetadata/>  
</metadata>

There are four general collections in the metadata part:

* metadataTypes - expresses various application runtime behaviors that apply to a set of metadata.
* 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.
* mdxMetadata - expresses the tuples in use in this workbook.
* 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 references additional element collections in the metadata part to define fully the metadata associated with this particular record, and therefore the full metadata definition for a particular cell's value. The records in valueMetadata serve as the bridge between the metadata definitions and the cells or values in the sheet.

##### Metadata Behaviors

The metadata type expresses operations on cells that allow the metadata to remain associated with the cell. Operations not listed or set to '0' would cause the metadata to no longer be associated with the cell.

<metadataTypes count="1">  
 <metadataType name="XLMDX" minSupportedVersion="120000" copy="1"  
 pasteAll="1" pasteValues="1" merge="1" splitFirst="1"  
 rowColShift="1" clearFormats="1" clearComments="1" assign="1"  
 coerce="1"/>  
</metadataTypes>

Regarding metadataTypes:

* count is the number of metadataType elements.
* type is a particular set of cell operations.

Regarding metadataType:

* name is the name of this particular metadata type.
* minSupportedVersion indicates the earliest version of the application which supports this metadata type.
* copy value of 1 indicates that this metadata will be copied to other cells when the cell is copied.
* pasteAll value of 1 indicates that this metadata will be pasted to another cell when 'paste all' is chosen during a copy/paste operation.
* pasteValues value of 1 indicates that this metadata will be pasted to another cell when only the values of the cell is pasted during a copy/paste operation.
* merge value of 1 indicates that when the cell is merged, the metadata associated with the cell remains.
* splitFirst value of 1 indicates that when a merged cell is split, the metadata associated with the merged cell is only applied to the first (from top left) cell resulting from the split.
* rowColShift value of 1 indicates that metadata associated with a cell remains after rows and columns are inserted, even when the cell is moved.
* clearFormats value of 1 indicates that the metadata remains after the cell has been cleared of all formatting.
* clearComments value of 1 indicates that the metadata remains after comments have been cleared from the cell.
* assign value of 1 indicates that the metadata propagates through formula assignment operations
* coerce value of 1 indicates that the metadata can be removed when the data type is coerced to another type.

##### Metadata Strings

This collection is a set of string resources for the metadata part. Most follow the format of an mdx expression. Connection names (to OLAP cubes) are also expressed here.

<metadataStrings count="12">  
 <s v="xlextdat9 Adventure Works"/>  
 <s v="[Measures].[Internet Sales Amount]"/>  
 <s v="[Date].[Calendar].[Calendar Year].&amp;[2003]"/>  
 <s v="[Date].[Calendar].[Calendar Year].&amp;[2004]"/>

<s v="[Product].[Product Categories].[All Products]"/>  
 <s v="[Customer].[Customer Geography].[All Customers].children"/>  
 <s v="[Customer].[Customer Geography].[Country].&amp;[Australia]"/>  
 <s v="[Customer].[Customer Geography].[Country].&amp;  
 [United States]"/>

<s v="[Customer].[Customer Geography].[Country].&amp;  
 [United Kingdom]"/>  
 <s v="[Customer].[Customer Geography].[Country].&amp;[Germany]"/>  
 <s v="[Customer].[Customer Geography].[Country].&amp;[France]"/>  
 <s v="[Customer].[Customer Geography].[Country].&amp;[Canada]"/>  
</metadataStrings>

Regarding metadataStrings:

* count indicates the number of strings in the collection.
* s is the string container element
* v is the string value itself.

##### mdxMetadata

This collection expresses mdx metadata, and builds up the mdx members, sets, KPIs, and member properties.

valueMetadata records reference these records.

<mdxMetadata count="26">  
 <mdx n="0" f="m">  
 <t c="1">  
 <n x="1"/>  
 </t>  
 </mdx>

<mdx n="0" f="m">  
 <t c="1">  
 <n x="2"/>  
 </t>  
 </mdx>

<mdx n="0" f="m">  
 <t c="1">  
 <n x="3"/>"  
 </t>  
 </mdx>

<mdx n="0" f="m">  
 <t c="1">  
 <n x="4"/>  
 </t>  
 </mdx>

<mdx n="0" f="m">  
 <t c="3">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 </t>  
 </mdx>

<mdx n="0" f="m">  
 <t c="3">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 </t>  
 </mdx>

<mdx n="0" f="r">  
 <t c="1">  
 <n x="6"/>  
 </t>  
 </mdx>

<mdx n="0" f="r">  
 <t c="1">  
 <n x="7"/>  
 </t>  
 </mdx>

<mdx n="0" f="r">  
 <t c="1">  
 <n x="8"/>  
 </t>  
 </mdx>

<mdx n="0" f="r">  
 <t c="1">  
 <n x="9"/>  
 </t>  
 </mdx>

<mdx n="0" f="r">  
 <t c="1">  
 <n x="10"/>  
 </t>  
 </mdx>

<mdx n="0" f="r">  
 <t c="1">  
 <n x="11"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 <n x="6"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 <n x="7"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 <n x="7"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 <n x="8"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 <n x="8"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 <n x="9"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 <n x="9"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 <n x="10"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 <n x="10"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 <n x="11"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 <n x="11"/>  
 </t>  
 </mdx>

<mdx n="0" f="v">  
 <t c="4" ct="en-US">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 <n x="6"/>  
 </t>  
 </mdx>

<mdx n="0" f="s">  
 <ms ns="5" c="3" o="d">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 </ms>  
 </mdx>

<mdx n="0" f="c">  
 <ms ns="5" c="3" o="d">  
 <n x="1"/>  
 <n x="3"/>  
 <n x="4"/>  
 </ms>  
 </mdx>  
</mdxMetadata>

Regarding mdxMetadata:

* count indicates the number of mdx statements in the collection.

Regarding mdx, which is a particular mdx statement:

* n indicates the index of the record in metadataStrings containing the connection name.
* f indicates the name of the calling cube function in the workbook.

Regarding t, which is an mdx tuple:

* c is the count of member expressions in the mdx tuple.

Regarding n:

* x is the index value into metadataStrings indicating the particular member expression for this dimension of the tuple expression.

For example, cell C5 has a CUBEMEMBER function call expressing the result of "Internet Sales Amount of All Products for CY 2003". In sheet1.xml, cell C5 has vm="5", which means it has an associated valueMetadata record whose index is "5". Looking ahead into the valueMetadata records, the 5th (1-based) record points to the 4th (zero-based) mdx collection in mdxMetadata.

The 5th mdx collection:

<mdx n="0" f="m">  
 <t c="3">  
 <n x="1"/>  
 <n x="2"/>  
 <n x="4"/>  
 </t>  
 </mdx>

Where <n x="1"/> corresponds to the 1st position in the string store, namely

<s v="[Measures].[Internet Sales Amount]"/>

and where <n x="2"/> corresponds to the 2nd position in the string store, namely

<s v="[Date].[Calendar].[Calendar Year].&amp;[2003]"/>

and where <n x="4"/> corresponds to the 4th position in the string store, namely

<s v="[Product].[Product Categories].[All Products]"/>.

Therefore this data point in the cube is addressed by intersecting these three hierarchies, one in each dimension of the OLAP cube:

* [Measures].[Internet Sales Amount]
* [Date].[Calendar].[Calendar Year].[2003]
* [Product].[Product Categories].[All Products]

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.

##### valueMetadata

This collection defines cell or value metadata information (depending on the value of metadataType's cellMeta)

<valueMetadata count="26">  
 <bk>  
 <rc t="1" v="0"/>  
 </bk>

<bk>  
 <rc t="1" v="1"/>  
 </bk>

<bk>  
 <rc t="1" v="2"/>  
 </bk>

<bk>  
 <rc t="1" v="3"/>  
 </bk>

<bk>  
 <rc t="1" v="4"/>  
 </bk>

<bk>  
 <rc t="1" v="5"/>  
 </bk>

<bk>  
 <rc t="1" v="6"/>  
 </bk>

<bk>  
 <rc t="1" v="7"/>  
 </bk>

<bk>  
 <rc t="1" v="8"/>  
 </bk>

<bk>  
 <rc t="1" v="9"/>  
 </bk>

<bk>  
 <rc t="1" v="10"/>  
 </bk>

<bk>  
 <rc t="1" v="11"/>  
 </bk>

<bk>  
 <rc t="1" v="12"/>  
 </bk>

<bk>  
 <rc t="1" v="13"/>  
 </bk>

<bk>  
 <rc t="1" v="14"/>  
 </bk>

<bk>  
 <rc t="1" v="15"/>  
 </bk>

<bk>  
 <rc t="1" v="16"/>  
 </bk>

<bk>  
 <rc t="1" v="17"/>  
 </bk>

<bk>  
 <rc t="1" v="18"/>  
 </bk>

<bk>  
 <rc t="1" v="19"/>  
 </bk>

<bk>  
 <rc t="1" v="20"/>  
 </bk>

<bk>  
 <rc t="1" v="21"/>  
 </bk>

<bk>  
 <rc t="1" v="22"/>  
 </bk>

<bk>  
 <rc t="1" v="23"/>  
 </bk>

<bk>  
 <rc t="1" v="24"/>  
 </bk>

<bk>  
 <rc t="1" v="25"/>  
 </bk>  
</valueMetadata>

Regarding valueMetadata:

* count indicates the number of metadata block records.

Regarding bk, which is a metadata block, and rc, which is a metadata record:

* t indicates the index of the metadataType record in metadataTypes collection.
* v is the index of metadata record value in the storage corresponding to record type.

Looking at the first block using the bk element, the type of metadata with which this record is associated is the first (and only) metadataType record, which is of type "XLMDX". This indicates that the v index is pointing to the 0th mdxMetadata record.

## Shared Workbook Revisions

### Overview

The Shared Workbooks architecture enables a spreadsheet application to record revisions made to a workbook (e.g., track changes), and is designed to enable multiple users to edit the same workbook at the same time. Therefore, the application needs to support the ability to read changes made by another user, and update its own state of the same workbook with those changes, even when those changes are made concurrently with other changes made by other users. Inevitably, there will be conflicts, and therefore merge conflict resolution should be supported by the runtime application.

This architecture supports the ability to track changes made by a single user as well.

### How It Works

Workbook

Relationship diagram

A Shared Workbook must have shared mode turned on. For unsaved workbooks, this will require a save, because revisions will be stored in the file.

Changes to the workbook are saved as Shared Workbook Revision Header parts within the document at each save or time interval specified.

A table summarizing the revision logs (revisionHeaders.xml) tracks when changes are made, who made them, and lists the relationship  id to the specific Shared Workbook Revision Log part.

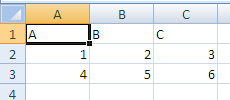
The application scans the summary table for new change logs and merges them into the workbook.

### Example

Consider a series of edits made by different users.

#### 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:



Once the file is saved to disk after these edits, the summary table is updated and the revision log for this change is written.

##### 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:

<header guid="{902054C2-C7B5-48BA-BFB2-4D439D9758D6}"  
 dateTime="2006-04-14T10:33:16" maxSheetId="4" userName="User 1"  
 r:id="rId2" minRId="1" maxRId="11">

<sheetIdMap count="3">  
 <sheetId val="1"/>  
 <sheetId val="2"/>  
 <sheetId val="3"/>  
 </sheetIdMap>  
</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.

##### First Edit Revision Log

Inside the corresponding Shared Workbook Revision Log part is the following content:

<revisions xmlns="…" xmlns:r="…">  
 <rcc rId="1" sId="1">  
 <nc r="A1" t="inlineStr">  
 <is>  
 <t>A</t>  
 </is>  
 </nc>  
 </rcc>

<rcc rId="2" sId="1">  
 <nc r="B1" t="inlineStr">  
 <is>  
 <t>B</t>  
 </is>  
 </nc>  
 </rcc>

<rcc rId="3" sId="1">  
 <nc r="C1" t="inlineStr">  
 <is>  
 <t>C</t>  
 </is>  
 </nc>  
 </rcc>

<rrc rId="4" sId="1" eol="1" ref="A2:XFD2" action="insertRow"/>

<rcc rId="5" sId="1">  
 <nc r="A2">  
 <v>1</v>  
 </nc>  
 </rcc>

<rcc rId="6" sId="1">  
 <nc r="B2">  
 <v>2</v>  
 </nc>  
 </rcc>

<rcc rId="7" sId="1">  
 <nc r="C2">  
 <v>3</v>  
 </nc>  
 </rcc>

<rrc rId="8" sId="1" eol="1" ref="A3:XFD3" action="insertRow"/>

<rcc rId="9" sId="1">  
 <nc r="A3">  
 <v>4</v>  
 </nc>  
 </rcc>

<rcc rId="10" sId="1">  
 <nc r="B3">  
 <v>5</v>  
 </nc>  
 </rcc>

<rcc rId="11" sId="1">  
 <nc r="C3">  
 <v>6</v>  
 </nc>  
 </rcc>  
</revisions>

rId is the revision Id, and indicates the order in which the particular revision should be applied.

sId indicates the sheet to which this revision applies.

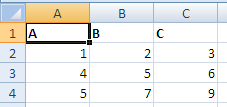
rcc means "revision cell change"

nc means new cell, and is of type CT\_Cell (see §3.2 for more information on the cell definition). Note that instead of using a shared string table, strings are expressed inline for these cells.

rrc means "revision row/column". Note that rrc can have an associated action, like insertRow (or deleteRow), which would cause a row to be inserted (or deleted) at that step in the series of revisions.

#### Second Edit

During the second edit, bold facing has been applied to A1:C1, and a formula has been applied to A4:C4 to sum the data in the table. For example, A4 contains =SUM(A2:A3).



Once the file is saved to disk after these edits, the summary table is updated and the revision log for this change is written.

##### 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:

<header guid="{A3A5EE09-2092-433C-895D-77D5A15DC847}"  
 dateTime="2006-04-14T10:34:10" maxSheetId="4" userName="User 2"  
 r:id="rId3" minRId="12" maxRId="15">

<sheetIdMap count="3">  
 <sheetId val="1"/>  
 <sheetId val="2"/>  
 <sheetId val="3"/>  
 </sheetIdMap>  
</header>

This time the user name has been updated. Use the r:id value of rId3 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way, the corresponding Shared Workbook Revision Log can be located.

##### Second Edit Revision Log

Inside the corresponding Shared Workbook Revision Log part is the following content:

<revisions xmlns="…" xmlns:r="…">  
 <rfmt sheetId="1" sqref="A1:C1" start="0" length="2147483647">  
 <dxf>  
 <font>  
 <b/>  
 </font>  
 </dxf>  
 </rfmt>

<rrc rId="12" sId="1" eol="1" ref="A4:XFD4" action="insertRow"/>

<rcc rId="13" sId="1">  
 <nc r="A4">  
 <f>SUM(A2:A3)</f>  
 </nc>  
 </rcc>

<rcc rId="14" sId="1">  
 <nc r="B4">  
 <f>SUM(B2:B3)</f>  
 </nc>  
 </rcc>

<rcc rId="15" sId="1">  
 <nc r="C4">  
 <f>SUM(C2:C3)</f>  
 </nc>  
 </rcc>  
 <rcv guid="{34804977-BBD3-40C9-87A7-1779BEE2183C}" action="add"/>  
</revisions>

rfmt indicates a formatting revision

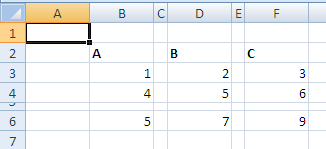
start and length indicate where to apply the formatting on the string

eol indicates that an insert is happening at the end of a list of data (end row)

rcv means "revision custom view", and indicates that a custom view is to be added.

#### Third Edit

During this editing session, column A has been inserted, and columns have been inserted between the data. Additionally, a row has been inserted between the data and the summary formula row, and at the top of the worksheet.



Once the file is saved to disk after these edits, the summary table is updated and the revision log for this change is written.

##### Summary Revision Table

Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

<header guid="{894981D2-DACF-4C1B-951C-EB199EA01DBF}"  
 dateTime="2006-04-14T10:36:10" maxSheetId="4" userName="User 2"  
 r:id="rId4" minRId="16" maxRId="20">

<sheetIdMap count="3">  
 <sheetId val="1"/>  
 <sheetId val="2"/>  
 <sheetId val="3"/>  
 </sheetIdMap>  
</header>

Use the r:id value of rId4 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way, the corresponding Shared Workbook Revision Log part can be located.

##### Third Edit Revision Log

Inside the corresponding Shared Workbook Revision Log part is the following content:

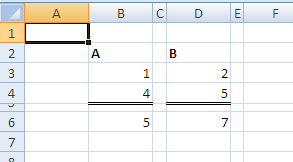
<revisions xmlns="…" xmlns:r="…">  
 <rrc rId="16" sId="1" ref="A1:XFD1" action="insertRow"/>  
 <rrc rId="17" sId="1" ref="A1:A1048576" action="insertCol"/>  
 <rrc rId="18" sId="1" ref="C1:C1048576" action="insertCol"/>  
 <rrc rId="19" sId="1" ref="E1:E1048576" action="insertCol"/>

<rrc rId="20" sId="1" ref="A5:XFD5" action="insertRow"/>  
 <rcv guid="{34804977-BBD3-40C9-87A7-1779BEE2183C}" action="delete"/>  
 <rcv guid="{34804977-BBD3-40C9-87A7-1779BEE2183C}" action="add"/>  
</revisions>

rrc indicates a "revision to row/column". There are several row inserts and column inserts expressed here.

#### Fourth Edit

During this edit, a double-underscore cell border was applied to B4 and D4. Also, column F (the data titled "C") was deleted.



Once the file is saved to disk after these edits, the summary table is updated and the revision log for this change is written.

##### Summary Revision Table

Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

<header guid="{A478A962-DEB9-43AA-BB25-2C54AFA155F1}"  
 dateTime="2006-04-14T10:37:14" maxSheetId="4" userName="User 2"  
 r:id="rId5" minRId="21">

<sheetIdMap count="3">  
 <sheetId val="1"/>  
 <sheetId val="2"/>  
 <sheetId val="3"/>  
 </sheetIdMap>  
</header>

Use the r:id value of rId5 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way, the corresponding Shared Workbook Revision Log part can be located.

##### Fourth Edit Revision Log

Inside the corresponding Shared Workbook Revision Log part is the following content:

<revisions xmlns="…" xmlns:r="…">  
 <rrc rId="21" sId="1" ref="F1:F1048576" action="deleteCol">  
 <rfmt sheetId="1" xfDxf="1" sqref="F1:F1048576" start="0" length="0"/>  
 <rcc rId="0" sId="1" dxf="1">  
 <nc r="F2" t="inlineStr">  
 <is>  
 <t>C</t>  
 </is>  
 </nc>

<ndxf>  
 <font>  
 <b/>  
 <sz val="11"/>  
 <color theme="1"/>  
 <name val="Calibri"/>  
 <scheme val="minor"/>  
 </font>  
 </ndxf>  
 </rcc>

<rcc rId="0" sId="1">  
 <nc r="F3">  
 <v>3</v>  
 </nc>  
 </rcc>

<rcc rId="0" sId="1">  
 <nc r="F4">  
 <v>6</v>  
 </nc>  
 </rcc>

<rcc rId="0" sId="1">  
 <nc r="F6">

<f>SUM(F3:F4)</f>  
 </nc>  
 </rcc>  
 </rrc>

<rfmt sheetId="1" sqref="B4" start="0" length="0">  
 <dxf>  
 <border>  
 <left/>  
 <right/>  
 <top/>  
 <bottom style="double">  
 <color auto="1"/>  
 </bottom>  
 </border>  
 </dxf>  
 </rfmt>

<rfmt sheetId="1" sqref="D4" start="0" length="0">  
 <dxf>  
 <border>  
 <left/>  
 <right/>  
 <top/>  
 <bottom style="double">  
 <color auto="1"/>  
 </bottom>  
 </border>  
 </dxf>  
 </rfmt>  
</revisions>

The first rrc element, with action="deleteCol" expresses that column F was deleted. Additionally, child collections of rrc contain all the column, formatting, and cell information (values and formulas) that was deleted as part of deleting column F.

xfDxf true means a whole row/column of formatting was affected.

dxf true means cell change includes format change

The rfmt collections at the bottom of this XML indicate that borders were applied to B4 and D4.

#### Fifth Edit

Rename "Sheet1" to "Published Numbers".

Before:



After:



Once the file is saved to disk after these edits, the summary table is updated and the revision log for this change is written.

##### Summary Revision Table

Contents of the Shared Workbook Revision Header part (revisionHeaders.xml).

<header guid="{B0CB8BC9-63A4-4830-8821-E03C053BD326}"  
 dateTime="2006-04-14T10:40:24" maxSheetId="4" userName="User 2"  
 r:id="rId6" minRId="22">

<sheetIdMap count="3">  
 <sheetId val="1"/>  
 <sheetId val="2"/>  
 <sheetId val="3"/>  
 </sheetIdMap>  
</header>

Use the r:id value of rId6 and then follow the relationship expressed in revisionHeaders.xml.rels. In this way the corresponding Shared Workbook Revision Log part can be located.

##### Fifth Edit Revision Log

Inside the corresponding Shared Workbook Revision Log part is the following content:

<revisions xmlns="…" xmlns:r="…">  
 <rsnm rId="22" sheetId="1" oldName="[SharedWorkbook.xlsx]Sheet1"  
 newName="[SharedWorkbook.xlsx]Published Numbers"/>  
</revisions>

rsnm means "revision: sheet name".

oldName indicates the name of the sheet before renaming it.

newName indicates the name of the sheet after renaming it.

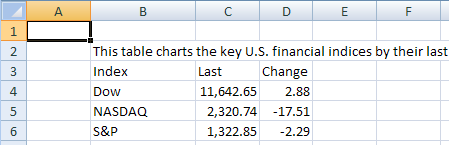
## Query Tables

### Overview

A QueryTable object is a range that is bound to an external data source. It is a cohesive range of cells in a sheet that share a common collection of properties and behaviors, separate from the connection itself. A QueryTable object can be associated with a cell range or a table definition.

### Web Query Example

This example illustrates a range, B2:D6, QueryTable rendering, which is data-bound to a table found on <http://www.msn.com>, specifically the financial information table usually found on that page.



#### QueryTable XML

<queryTable xmlns="…" name="www.msn" preserveFormatting="0"  
 connectionId="1"  
 autoFormatId="16" applyNumberFormats="0" applyBorderFormats="0"   
 applyFontFormats="1" applyPatternFormats="1" applyAlignmentFormats="0"   
 applyWidthHeightFormats="0"/>

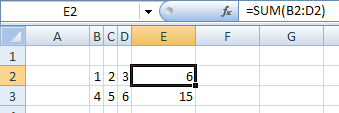
In the context of queryTable:

* name is the name of the QueryTable.
* preserveFormatting indicates whether to retain user-applied formatting after refresh or re-apply source data formatting.
* connectionId is the workbook connection's id.
* autoFormatId identifies (by implied index) the auto-format applied to the QueryTable.

All remaining attributes beginning with apply… indicate whether to apply this particular aspect of the auto-format definition.

### Text Import Example

This example illustrates a range (B2:D3) QueryTable rendering which is data-bound to a text file. Notice that formulas are entered in E2:E3, the column directly to the right of the QueryTable range.



#### QueryTable XML

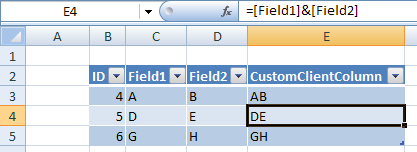
<queryTable xmlns="…" name="Text" refreshOnLoad="1" fillFormulas="1"  
 removeDataOnSave="1" connectionId="3" autoFormatId="16"   
 applyNumberFormats="0" applyBorderFormats="0" applyFontFormats="1"   
 applyPatternFormats="1" applyAlignmentFormats="0"   
 applyWidthHeightFormats="0"/>

In the context of queryTable:

* refreshOnLoad value of 1 indicates that this QueryTable should be refreshed when the workbook is opened.
* fillFormulas indicates that this QueryTable has immediately adjacent columns (which are not part of the QueryTable range) containing formulas that need to be filled down as the QueryTable grows and shrinks in size after refresh.
* removeDataOnSave indicates that the data in the worksheet resulting from the QueryTable refresh should be removed from the worksheet when saved and closed.

### Access Table Example

This example demonstrates a QueryTable that is applied to a Table object. This data came from connecting to an Access database table with four fields: "ID", "Field1", "Field2", and "Field3". "Field3" has been deleted from the QueryTable in the worksheet below. Notice that a calculated column has been added to the Table, in the column titled "CustomClientColumn", which concatenates the values from "Field1" and "Field2".



#### QueryTable XML

<queryTable xmlns="…" name="Database1.accdb" connectionId="2"  
 autoFormatId="16" applyNumberFormats="0" applyBorderFormats="0"   
 applyFontFormats="0" applyPatternFormats="0" applyAlignmentFormats="0"   
 applyWidthHeightFormats="0">

<queryTableRefresh nextId="6" unboundColumnsRight="1">  
 <queryTableFields count="4">  
 <queryTableField id="1" name="ID" tableColumnId="1"/>  
 <queryTableField id="2" name="Field1" tableColumnId="2"/>  
 <queryTableField id="3" name="Field2" tableColumnId="3"/>  
 <queryTableField id="5" dataBound="0" tableColumnId="4"/>  
 </queryTableFields>

<queryTableDeletedFields count="1">  
 <deletedField name="Field3"/>  
 </queryTableDeletedFields>  
 </queryTableRefresh>  
</queryTable>

In the context of queryTableRefresh:

* 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.
* 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)

In the context of queryTableFields:

* Each of the queryTableField elements expresses information about one of the columns in the Table that is part of the QueryTable. For example, the right-most column's dataBound is set to 0, indicating that this column is not bound to external data.

In the context of queryTable:

* queryTableDeletedFields collection expresses which fields returned by the connection have been deleted from the QueryTable. This is tracked so that the connection information does not have to be updated with which columns are no longer required.

## Volatile Dependencies

Yet to be supplied

## Custom XML Mappings

### Overview

With the pervasiveness of XML data structures and XML web services, it is appropriate for a spreadsheet application to consume XML data structures and render the data in the sheet grid. Furthermore, it is appropriate and desirable for the spreadsheet application to be able to generate XML data structures. Finally, since XML is extensible, the kinds of XML structures that can be consumed or produced by a spreadsheet application should be as varied as the number of XML schemas that exist.

The XML Mapping feature enables the addition of 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.

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.

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.

Additional properties are stored on each cell and each column of a Table that has an XML map association.

### File Architecture: Relationships

The solid lines represent relationships. The workbook owns sheets and XML map definitions. Each sheet references Tables and single cells that are mapped to XML structures.

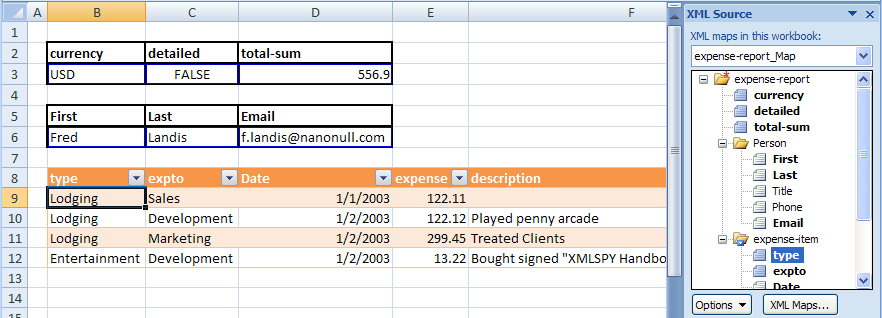
### Conceptual Model

Conceptually, all the objects reference a common Map Object, by id. It is in this way that they come together into a single working feature that can import and export custom XML data.

### Example

In the following example, both single cells and a Table are shown to have a data binding to an XML structure.

#### Illustration



The table in B8:G12 is also data bound to an XML mapping object. The first column, titled "type", is associated with the XML Map named "expense-report\_Map", specifically the attribute identified by the xpath expression /expense-report/expense@type pointing into the corresponding XML structure. In similar fashion, each of the columns in the Table correspond with elements or attributes in the related XML Map structure.

Additionally, cells B3:D3 and B6:D6 are each bound to a single, non-repeating element or attribute from the same XML Map structure. For example, cell B3 corresponds to /expense-report@currency.

In this way XML, instance structures can be refreshed into the cells and Table region, and XML instance structures can be generated from the data in those ranges of the spreadsheet. In other words, XML structures can be imported and exported to and from the worksheet via the XML Mapping feature.

#### The xmlMap XML

The Custom XML Mappings part, xmlMaps, part stores the custom schema that has been added to the workbook, and stores the xmlMap definitions. There can be multiple schemas and xmlMaps in a single workbook.

<MapInfo SelectionNamespaces="">  
 <Schema ID="Schema1">  
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
 <xsd:element nillable="true" name="expense-report">  
 <xsd:complexType>  
 <xsd:sequence minOccurs="0">

<xsd:element minOccurs="0" nillable="true" name="Person"  
 form="unqualified">  
 <xsd:complexType>  
 <xsd:sequence minOccurs="0">  
 <xsd:element minOccurs="0" nillable="true"  
 type="xsd:string" name="First"   
 form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true"  
 type="xsd:string" name="Last"   
 form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true"  
 type="xsd:string" name="Title"   
 form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true"  
 type="xsd:string" name="Phone"   
 form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true"  
 type="xsd:string" name="Email"   
 form="unqualified"></xsd:element>  
 </xsd:sequence>  
 </xsd:complexType>  
 </xsd:element>

<xsd:element minOccurs="0" maxOccurs="unbounded"  
 nillable="true"  
 name="expense-item" form="unqualified">  
 <xsd:complexType>  
 <xsd:all>  
 <xsd:element minOccurs="0" nillable="true"  
 type="xsd:date"  
 name="Date" form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true"  
 type="xsd:double" name="expense"   
 form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true"  
 type="xsd:string" name="description"   
 form="unqualified"></xsd:element>

<xsd:element minOccurs="0" nillable="true" name="Misc"  
 form="unqualified">  
 <xsd:complexType>  
 <xsd:attribute name="misctype" form="unqualified"  
 type="xsd:string"></xsd:attribute>  
 </xsd:complexType>  
 </xsd:element>  
 </xsd:all>

<xsd:attribute name="type" form="unqualified"  
 type="xsd:string"></xsd:attribute>

<xsd:attribute name="expto" form="unqualified"  
 type="xsd:string"></xsd:attribute>  
 </xsd:complexType>  
 </xsd:element>  
 </xsd:sequence>

<xsd:attribute name="currency" form="unqualified"  
 type="xsd:string"></xsd:attribute>

<xsd:attribute name="detailed" form="unqualified"  
 type="xsd:boolean"></xsd:attribute>

<xsd:attribute name="total-sum" form="unqualified"  
 type="xsd:double"></xsd:attribute>  
 </xsd:complexType>  
 </xsd:element>  
 </xsd:schema>  
 </Schema>

<Map ID="1" Name="expense-report\_Map" RootElement="expense-report"  
 SchemaID="Schema1" ShowImportExportValidationErrors="false"  
 AutoFit="true" Append="false" PreserveSortAFLayout="true"   
 PreserveFormat="true">  
 <DataBinding FileBinding="true" DataBindingLoadMode="1"/>  
 </Map>  
</MapInfo>

In the context of MapInfo:

* SelectionNamespaces ties the prefix to the actual namespace. This is used when writing xpath expressions at runtime against the XML instance structures, because the xpath expressions use namespace prefixes instead of the fully spelled out namespace.
* Schema stores the schemas for a particular XML map object. There can be multiple Schema elements in a workbook, one for each XML map.

In the context of Schema:

* ID identifies the schema collection used to define a particular XML map object.

In the context of Map:

* D identifies the map object.
* Name is the friendly name of the map object.
* RootElement is the name of the root element of the XML instance (schemas can define more than one root node).
* SchemaID identifies which schema collection the map uses.
* ShowImportExportValidationErrors indicates that when an XML instance is imported or exported, the schema should be used to validate the instance, and schema errors should be shown to the user.
* AutoFit indicates that after refresh, all the cells should be ‘best fitted’.
* Append means that when refreshed, don’t discard existing data, but append new data to it.
* PreserveSortAFLayout indicates whether to keep filters on (Tables).
* PreserveFormat indicates whether to keep the cell formatting applied or re-apply based on schema data type.

#### 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 that has an XML data binding.

<table xmlns="…" id="7" name="Table7" displayName="Table7" ref="B8:G12"  
 tableType="xml" totalsRowShown="0" connectionId="1">  
 <autoFilter ref="B8:G12"/>  
 <tableColumns count="6">  
 <tableColumn id="1" uniqueName="type" name="type">  
 <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/@type"   
 xmlDataType="string"/>  
 </tableColumn>

<tableColumn id="2" uniqueName="expto" name="expto">  
 <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/@expto"   
 xmlDataType="string"/>  
 </tableColumn>

<tableColumn id="3" uniqueName="Date" name="Date">  
 <xmlColumnPr mapId="1" xpath="/expense-report/expense-item/Date"   
 xmlDataType="date"/>  
 </tableColumn>

<tableColumn id="4" uniqueName="expense" name="expense">  
 <xmlColumnPr mapId="1"  
 xpath="/expense-report/expense-item/expense"   
 xmlDataType="double"/>  
 </tableColumn>

<tableColumn id="5" uniqueName="description" name="description">  
 <xmlColumnPr mapId="1"  
 xpath="/expense-report/expense-item/description"  
 xmlDataType="string"/>  
 </tableColumn>

<tableColumn id="6" uniqueName="misctype" name="misctype">  
 <xmlColumnPr mapId="1"  
 xpath="/expense-report/expense-item/Misc/@misctype"  
 xmlDataType="string"/>  
 </tableColumn>  
 </tableColumns>

<tableStyleInfo name="TableStyleMedium7" showFirstColumn="0"  
 showLastColumn="0" showRowStripes="1" showColumnStripes="0"/>  
</table>

The column in the Table titled "type" is bound to an XML mapping, whose map object I mapId is 1 The xpath value indicates an xpath expression to which this Table column is associated. In this example, the Table column "type" corresponds to type, which is an attribute of the expense-item collection. The corresponding custom schema definition for type indicates a data type of string. This is stored as an xml column property as well, in xmlDataType. This is used for interpreting the data on import and export, and to format the cells for proper rendering in the range.

The remaining columns have similar properties set and can be understood from the discussion above.

#### Single Cell XML

The contents of Single Cell Table Definitions part, tableSingleCells.xml, are:

<singleXmlCells xmlns="…">  
 <singleXmlCell id="1" name="Table1" displayName="Table1" r="B3"  
 connectionId="1">  
 <xmlCellPr id="1" uniqueName="currency">  
 <xmlPr mapId="1" xpath="/expense-report/@currency"   
 xmlDataType="string"/>  
 </xmlCellPr>  
 </singleXmlCell>

<singleXmlCell id="2" name="Table2" displayName="Table2" r="C3"  
 connectionId="1">  
 <xmlCellPr id="1" uniqueName="detailed">  
 <xmlPr mapId="1" xpath="/expense-report/@detailed"   
 xmlDataType="boolean"/>  
 </xmlCellPr>  
 </singleXmlCell>

<singleXmlCell id="3" name="Table3" displayName="Table3" r="D3"  
 connectionId="1">  
 <xmlCellPr id="1" uniqueName="total-sum">  
 <xmlPr mapId="1" xpath="/expense-report/@total-sum"   
 xmlDataType="double"/>  
 </xmlCellPr>  
 </singleXmlCell>

<singleXmlCell id="4" name="Table4" displayName="Table4" r="B6"  
 connectionId="1">  
 <xmlCellPr id="1" uniqueName="First">  
 <xmlPr mapId="1" xpath="/expense-report/Person/First"   
 xmlDataType="string"/>  
 </xmlCellPr>  
 </singleXmlCell>

<singleXmlCell id="5" name="Table5" displayName="Table5" r="C6"  
 connectionId="1">  
 <xmlCellPr id="1" uniqueName="Last">  
 <xmlPr mapId="1" xpath="/expense-report/Person/Last"   
 xmlDataType="string"/>  
 </xmlCellPr>  
 </singleXmlCell>

<singleXmlCell id="6" name="Table6" displayName="Table6" r="D6"  
 connectionId="1">  
 <xmlCellPr id="1" uniqueName="Email">  
 <xmlPr mapId="1" xpath="/expense-report/Person/Email"   
 xmlDataType="string"/>  
 </xmlCellPr>  
 </singleXmlCell>  
</singleXmlCells>

A single cell that has been mapped to an XML node is expressed in much the same way that an entire table is expressed.

The singleXmlCell collection is the top-level object, like the Table, which identifies the cell in question.

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.

The xmlPr collection expresses the xml properties for this cell.

End of informative text.

# Introduction to PresentationML

This clause is informative.

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

The PresentationML file format can be broken down into the following subjects:

* Presentation
* Slides
* Slide Content
* Animation

There are other schemas—most notably DrawingML—that make up a sizeable chunk of the PresentationML file format. These schemas are addressed separately in §5.

## Presentation

### Basic Utilities

PresentationML defines a set of complex types and simple types that are used by other schemas. The types—or utilities—are used in a variety of cases. Their single implementation provides for rapid and less error-prone changes throughout an implementation.

To provide some insight into the type of information that is being repurposed, here is a sample of these utilities:

* Empty Element (§4.1.2)
* Name (§4.1.3)
* Direction (§4.1.4)
* Index and Index Range (§4.1.5)
* Slide Show ID (§4.1.6)
* Slide List Choice (§4.1.7)
* Slide Relationship (§4.1.8)
* Customer Data (§4.1.9)
* Future Extensibility (§4.1.10)

Each of these is discussed in the following subclauses.

### Empty Element

Sometimes the simple presence of an element is sufficient to convey meaning. That is, in some cases, information need not necessarily have a Boolean, an integer, or complex type.

A simple example is the ShowType element group. In this case, a slide show can be one of three types: present, browse, or kiosk (all of which are defined in §4.1.17). The schema for this element group is as follows:

<xsd:group name="EG\_ShowType">  
 <xsd:choice>  
 <xsd:element name="present" type="CT\_Empty"/>  
 <xsd:element name="browse" type="CT\_ShowInfoBrowse"/>  
 <xsd:element name="kiosk" type="CT\_Empty"/>  
 </xsd:choice>  
</xsd:group>

Naturally, this schema can result in the following XML:

<AR:INCLUDE XML HERE>

The presence of the present element is sufficient to declare that this slide show is of the “present” type.

### Name

Many constructs within a presentation have names associated with them. In some cases, these names are machine-generated, such as shape names (e.g., rectangle1), while others are user-defined, such as slide shows (e.g., customer-ready).

### Direction

This multi-purpose simple type is used to convey horizontal versus vertical direction of a variety of types. Such usage can be found in the definition of slide transitions and various shape effects.

### Index and Index Range

These two utilities are generally used to denote a contiguous set of items within a list. The classic example of usage would be the selection of a set of slides to be printed.

From a schema-perspective, there is no way to enforce that the start index by equal to or less than the end index; rather, this is a function of business logic within the application.

### Slide Show ID

Somewhat related to the name simple type is the simple type that defines the ID for a slide show (also called a custom show). Because a slide show can be named, and that name can change, an implementation needs a method of referring to a slide show, that can withstand name changes made by the user. In many cases, for example, with a slide, the fact that each slide has a part within the package can be leveraged, in which case, the relationship ID can be used. However, since there is no slide show-specific part, an unsigned integer is generated for each slide show, and that number is used.

There is nothing in the schema that prevents two or more slide shows from having the same ID; error handling for this case is implemented in the business logic of the application.

### Slide List Choice

There are many cases in which a user needs to specify a set of slides for an operation. The canonical example is the set of slides to be included in a slide show.

<xsd:group name="EG\_SlideListChoice">  
 <xsd:choice>  
 <xsd:element name="sldAll" type="CT\_Empty" />  
 <xsd:element name="sldRg" type="CT\_IndexRange" />  
 <xsd:element name="custShow" type="CT\_CustomShowId" />  
 </xsd:choice>  
</xsd:group>

As the schema above declares, when selecting a set of slides, the user can select all of the slides, a slide range (by declaring a pair of start and end indices) or a particular custom show.

### Slide Relationship

As described in §4.1.6, there are many situations where the format needs to store an ordered list of slides, and does so by storing their slide IDs. This is implemented using two types: a list entry complex type and a list complex type:

<xsd:complexType name="CT\_SlideRelationshipListEntry">  
 <xsd:attribute ref="r:id" use="required"/>  
</xsd:complexType>

<xsd:complexType name="CT\_SlideRelationshipList">  
 <xsd:sequence>  
 <xsd:element name="sld" type="CT\_SlideRelationshipListEntry"  
 minOccurs="0" maxOccurs="unbounded"/>  
 </xsd:sequence>  
</xsd:complexType>

### Customer Data

There is a set of utilities that facilitate the storage of customer XML data within the file format. Essentially, this functionality comes down to the ability to store customer-defined XML in the file format in a way that it can be easily queried, modified and/or surfaced in the presentation. Suffice it to say, the data is stored in a separate part within the package, and hence the utility pairs the object using it with that part.

### Future Extensibility

Functionality exists to provide the ability to extend a subset of objects within the file format for inclusion of additional data over the lifetime of the file format. The utilities provide both the ability to add an alternative representation (e.g., provide a raster image in addition to the XML data for a diagram) as well as additional properties to the objects.

### The Presentation Object

The schema pPresentation.xsd defines the content of the start part for a PresentationML document. This content includes both structural and presentation-level data for the presentation.

It might appear that there is an apparent duplication of presentation-level data, as there is also a separate schema file, pPresProps.xsd, which contains presentation-level data. However, there is no duplication. Rather, the differentiation of what presentation-level data goes into which part is based on two user scenarios: document signatures and document sanitization.

In a document signature scenario, assume a user digitally signs a presentation. There exist two types of data within the presentation package: data which changes the “content” of the presentation, and data which is intended to configure an editor or the behavior of an editor. In the first case, any modification to data which changes the “content” of the presentation must invalidate the signature; in the second case, any modification to that data should not invalidate the signature.

An example of this scenario deals with Kinsoku information and the publish path in the HTML settings. If the user changes the Kinsoku information in a file, that file will look (and potentially mean) something different. This is in contrast to a user setting a new HTML publish path for his particular computer.

In a document sanitization scenario, users want to remove all non-necessary information from the file. A typical usage case would be posting a presentation to a company’s Internet site. In this case, certain configuration information should not be made available publicly. The ideal manner of removal would be to remove an entire part from the presentation package, as opposed to editing a part within a package.

Going back to the Kinsoku and HTML publish path example above, the Kinsoku information needs to remain with the file. The HTML publish path could give away internal information about web servers that could be used to facilitate an attack or, more likely, simply provide information about the author to the public (e.g., the path c:\documents and settings\shawnv\webpages strongly implies that “shawnv” published this document).

Going back to the original question—what presentation-level data goes in which part—data that will not invalidate a digital signature, or data that should be removed during a sanitization pass is stored in the part associated with the pPresProps.xsd schema, and other presentation-level data is stored in the part associated with the pPresentation.xsd schema.

In addition to structural and presentation-level data defined by this schema, there are also definitions for handling customer data (§4.1.9) and future extensibility (§4.1.10).

### Structural Information

From a structural information perspective, there are two sets of data defined in this schema: core lists and sizes.

The schema first defines a number of lists that serve as the foundation for most objects in the presentation. These lists are as follows:

* Slide IDs
* Slide Masters
* Notes Masters
* Handout Masters
* Custom Shows

It is essential that the reader fully understand the implementation of usage of these lists, as they are the foundation for almost all solutions that operate—open, interrogate, modify, write—against the PresentationML file format.

As mentioned above, the lists are defined as a part of list entry and list complex types. The slide master list is defined as follows:

<xsd:complexType name="CT\_SlideMasterIdListEntry">  
 <xsd:attribute ref="r:id" use="required" />  
</xsd:complexType>

<xsd:complexType name="CT\_SlideMasterIdList">  
 <xsd:sequence>  
 <xsd:element name="sldMasterId" type="CT\_SlideMasterIdListEntry"  
 minOccurs="0" maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

Although not complex or difficult to understand, the lists are called out because they are vital to any solution.

The next pieces of structural information are the sizes for the slides and the notes slides. By storing this information at the presentation level, the implication is that all slides (or all notes slides) in a presentation have the same size. This further implies that all slides in a presentation share the same orientation (i.e., they are all landscape-oriented or all portrait-oriented).

### Presentation-Level Properties

The presentation-level properties defined in this schema can be grouped as follows:

* Text-Related (§4.1.13.1)
* Save-Related (§4.1.13.2)
* Editor-Related (§4.1.13.3)
* Content-Related (§4.1.13.4)

A description of the properties within each of these groups follows.

#### Text-Related Properties

The first property stores information related to the Kinsoku settings. Kinsoku settings define the list of characters that are not allowed to start or end a line of text for a given East Asian language.

The schema definition of the Kinsoku settings is relatively straightforward: identify the language, the set of invalid start characters, and the set of invalid end characters:

<xsd:complexType name="CT\_Kinsoku">  
 <xsd:attribute name="lang" type="xsd:string" use="optional">  
 </xsd:attribute>  
 <xsd:attribute name="invalStChars" type="xsd:string" use="required">  
 </xsd:attribute>  
 <xsd:attribute name="invalEndChars" type="xsd:string" use="required">  
 </xsd:attribute>  
</xsd:complexType>

The second property stores a flag to use strict characters for starting and ending a line of Japanese text. This is a simple Boolean attribute:

<xsd:attribute name="strictFirstAndLastChars"  
 type="xsd:boolean" use="optional" default="true"/>

The final text-related property stores information related to any fonts that are embedded in the presentation. To do this, a list of embedded fonts that reference each part that stores font data (generally, there is a one-font-to-one-part mapping, although this is not a strict rule) must be stored. This information is defined using three complex types:

<xsd:complexType name="CT\_EmbeddedFontList">  
 <xsd:sequence>  
 <xsd:element name="embeddedFont" type="CT\_EmbeddedFontListEntry"  
 minOccurs="0" maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

<xsd:complexType name="CT\_EmbeddedFontListEntry">  
 <xsd:sequence>  
 <xsd:element name="font" type="a:CT\_TextFont" minOccurs="1"  
 maxOccurs="1" />  
 <xsd:element name="regular" type="CT\_EmbeddedFontDataId"  
 minOccurs="0" maxOccurs="1"/>  
 <xsd:element name="bold" type="CT\_EmbeddedFontDataId"  
 minOccurs="0" maxOccurs="1"/>  
 <xsd:element name="italic" type="CT\_EmbeddedFontDataId"  
 minOccurs="0" maxOccurs="1"/>  
 <xsd:element name="boldItalic" type="CT\_EmbeddedFontDataId"  
 minOccurs="0" maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

<xsd:complexType name="CT\_EmbeddedFontDataId" >  
 <xsd:attribute ref="r:id" use="required"/>  
</xsd:complexType>

#### Save-Related Properties

This set of properties indicates to the editor what should be saved as part of the presentation.

The first property controls the inclusion of Personally Identifiable Information (PII). PII is any information that can be used to identify the author or contributor to a presentation. And while there are cases where this information is exposed visually to the user (e.g., author name in a comment shape), there are other cases where the information is not immediately evident to the user (e.g., the document author name in the list of document properties).

An implementation can provide a mechanism by which the author of a presentation can configure a file to always remove any PII that might otherwise by normally included during a regular save operation. While not a guarantee that no PII is stored in the file (e.g., consider a shape with some person's name in it—in some cases it describes content in the file [that person's position in their group’s organization chart] whereas in others it is an editorial directive [“check with ShawnV on this point”]). Given this ambiguity, it is not possible to solve all cases of this. As a result, this is more a convenience feature than a privacy management feature.

The second set of properties has two groupings of properties. The first controls whether or not fonts will be embedded into the package representing the presentation. The second, enabled by setting the first, allows an implementation to optimize such font embedding to keep the size minimal, at the cost of future editing on other machines.

<xsd:attribute name="embedTrueTypeFonts" type="xsd:boolean"  
 use="optional" default="false"/>

<xsd:attribute name="saveSubsetFonts" type="xsd:boolean"  
 use="optional" default="false"/>

The user scenario behind these properties is as follows. Assume you are putting together a presentation to distribute to external customers. You happen to use an East Asian font with an on-disk file size of around 5 megabytes.

Assuming that this font is not a standard font that is widely distributed, not including this font will cause font substitution when the presentation is opened on machines that don’t have a copy of the font. In any case, this can radically change the visual appearance of the presentation; in some cases, it can render the presentation unreadable.

Because you cannot afford the presentation to be unreadable or to look unprofessional, you decide to embed the font. By default, the implementation will set embedTrueTypeFonts to true and embed the entire 5 megabyte font file in the presentation package. This will clearly bloat the file, but will ensure that anyone viewing or editing this file will have the same font experience as you originally had (subject to licensing restrictions, of course).

Since you are distributing the presentation, and your primary purpose is for people to view the presentation, you can reduce the amount of font data embedded in the presentation package. By setting the second property, saveSubsetFonts, to true, only those characters in the font that were actually used to create the presentation are saved. This yields less font data stored in the file at the cost of not being able to use unused characters in future edits of the presentation on different machines.

The third property related to saving, controls whether or not an implementation can automatically compress pictures contained in the presentation. This is particularly important given the proliferation of digital cameras and scanners and the increasing importance of small files (e.g., to save network bandwidth, reduce storage required for mail and file servers, etc.).

The final property in this set specifies a password that is required to enable editing of the file using the implementation. Because this is a convenience feature intended to prevent accidental changes to information, it is stored in clear text as an xsd:string.

By storing this information in the file, the implementation will prompt the user for this password in order to open the file read/write; if the user does not provide the correct modify password, the implementation will open the file read-only.

#### Editor-Related Properties

The presentation file itself contains data that provide configuration information for the implementation’s editor.

PresentationML objects can be embedded into OLE containers, during which time a customer can set a zoom scale. This is stored in the file as a percentage called serverZoom.

An implementation can be an internationalized application, in which case, it supports configuring the editor to respect different screen orientations. For example, in regions of the world where Complex Scripts are in use, it is customary to orient the screen right-to-left. As such, a presentation can request the editor reconfigure itself for such usage scenarios.

#### Content-Related Properties

This set of properties is related to the actual content in the presentation.

End-users can define the starting slide number for numbered slides in each presentation. While it typically starts at one, the user can select any positive number. The primary user scenario involves compiling a presentation that is really a collection of multiple presentations. A secondary user scenario involves including a presentation in the middle of, or at the end of, a printed document where you want the slide/page numbers to continue.

Another content-related property controls whether or not header/footer placeholders are to be shown on title slides. In many cases, users will use special shapes called header and footer placeholder that contain built-in field codes that control the display of various sorts of information like the date/time and slide number.

In most cases, users like to keep their title slides as simple as possible (much like in the printed world where you want your first page to be clean and streamlined) and hence do not want data like date/times and slide numbers to show up on such slides. This attribute defines this presentation-wide.

The final property relates to creating photo albums. The implementation has a feature that allows the user to generate automatically a presentation based on a set of pictures. During this process, the user can select from a variety of settings, including, but not limited to, which pictures to include, the layout of the pictures on the slides (e.g., one picture per slide, two pictures per slide, etc.), what type of frame shape to use, and so on. All of this information is stored in the presentation for future photo album creation.

### Presentation Properties

Those properties that apply to the presentation as a whole, and that are likely to be removed during document sanitization, or are not going to invalidate a digital signature, are defined in pPresProps.xsd. These properties can be grouped into three primary groupings.

* HTML Publish Properties (§4.1.15)
* Print Options Properties (§4.1.16)
* Slide Show Properties (§4.1.17)

In addition to this grouping, there are properties that define a Most Recently Used (MRU) list of colors as well as providing for future extensibility.

### HTML Publish Properties

An implementation must have the ability to save (and publish) a presentation to a web-friendly format like HTML or MHTML. Various parameters are used to configure the application for saving such formats as well as to control what content gets generated. The parameters that configure the application are the HTML Publish properties whereas the content properties are the Web Properties (§4.1.15.1).

The HTML Publish properties provide the author with the ability to control what content gets displayed in the browser when the resulting file—either HTML or MHTML—is viewed using that type of an application. For example, the speaker notes can either be displayed in the frameset or can be hidden from view. This is particularly useful when a speaker’s notes are not necessarily in a customer-ready format. It’s useful but not necessarily secure.

The author can also specify the title to be displayed in the browser. Although this defaults to the actual file name, or if that is missing, to the content of the first slide’s title placeholder, it can be overridden by the author.

Finally the author can specify a publish path to use when saving this file in this format. This is particularly useful for two reasons.

First, because there is a transformation happening, it sometimes takes a few iterations of publishing to get the browser-based experience to be exactly what you want. A classic example of this is the differing animation capabilities between the implementation and certain browsers: it is important to verify that the change in animation behavior is after publishing; if you are not satisfied with the experience, sometimes you need to change the animation in the implementation and republish.

The second reason storing the path is useful is that web server paths can be cumbersome and are often not on the tip of each user’s tongue. This allows the user to specify the path once and then publish using the same location without having to re-specify it. Naturally, being stored in the file format, this allows this data to persist across session.

Indirectly, the HTML Publish properties can prime the Web Properties by defining a target web browser generation (i.e., third, fourth or third and fourth). This is done by setting the appropriate ST\_HtmlPublishWebBrowserSupport attribute:

<xsd:complexType name="CT\_HtmlPublishProperties">  
 <xsd:sequence>  
 <xsd:group ref="EG\_SlideListChoice" minOccurs="1" maxOccurs="1">  
 </xsd:group>  
 </xsd:sequence>

<xsd:attribute name="showSpeakerNotes" type="xsd:boolean"  
 use="optional" default="true" />

<xsd:attribute name="pubBrowser" type="ST\_HtmlPublishWebBrowserSupport"  
 use="optional" default="v3v4" />

<xsd:attribute name="title" type="xsd:string" use="optional" default="">  
 </xsd:attribute>

<xsd:attribute ref="r:id" use="required">  
 </xsd:attribute>  
</xsd:complexType>

By providing a target generation, the Web Properties will be set to a predefined package defined for the specified browser generation. Naturally, the user can override the individual Web Property settings.

#### Web Properties

As mentioned in the previous subclause, these properties configure the output of the presentation when saved using the HTML or MHTML formats. In this case, a number of parameters can be controlled.

In all multi-slide cases where the presentation is saved using one of these formats, the implementation will create a frameset to bring the various parts of a presentation—the slide content, the speaker notes and the outline—together as well as provide for simple navigation. The color of the HTML frames, the background used and the user interface controls can be controlled to leverage browser settings, use high contrast, and so on.

The author can also control how much interactivity will be exposed in the resulting output. For example, the user may elect to disable slide animations and transitions, and opt for a more static presentation. Similarly, the author may elect to disable certain scripting features like the ability to resize dynamically the output to match the size of the browser window.

Somewhat related to this is the ability to specify the target screen size which is especially important when targeting the earlier browser generations or user environments where features like JavaScript are disabled.

Given the potential international usage of an implementation as well as the need for users to distribute widely their presentations internationally, there is the ability to control the encoding of text used in the generation of the HTML or MHTML output.

Finally, there is a set of parameters that is used to configure the on-disk storage of the resulting output. For example, if the customer knows something about the machine configurations of her audience, she can opt to use better raster graphic formats like PNG that support alpha transparency or elect to include VML representations only for vector images.

The customer can also provide some indication as to how the output will be used. If the customer knows that the output will be used like regular files (perhaps passed around on CDs or moved between file shares) the user may elect to store the files in a folder to ensure that a straggling file is not lost; if, however, the target scenario is to put the files on a web server, the user can skip the folder and save the individual files in a flat directory. Similarly, if the customer knows that they are using a web server that only handles “8.3” file names, they can configure the implementation to generate files using names that are “8.3”-compliant, as opposed to using long file names that may otherwise cause such web servers problems.

### Print Options Properties

This set of properties controls the default print behavior for a presentation. The inclusion of this information in the file format simply primes any print-related dialog when this presentation is used. It does not force options nor does it represent the last-used set of print options for a presentation.

Using these properties, the author can control the type of output printed. For example, in some cases, authors need to print their slides (one slide per printed page) while in other cases, they want to provide printed handouts for the audience on which to take their own notes (handout pages that can contain anywhere from three to nine slides per printed page, as well as optional lines for note taking). In other cases, the author would like to print out notes where each printed page has one slide (anchored at the top) and a text box (anchored at the bottom) with the speaker notes included, or simply print the textual outline of the presentation.

The author can also control whether or not hidden slides are included in the printed output, as well as whether or not the output is sent to the printer in color, in grayscale, or in pure black and white.

There is also a set of properties that the author can set that determine if slides are framed on the printed page, if the slides are scaled up to the printed page (e.g., consider non 4x3 aspect ratio slides), and so on.

### Slide Show Properties

This set of presentation-level properties controls the default slide show.

Among the parameters that can be controlled is one that defines the type of slide show. Generally, the classic slide show is characterized by a presenter showing the presentation to an audience. The presenter controls the flow of the presentation. This is referred to as a present slide show. In some cases, however, the presentation is distributed and individuals walk themselves through the slide show. This is referred to as a browse slide show. Finally, there are cases where a slide show is prepackaged and used as a kiosk; naturally, it is referred to as a kiosk slide show.

Furthermore, the customer can control which slides are to be included in the slide show, what color the pen should be, and so on.

Finally, the customer can control various interactivity settings that are to be used for the slide show. This provides the customer the ability to configure their slide show outside the typical settings for a particular slide show type. For example, the user may create a slide show that has a pre-configured animation built with timings (i.e., the time between particular builds or the time between slide transitions), even though she is going to be presenting the content to an audience.

### View Properties

The schema pViewProps.xsd defines the properties on all of the views found in the implementation.

There are two types of views: simple views and view sets. Whereas simple views are views focused on a particular object type such as a slide, notes text or the outline, view sets are more complex views built using two or more simple views.

Simple views have a common set of properties that are defined in CT\_CommonViewProperties. These properties control the zoom scale as well as the quality of the rendering within the view.

Each simple view has a scale factor, expressed as a horizontal and vertical ratio, and there is always an origin for the view.

In cases where the content should be scaled to take as much space as possible, a flag called varScale can be set that will tell an implementation to use a variable scale.

The final common property is a flag called draftMode, which tells an implementation to draw the view's content as fast as possible at the expense of non-WYSIWYG rendering.

But some simple views are more complex; in particular, views that are based on slides have an additional set of common properties. These additional common properties, defined in CT\_CommonSlideViewProperties, define guides and control various snap-to-grid behaviors.

Guides are defined as either horizontal or vertical (that is, they have an orientation), and are positioned on a slide surface using a one dimensional unit along the appropriate axis. Naturally, slides can have zero or more guides, and control whether all or none of the guides are shown.

The snap functionality tells an implementation whether or not drawn or moved objects should snap to the grid or to other objects.

The set of simple views includes the following five views:

* Slide View
* Outline View
* Notes Text View
* Slide Sorter View
* Notes View

Normal View is the default view for presentations. This view is actually a view set made up of slide view, notes text view and outline view.

View sets also have an additional set of properties that relate to how the simple views are composed into a single view set.

Each simple view is separate from another simple view, in a view set, by a splitter bar. The splitter bar's primary role is to determine how much screen size each view gets. These bars can be minimized or maximized; further, they can be restored to their non-extreme sizes.

View sets also have more advanced properties that control other view-specific properties such as whether or not slide icons will be shown in the outline view when the view is significantly shrunken.

View sets also have the capability of reducing the view set to a single, simple view, which generally takes the form of a slide view. This support is most often used by presentation authors when they are about to distribute their presentation to viewers who they do not want, or do not perceive the need to, see the presentation content in a form other than slide view.

All of the view data for a presentation is stored together in a list of elements containing the properties of each of the views. This storage also contains data like the last view in which the presentation was saved in as well as whether or not comments are currently visible. Finally, there is information regarding grid spacing.

## Comments

### Introduction

This document describes the commenting feature for presentations as expressed in PresentationML. The schema that defines this feature is pComments.xsd

### Functional Overview

Readers of a presentation can provide feedback to the presentation author in the form of comments. Comments can only be applied to slides; they cannot be applied to masters of any type or to notes slides.

At first glance, comments appear to be shapes on the slide surface; however, they are not. Comments differ from regular shapes in two ways:

* Comments cannot be formatted or resized
* The text contained within a comment cannot be formatted

### Comment Author List

Presentations contain a list of all authors who have comments in the presentation. This list is commonly referred to as the Comment Author List (CAL). The CAL contains one entry for each author. Each entry is made up of five pieces of data: ID, Author Name, Author Initials, Last Index, and Color Index.

Each author that comments on a presentation is assigned an ID, which is a simple integer. This ID is unique within the presentation, and is assigned by the application itself.

The Author Name and Author Initials are taken from the application itself. If no initials are known to the application, the comment author is prompted upon the insertion of the initial comment. Both the Author Name and Author Initials are simple strings; that is, there is no association of the values with an identity (from a security or authentication perspective).

The Last Index (lastIdx) is an integer that documents how many comments the associated author has made in this presentation. When the author makes another comment, that comment is numbered using the next integer, and then this value is updated once again.

The Color Index (clrIdx) is an integer into a color table that is used to provide the solid background fill for the comment shape. The utility that this provides is that all of the comments by a particular author share the same color.

Here is an example of such a CAL:

<p:cmAuthorLst>  
 <p:cmAuthor id="0" name="Shawn" initials="SV" lastIdx="3" clrIdx="0" />   
 <p:cmAuthor id="1" name="Brian" initials="BJ" lastIdx="7" clrIdx="1" />   
</p:cmAuthorLst>

To determine if an author is already in the CAL, one must consider only the Author Name and Author Initials data. If they both match an entry in the CAL, the author is already considered to be in the CAL; otherwise, the author is considered unique, and a separate entry is added for that author in the CAL.

When the presentation is saved using PresentationML, a separate Comment Authors part is created that contains the CAL.

### Comment List

Each slide within a presentation may contain zero or more comments. Each slide with at least one comment starts a list of comments for that slide. Each entry in that list is made up of the following pieces of data:

* Author ID: This represents the ID of the author who created the comment. It matches an entry in the CAL.
* Date/Time: This represents the date and time of the last modification of this particular comment. Although expressed in UTC, its accuracy is dependent on the state of the machine making the edits.
* Index: This is the number assigned to this particular comment, and is one of the comments associated with the specified author. This number should be equal to, or less than, the Last Index value for the author in the CAL. There cannot be duplicate Indexes for the same author.
* Position: This defines the 2D coordinate for the location at which the comment shows up on the slide surface. This is the position of the upper left point of the comment shape.
* The Text data includes all of the text that makes up the body of the comment. Note that this text is expressed differently than other text as expressed in DrawingML. As this text contains no formatting, and is strictly limited to text input, there is no additional data that needs to be stored.

Here is an example of a comment list for a slide:

<p:cmLst>  
 <p:cm authorId="0" dt="2006-01-30T22:45:13.597" idx="3">  
 <p:pos x="10" y="10" />   
 <p:text>Need to check with Mary on exact data values</p:text>   
 </p:cm>

<p:cm authorId="1" dt="2006-01-30T22:46:22.082" idx="1">  
 <p:pos x="106" y="106" />   
 <p:text>This chart is hard to read from afar</p:text>  
 </p:cm>  
</p:cmLst>

When the presentation is saved using PresentationML, a separate Comments part is created for each comment list.

## Slides, Masters, Layouts, and Placeholders

Yet to be supplied

## Future Extensibility

Yet to be supplied

## Slide Synchronization

This subclause provides an overview of the p:SldSyncPr element in pml-slidesyncinfo.xsd.

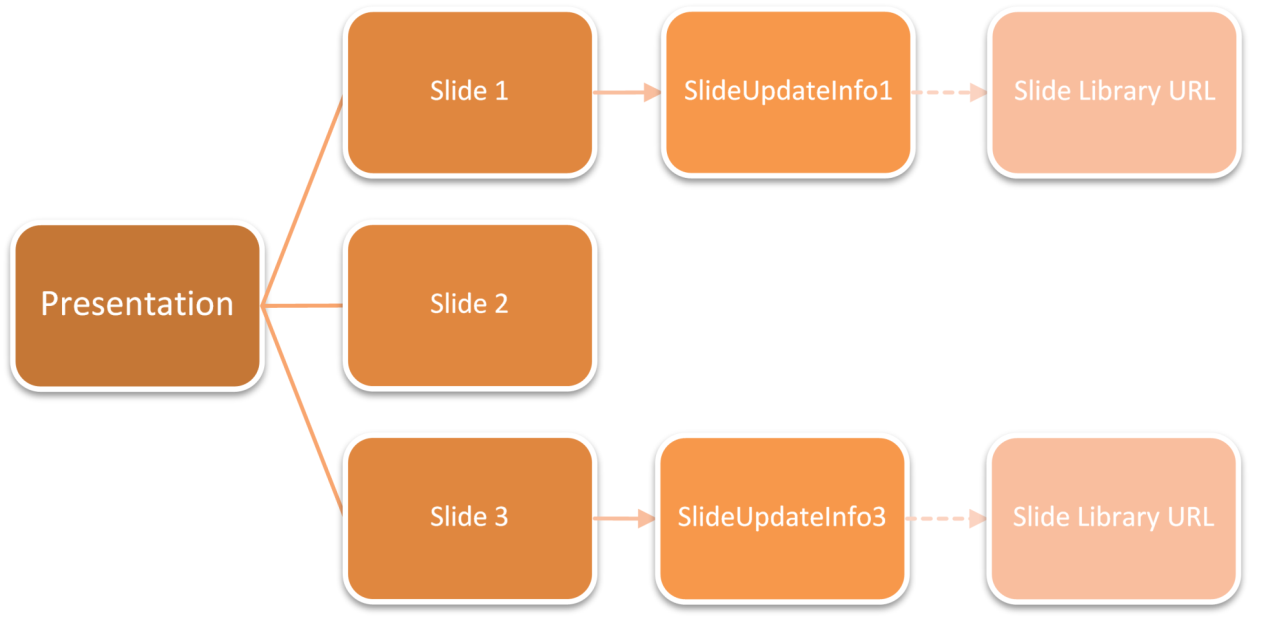
### Introduction

A Slide Library is a library type in SharePoint Server that exclusively contains single-slide presentations. Users are able to publish and reuse slides to/from these libraries. Furthermore, when a user inserts a slide from the library into a presentation, she is able to create an update relationship so that she is notified when the original slide on the server changes.

It is worth noting that the SlideUpdateInfo part in itself does not define the complete slide update functionality. That part requires a SharePoint Slide Library or compatible server (e.g., a webdav server that emulates SharePoint SOAP methods).

### Slide Update Info

For each slide in a presentation that has an update relationship with its counterpart in a Slide Library, a Slide Update Info part is created. The diagram below provides an overview of this relationship.



Each Slide Update Info part is stored under its own folder. For example:

/ppt/slideUpdateInfo/slideUpdateInfo1.xml  
/ppt/slideUpdateInfo/slideUpdateInfo3.xml

The part is indentified for each slide by a relationship with the following characteristics:

Type: [http://…/slideUpdateInfo](http://.../slideUpdateInfo)

TargetMode: Internal

Target= "<Uri of the slideupdateinfo part for the slide>"

The content type of the update info part is application/vnd.openxmlformats-officedocument.presentationml.slideUpdateInfo+xml.

It contains:

* Modified time of the slide on the server when it was inserted (stored in ISO 8061 format).
* Time the slide was inserted into the presentation.
* Regular ID of the slide on the server (saved as a string)

These Slide Update Info parts themselves have an external relationship to the Slide Library Url from which the Slide was inserted.

Type: <http://.../slidelibraryUrl>

TargetMode: External

Target = “<Url of the Slide Library>”

Every Slide Update Info part should have exactly one occurrence of this relationship.

Samples:

slideupdateinfo1.xml

<p:sldUpdatePr … serverSldId="7991" serverSldModifiedTime="2006-03-08T18:48:33"  
 clientInsertedTime="2006-03-10T06:02:33.975" />

slideupdateinfo1.xml.rels

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>  
 <Relationships xmlns="http://.../relationships">  
 <Relationship Id="rId1" Type=<http://.../slideUpdateUrl>  
 Target="http://content/slides" TargetMode="External" />  
</Relationships>

## Animation

### Introduction

This subclause provides a high-level overview of the animation settings in PresentationML. This schema is loosely based on the syntax and concepts from the Synchronized Multimedia Integration Language (SMIL), a [W3C](http://en.wikipedia.org/wiki/W3C) Recommendation for describing [multimedia](http://en.wikipedia.org/wiki/Multimedia) presentations using [XML](http://en.wikipedia.org/wiki/XML).

The schema describes all the animations effects on that reside on a slide; it also describes the animation that occurs when going from slide to slide (slide transition).

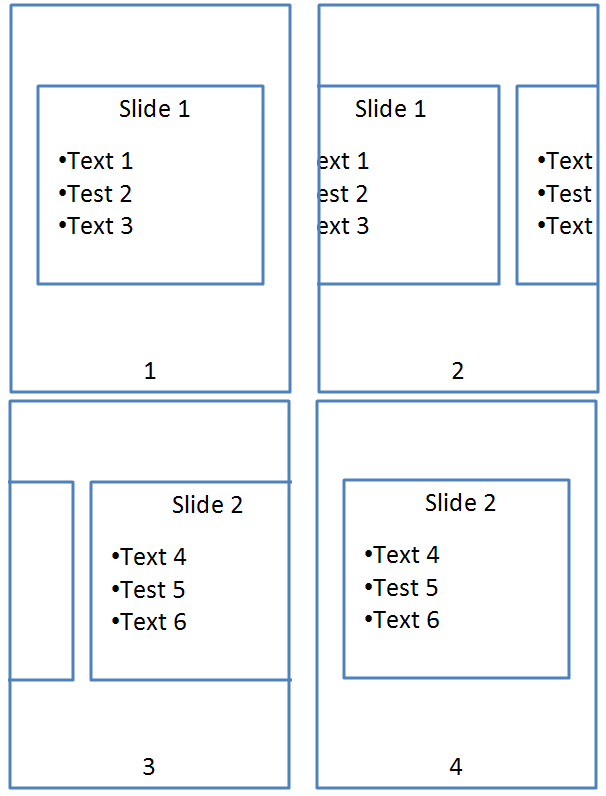
Animations on a slide are inherently time-based and consist of an animation effects on an object or text. However, slide transitions do not follow this concept and always appear before any animation on a slide.

All elements described in this schema are contained within the slide XML file. Superficially, they are in the transition and the timing element as shown below:

<p:sld>  
 <p:cSld> …  
 <p:clrMapOvr> …  
 <p:transition> …  
 <p:timing> …  
</p:sld>

### Slide Transitions

Slide transitions are the animation effects that displayed in between slides. They are specified in the transition element in the slide XML file. For example, consider a slide with a "push" slide transition as shown below:



The push element should be used as follows:

<p:transition>  
 <p:push dir="r"/>  
</p:transition>

### Timeline Overview

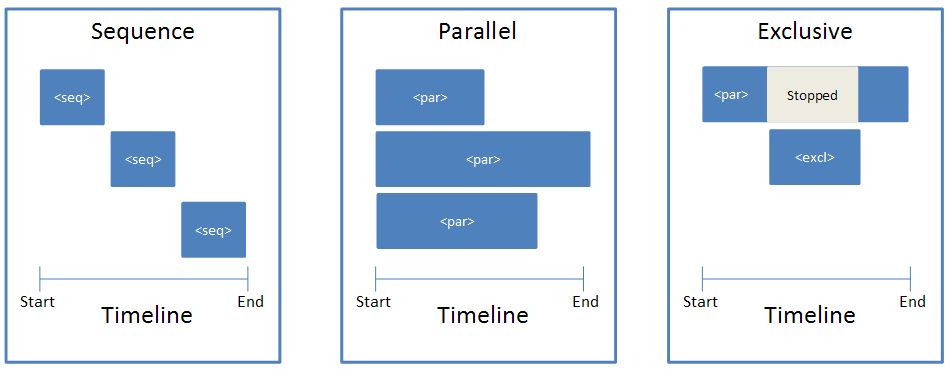
The timeline is an important aspect for animations on a slide. It moderates the amount of time that the animations are run from beginning to end. For example, it allows animation to be started when the slide is loaded or based on an event.

A timeline is comprised of timing nodes that dictate at which point a certain animation is shown. A timeline can contain unlimited number of timing nodes; it can also have time nodes nested within them.

There are three types of time nodes:

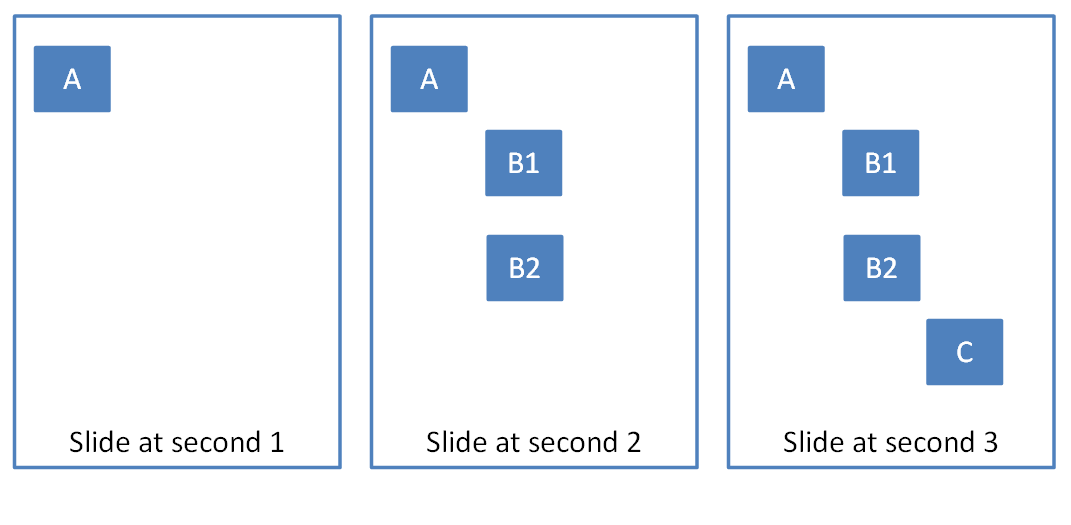
|  |  |  |
| --- | --- | --- |
| Element | Name | Description |
| par | Parallel | This is a parallel time node and can be activated along with other parallel time node containers. |
| seq | Sequence | This is a sequence time node and it can only be activated when the one before it finishes. |
| excl | Exclusive | This time node is used to pause all other timelines when it is activated. |

A conceptual diagram of this is shown below:

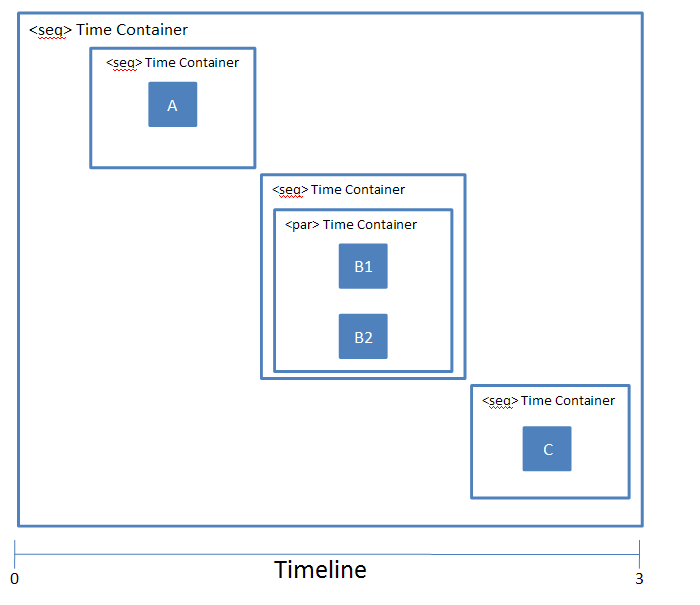


### Timeline Construction

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:



The timeline and time containers could look something like:



A typical timeline consists of the following structure:

<p:timing>  
 <p:tnLst>  
 <p:seq concurrent="1" nextAc="seek">  
 <p:stCondLst> …

<p:cTn id="2" dur="indefinite" nodeType="mainSeq">  
 <p:childTnLst>  
 <p:seq> … // Square A  
 <p:seq>  
 <par>…. // Square B1  
 <par>…. // Square B2  
 </p:seq>  
 <p:seq> … // Square C  
 </p:childTnLst>  
 </p:cTn>

<p:prevCondLst> …  
 <p:nextCondLst> …  
 </seq>  
 </p:tnLst>  
 <p:bldLst> …  
</p:timing>

As show, this timeline starts with a timing element that represents the timeline. Within this timeline, there is a child element tnList, which contains a list of time nodes.

In this case, there is one main timing container, which is the seq element. Within this element, there are a three of conditional elements, stCondList, nextCondList, and prevCondList. These elements contain condition properties that allow for the starting/stopping of the particular time node. This is explained in more detail in §4.6.6.

Following the stCondList element is the cTn element, which describes the properties for this node. Within this element is the childTnList, which contains the nested time nodes that describe the animation sequence mentioned above.

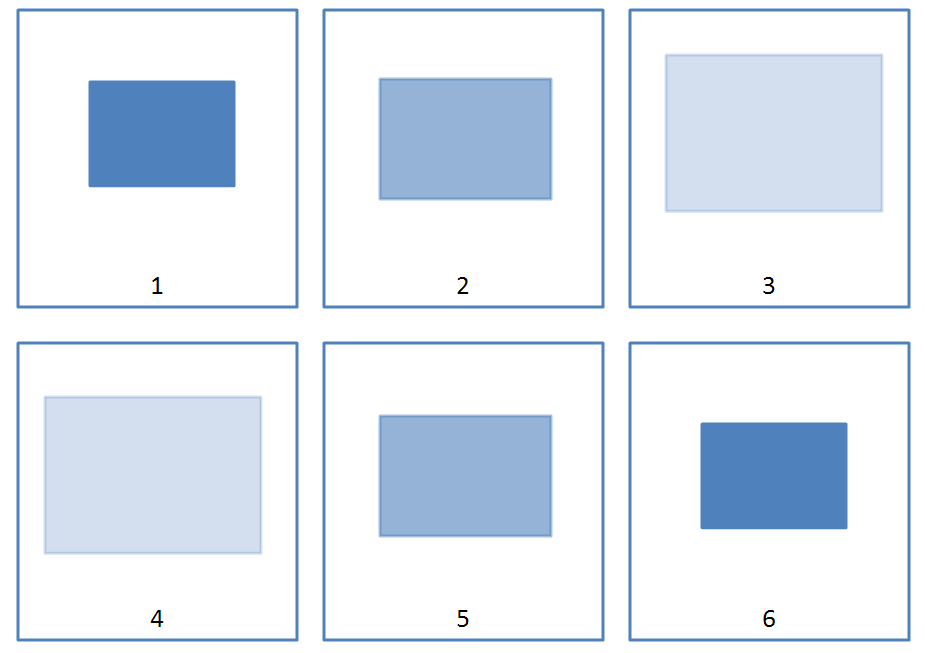
Finally, we have the bldList element, which is used to specify how objects with sub-shapes should be animated. More information can be found in §4.6.7.

### Animation Behaviors

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. |
| 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. |

A time node can combine multiple animations for a range of effects. For example, the "flash bulb" animation which scales a shape larger while at the same time having it fade uses two animation behavior elements. An example is shown below:



The representation for this animation effect in the time node element appears like:

<p:par>  
 <p:cTn id="5">  
 <p:stCondLst>….  
 <p:childTnLst>  
 <p:animEffect transition="out" filter="fade"> …  
 <p:animScale>  
 <p:cBhvr>  
 <p:cTn id="7" dur="500" autoRev="1" fill="hold"/>  
 <p:tgtEl>  
 <p:spTgt spid="9"/>  
 </p:tgtEl>  
 </p:cBhvr>

<p:by x="105000" y="105000"/>  
 </p:animScale>  
 </p:childTnLst>  
 </p:cTn>  
</p:par>

In this time node, we have two animation effects. One is creating a "fade" effect on the shape using the animEffect element and the other is creating a "scale" effect using the animScale element. All animation behaviors have a cBhvr and cTn element, which stores properties for the animation. For example, we can give the animation behaviors an ID and attributes that set the duration of the animation. The spTgt specifies the target shape to which this animation effect will be applied.

### Conditional Properties

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. |

Conditional properties are useful for providing finer granularity as to exactly when a time node should be activated. For example, suppose we have a shape with an entrance appearance after five seconds. The stCondLst element should be used as follows:

<p:par>  
 <p:cTn id="5" nodeType="clickEffect">  
 <p:stCondLst>  
 <p:cond delay="5000"/>  
 </p:stCondLst>

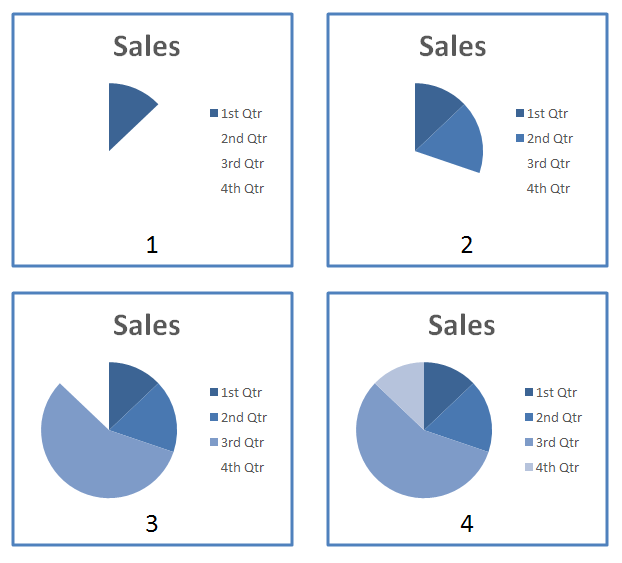
<p:childTnLst>  
 <p:animRot by="21600000"> …  
 </p:childTnLst>  
 </p:cTn>  
</p:par>

### Build Animations

Another important aspect of animations is how they are built. This refers to how the different sub-shapes or sub-components of an object are displayed. The different objects that can have build properties are text, diagrams, and charts.

This is specified in the bldLst element.

For example, suppose we want to animate a pie chart, but based on category as shown below:



The representation of this in the slide XML looks like:

<p:bldLst>  
 <p:bldGraphic spid="4" grpId="0">  
 <p:bldSub>  
 <a:bldChart bld="category"/>  
 </p:bldSub>  
 </p:bldGraphic>  
</p:bldLst>

The bldLst element contains children elements that describe how the different objects should be built. In this case, there is only one graphic to be build, that with id 4. The bldGraphic element contains how the bldSub element, which describes how the object should be build. This element then contains the bldChart element with the attribute bld set to category.

End of informative text.

# Introduction to DrawingML

This clause is informative.

This clause contains a detailed introduction to the components of DrawingML.

## Audio and Video

### Introduction

DrawingML contains support for basic audio and video capabilities. The definitions for these structures can be found in oartaudiovideo.xsd.

### Functional Overview

Presentation authors can specify that both audio and video can play while a slide is shown in slide show. When such media is inserted into a presentation, a presentation author can specify that the media is to play automatically (that is, in accordance with the slide’s animation timeline) or in response to a mouse-click. In either case, media only plays for the duration of time specified, or until the slide changes, whichever is shorter.

Sources for audio content include CD-based files as well as the more traditional disc- or server-based files.

When inserting audio content stored on a CD, the author can specify a start and end track, as well as an index into each track. This information identifies the content from the CD to be played for the specified slide during a slide show.

In cases where the audio or video content is stored on a hard drive or server, the author can only specify the file itself; it will be played in its entirety, or until the slide changes.

### DrawingML Syntax

In all three cases, the media objects themselves are stored on the slides using a picture shape. The picture shape uses a blipFill to show the media object on the slide’s surface. In both audio cases, the picture used is the icon image, whereas in the video case, the picture used is the poster frame for the video file.

To express this information, the standard blipFill element is used to refer to the image file; and because this refers to a file within the package, a relationship ID is used:

<p:pic>  
 <p:nvPicPr> …</p:nvPicPr>  
 <p:blipFill>  
 <a:blip r:embed="rId4" r:link="" />   
 <a:stretch>  
 <a:fillRect />   
 </a:stretch>  
 </p:blipFill>  
 <p:spPr> … </p:spPr>  
</p:pic>

As the media objects are related to the slide’s timeline—in both the automatic and mouse-click cases—they must have interactivity information stored in the form of a hyperlink.

To express this information, a hyperlink is added to the non-visual shape properties; and because it is a hyperlink, it, too, uses a relationship ID:

<p:pic>  
 <p:nvPicPr>  
 <p:cNvPr id="15" name="Rectangle 15" descr="">  
 <a:hlinkClick r:id="rId3" tgtFrame="" tooltip="" />   
 </p:cNvPr>  
 <p:cNvPicPr />  
 <p:nvPr> … </p:nvPr>  
 </p:nvPicPr>  
 <p:blipFill> … </p:blipFill>  
 <p:spPr> … </p:spPr>  
</p:pic>

The final piece of information required for each media type is the source bits. In both file-based media cases, DrawingML needs to provide a mechanism to specify the location and file name for the media; this is in contrast to the CD-based audio where only track information is required. Regardless of the type of source information required, all of this is stored in application-specific non-visual properties. This is illustrated in the following three XML islands representing each of the three media cases:

|  |  |
| --- | --- |
| CD-Audio | <p:pic>  <p:nvPicPr> … <p:cNvPicPr />   <p:nvPr>  <a:audioCd>  <a:st track="2" time="50" />   <a:end track="3" time="22" />   </a:audioCd>  </p:nvPr>  … </p:pic> |
| File-Audio | <p:pic>  <p:nvPicPr> … <p:cNvPicPr />   <p:nvPr>  <a:audioFile r:embed="" r:link="rId1" />   </p:nvPr>  </p:nvPicPr>  … </p:pic> |
| File-Video | <p:pic>  <p:nvPicPr> … <p:cNvPicPr />   <p:nvPr>  <a:videoFile r:embed="" r:link="rId1" />   </p:nvPr>  </p:nvPicPr>  …  </p:pic> |

In the CD-Audio case, there is no capability to choose the particular CD drive that will contain the source. This is a functional limitation.

While the default case is that media is linked, file-based media can also have the source bits included in the package as a separate part. In this case, the relationships point not to an external file but, rather, to a part inside the package.

## Text

### Introduction

This subclause provides a high-level overview of the content described in the following schemas: dml-text.xsd, dml-textpara.xsd, dml-textrun.xsd, dml-textchar.xsd and dml-textbullet.xsd.

The best way to understand these schemas as they relate to one another is to learn about the DrawingML Text file format in the following order.

* Text Overview
* Body Level Properties
* Paragraph Level Properties
* Run and Character Level Properties

Companion schemas build on the ones discussed in this document. As these are encountered below, pointers to them are provided.

This subclause provides a structured breakdown of the text portion of the DrawingML file format. Other subclauses build on this foundation and explain more about topics such as text frame, text styles, text fields, and embedded fonts.

Note that the use of the "p" namespace within this document references to the PresentationML-specific schemas while the use of the "a" namespace within this document references to the DrawingML-specific schemas.

### Overview

Consider an XML tree that has the following basic structure:

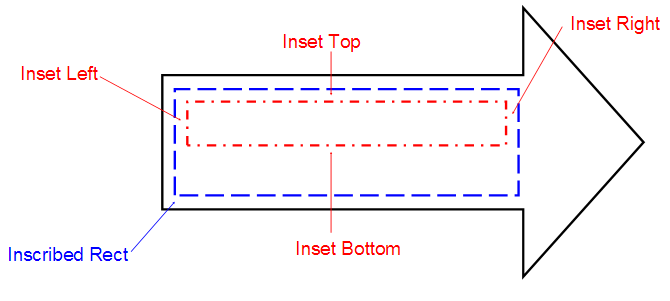
<p:sld>  
 <p:cSld>  
 <p:spTree>  
 <p:sp>  
 <p:txBody>  
 Your text here!  
 (not really, there are other elements needed within here)  
 </p:txBody>  
 </p:sp>  
 <p:sp>  
 </p:sp>  
 …  
 </p:spTree>  
 </p:cSld>  
</p:sld>

In the structure above, we are interested in the content contained within the matching p:txBody tags. The understanding of this tag in relation to the basic slide structure above encompasses the schema background needed to digest effectively the remainder of this description.

Note that shapes are the standard container within which all text resides. Usually, the shape does not have any visual properties attached to it and thus no visible shape is rendered; nonetheless, a shape is still present and does house any content text.

Each shape contains an inset rectangle that houses any text attached to that shape. The shape has margins or insets that buffer this rectangle on all four sides (top, bottom, left, and right) just like margins on a page. When thinking about text within a shape, it is useful to keep these inset properties in mind.

An illustration of this is provided below.



Let's look at the different element tags contained within p:txBody. Listed below are only those tags discussed here. (Note that this sample framework is a skeleton and does not fully show all elements and attributes needed.)

<p:txBody>  
 <a:bodyPr /> required, only listed once.  
 <a:lstStyle /> optional, if present only listed once.  
 <a:p> required, no limit on the number of instances.  
 <a:pPr /> optional, if present only listed once.  
 <a:r> required, no limit on the number of instances.  
 <a:rPr /> optional, if present only listed once.  
 <a:t>Your text here!</a:t>  
 Actual text for this run is contained here.  
 </a:r>  
 <a:endParaRPr /> optional, if present only listed once.  
 </a:p>  
</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 §5.2.3. |
| 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 §5.2.3.4. |
| 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 §5.2.3.4. |
| 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 §5.2.3.8. |
| 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 §5.2.3.8. |
| 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. |

### Body Level Properties

Schemas represented here: dml-text.xsd

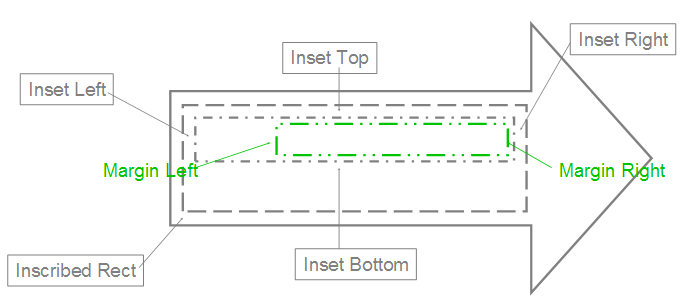
In this subclause, we'll explore the sorts of properties that can be attached to the body as a whole. As shown in the sample XML above, there are three essential property levels available. The body-level properties are to the broadest of these. Note that some of these body-level properties are applied as attributes to the body 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.

<p:txBody>  
 <a:bodyPr /> 🡨 Main element covered in this subclause  
 <a:lstStyle />  
 <a:p>  
 <a:pPr />  
 <a:r>  
 <a:rPr />  
 <a:t>Your text here!</a:t>  
 </a:r>  
 </a:p>  
</p:txBody>

#### Setting Up the Text Area

Let's start with how a text area might be initially described. This area is the container within which all the child text for this body resides. First, it is useful to understand the inset properties; specifically, the top, bottom, left and right inset properties that are also known as internal margins for the text body. These attributes are optional and have a default value of 0. The anchor attribute allows us to specify where the text area should be anchored within its bounding rectangle.

An illustration of this bounding rectangle is highlighted below by the inner green box. Notice here that the bounding rectangle is anchored to the right.



Here's how the text will appear inside. Attribute AutoFit allows for three basic scenarios:

* No AutoFit: The text is allowed to flow outside the container.
* Normal AutoFit: The text is resized using defined constraints to fit inside the container area. (This is used when the text is too large or long to fit in the text container.)
* Shape AutoFit: The actual text container is resized to contain all the text. (This is the only option that can cause the container to have its dimensions changed.)

The term flow is used to describe the way in which text moves around inside this text area, and to describe how each of the body properties affects the text within the text area.

One way that text can flow is from one line to the next. This can be done automatically by using the text-wrapping attribute. Another way is to use columns. The XML framework allows for the specification of a number of columns into which the text is to be automatically broken. This feature also allows for the specifying of the spacing of columns and a right-to-left layout instead of the default left-to-right. Another way that text can flow is vertical instead of horizontal. For this, there are many different types of vertical text that can be described: from text that appears rotated to text where the characters are truly stacked. The text can even be made to flow different when an East Asian font is specified.

When looking at the flow it is useful to discuss the potential for overflow. That is, the text must flow outside the text area because it is too large to fit inside. For this, there are two common types: vertical and horizontal. The vertical overflow can be handled in three ways:

* overflow: This allows the text to flow outside the text area.
* 
* ellipsis: This crops the text that overflows and adds "…" to denote that there is hidden text.
* 
* clip: This crops the text just as ellipsis but does not insert "…", so the user has no indication that there is hidden text.
* Horizontal overflow works exactly like vertical, but with only two options: overflow and clip, which both operate as described above.

#### Manipulating the Text

Let's look at the ways in which the text can be further enhanced at the text body level. Note that the properties that follow apply only to the text body as a whole and thus cannot be applied to a specific paragraph or run within the text body. Theses properties are as follows:

* Text Warping: Text within the text area is made to distort itself according to a predefined shape. This shape resides within the bounding box described earlier. This effect is known as text warping and has its preset shapes defined further within dml-shapepgeometry.xsd
* 
* 3D Text: Text can be described with respect to a 3D scene. Using this tag provides three basic options:
* The text resides within a 3D scene, but as planar text.
* The text is allowed to reside within the 3D scene and has 3D effects (such as bevel or extrusion) applied to it.
* The text resides on top of a 3D scene. The 3D scene properties are defined oartdml-shape3dstyles.xsd and oartdml-shape3dscene.xsd schemas.
* 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.

#### Backwards and Forwards Compatibility

The following areas are of interest when considering support for both past design and future innovation.

* 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.
* 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.

#### Paragraph-Level Properties

Schemas represented here: dml-textpara.xsd, dml-textbullet.xsd.

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.

<p:txBody>

<a:bodyPr />  
 <a:lstStyle />  
 <a:p>  
 <a:pPr /> 🡨 Main element covered in this subclause  
 <a:r>  
 <a:rPr />  
 <a:t>Your text here!</a:t>  
 </a:r>  
 </a:p>  
</p:txBody>

#### Spacing, Alignment, and Direction

The XML file format allows for the specifying of spacing both between lines in the form of line spacing, and also outside the paragraph via margins and special before/after spacing. In addition to this, there is also the ability to specify indent spacing for the beginning of the paragraph.

The standard alignment options include left-aligned, right-aligned, centered, justified, and distributed. Justified alignment causes each line of text to be stretched out to a certain point. To ensure that short lines remain readable, they are not stretched. Distributed alignment is quite similar, but stretches every line, regardless of that line's length.

Text direction is specified as either left-to-right (the default) or right-to-left using the specific rtl tag.

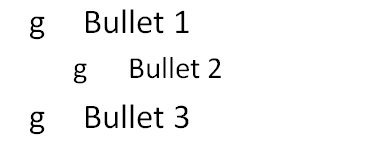
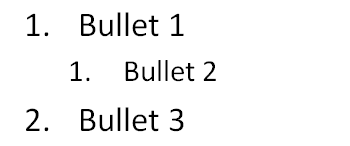
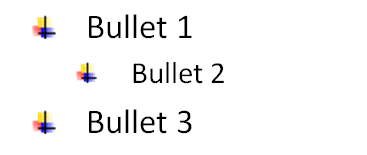
#### Tabs and Line Breaks

When the default tabs are not sufficient for the paragraph in question there is the option of including custom tab stops in the XML file format. The information required for this is both a default tab size attribute and a full tab stop list showing all tab stop positions that apply to this paragraph. Keep in mind that if tab stops are not explicitly stated in the file format that the business logic of the application must use its own default positions if tabs are needed.

Line break is a tag that informs the application as to whether it should break up a string of text onto multiple lines based on Latin grammar rules or East Asian grammar rules. The East Asian option uses the Kinsoku settings to determine whether a word is allowed to begin or end a line of text.

#### Adding Bullets

Bullets are specified per paragraph, so bullets can be mixed and matched within a single text body to appear as a coherent text group. The types of bullets available are:

* Character Bullets: Uses a font character to denote a bullet and can be set to appear in any size (percentage of text), color (all available including theme colors), and font. The properties are Bullet Color, Bullet Size, Bullet Font Typeface, Bullet Character (represents the actual bullet)
* 
* 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.
* 
* 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.
* 

#### Run- and Character-Level Properties

Schemas represented here: dml-textrun.xsd, dml-textchar.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.

<p:txBody>  
 <a:bodyPr />  
 <a:lstStyle />  
 <a:p>  
 <a:pPr />  
 <a:r>  
 <a:rPr /> 🡨 Main element covered in this subclause  
 <a:t>Your text here!</a:t>  
 </a:r>  
 </a:p>  
</p:txBody>

#### Visual Properties

When looking to format a run of text the first property that one might need to specify would be the font typeface. The XML file format allows for the specification of not only Latin Fonts but also East Asian, Complex Script, and Symbol fonts as well. These four font buckets give the application additional information that is used to layout text in a manner fitting for the specific font. Along with the actual font being used, comes the 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 points.

Other common formatting properties allowed in the XML framework are bold, italic, underline and strikethrough. The use of both the bold and italic properties is simply via a Boolean value of 0 or 1. The usage of the underline and strikethrough, however, allow a more specific selection to be made. There are 17 values for underline, which range from a single line to wavy double lines. In addition to specifying the style of underline that is to be used, the framework can also specify fill properties for the underline. These are solid color, multi-color gradient, and texture fill. For strikethrough, there are two options: single and double strike through.

When standard formatting isn't adequate, more complex effects can be defined for a specific run of text. The basic breakdown for these is line properties, fill properties and effect properties. Encapsulated within each of these areas is a wide range of customizable effects. A quick look at line properties, for example, reveals the ability to specify a color, gradient, or pattern fill, along with a width and style applied. Along these lines fill properties allows for transparent fill, solid fill, gradient fill, texture fill and even picture fill. While these features alone give the XML file format plenty of robustness in describing text, other features are also available. Because text is treated the same as a shape, a run of text can have virtually all shape effects applied to it just as if it were a shape. These effects include shadow, glow, and reflection, and are placed in an effect list under the run properties tag. An example of what these lines, fills and effects may look like is provided below. More information on these effects can be found in either dml-shapelineproperties.xsd or dml-shapeeffects.xsd.

<a:rPr>  
 <a:ln>  
 <a:solidFill … /> 🡨 Line properties here  
 </a:ln>  
 <a:gradFill>  
 <a:gsLst … /> 🡨 Fill properties here  
 </a:gradFill>  
 <a:effectLst>   
 <a:reflection … /> 🡨 Effect properties here  
 </a:effectLst>  
</a:rPr>

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.

#### Properties for Interactivity

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.

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.

#### International Language Support

There exists the notion of the language id, which is simply a value that assists the application in laying out the text. The tags that help with this are the language id tag and the alternate language id tag. Together, these allow the file format to be robust and handle multiple languages for a single run of text. In addition, there is also the kumimoji tag, which aids with the layout of East Asian text by specifying whether numbers appear vertically with text (default) or horizontally. An illustration of a run of text with kumimoji applied is provided below.



### Coordinate Systems and Transformations

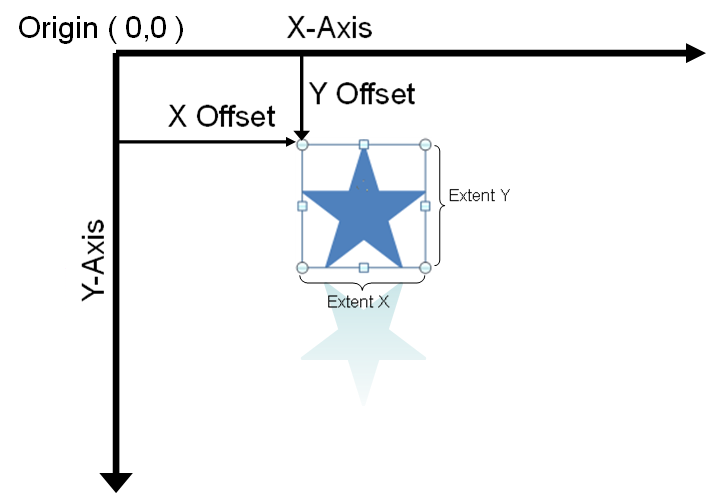
### Introduction

This document provides an overview of the transformation elements for shapes and groups, represented by a:xfrm in Dml-basetypes.xsd. These schemas are for the representation of scaling and rotation on individual shapes and groups.

### Base Representations

All DrawingML shapes are located on a 2-D Cartesian coordinate space with the origin (0,0) in the upper left-hand corner of the canvas. The x-axis coordinates grow positively from left to right, and the y-axis coordinates grow positively from top to bottom. The units of each axis are measured in EMUs (there are 914,400 EMUs per inch).

The location and dimensions of a shape or group (for simplicity, both classes of items are referred to here as "shapes", but we also mean to include "groups"), are represented within the canvas by specifying a point, known as a shape's offset, along with a bounding box representing the containing rectangle of that shape or group. The location is represented as a point in Cartesian coordinate space and referred to as an offset (a:off). The bounding box is represented as a 2-tuple of height and width values known as the extents (a:ext).



The following XML fragment represents the offset and extents for the star shape above:

<a:xfrm>  
 <a:off x="2209800" y="2514600"/>  
 <a:ext cx="1371600" cy="1371600"/>  
</a:xfrm>

Notice that as demonstrated with the example above, effect bounds have no impact on shape or group extent dimensions.

### Scalings and Rotations on Base Representations

For base representations (i.e., items not placed within a group), the offset and extents are all that are required. Scalings will impact the extents of an object.

The following XML Fragments represent the offset and extents for a star shape, before and after scaling.

Note: The two shapes above overlap as they have been placed at exactly the same offset to show that scaling has no impact on the offset values, but only the extents.

Before scaling (small star):

<a:xfrm>  
 <a:off x="1066800" y="990600"/>  
 <a:ext cx="1371600" cy="1371600"/>  
</a:xfrm>

After scaling (large star):

<a:xfrm>  
 <a:off x="1066800" y="990600"/>

<a:ext cx="2438400" cy="2133600"/>

</a:xfrm>

Rotations are represented with the rot attribute.

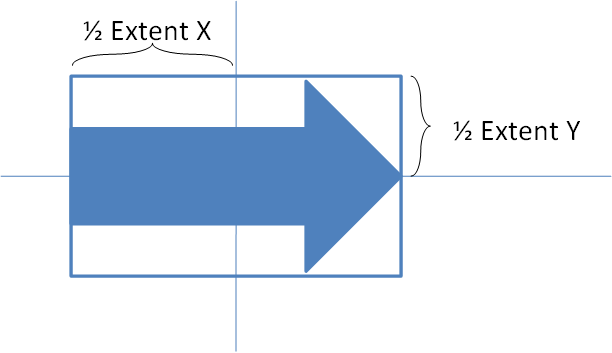
If we take the small star from above, and apply a 45-degree rotation to it, we get:

<a:xfrm rot="2700000">  
 <a:off x="1066800" y="990600"/>  
 <a:ext cx="1371600" cy="1371600"/>  
</a:xfrm>

Observed that rotations have no impact on the offset or extent on a base entity. The offset and extent represent the location and dimensions of that entity prior to any rotation.

### Flips

In addition to scalings and rotations, the transformation schemas also allow for the representation of flips. A horizontal flip flips an object about its x-axis centered upon the midpoint of its extents. Likewise, a vertical flip flips an object about its y-axis centered upon the midpoint of its extents.



Horizontal and vertical Flips are represented with the flipH and flipV attributes respectively.

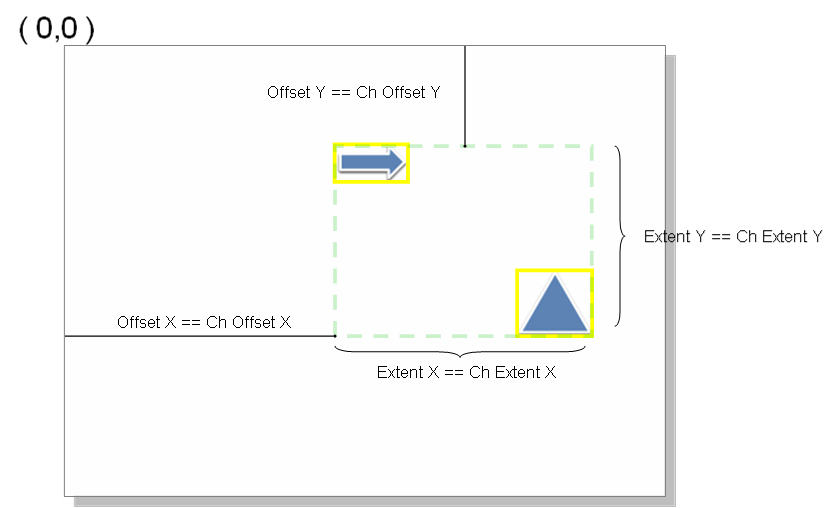
The following XML fragment illustrates a shape which has been flipped both horizontally and vertically

<a:xfrm flipH="1" flipV="1">  
 <a:off x="3964937" y="2652643"/>  
 <a:ext cx="168838" cy="1219199"/>  
</a:xfrm>

### Group Representations

Groups are represented very similarly with offsets and extents. Groups differentiate themselves from shapes in that there are another set of offsets and extents known as child offsets and child extents. When a group is created, the bounding box of a group's offset and extent is recorded into the group's child offset and child extent variables. In other words, offset and extent of the group's bounding box is saved. By saving this bounding box information, the relative geometries of each shape in the group are preserved. It is the preservation of these child offset and extent variables which enable us to perform transforms on groups while preserving the relative dimensions of each member of the group.

A group is created from two shapes: an arrow, and a triangle. The bounding box of the group is first computed. It's offset and extents are recorded into the group's child extent variables.



The following is an XML snippet representing the transform variables of the newly created group.

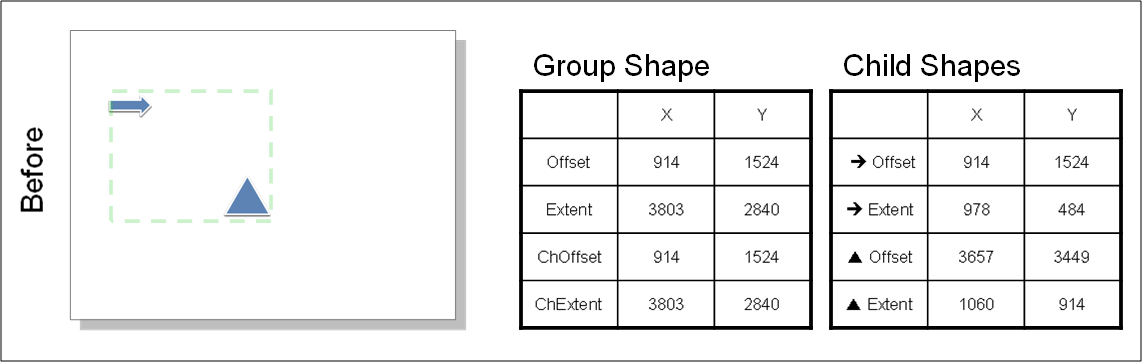
<p:grpSpPr>  
 <a:xfrm>  
 <a:off x="2209800" y="2514600"/>  
 <a:ext cx="4038600" cy="2286000"/>  
 <a:chOff x="2209800" y="2514600"/>  
 <a:chExt cx="4038600" cy="2286000"/>  
 </a:xfrm>  
</p:grpSpPr>

Observe that the offset and extents for the group are identical to the child offset and child extents of the group at creation time.

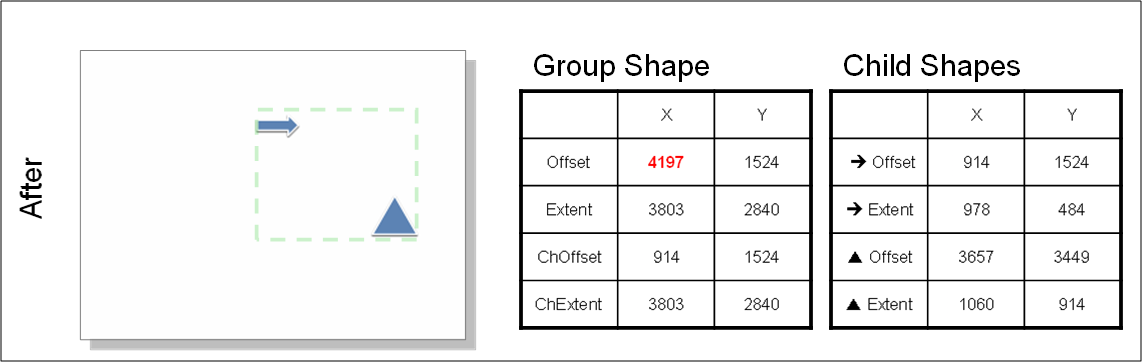
### Translation (Moving of Groups)

When groups are moved, the only affected variables are the offset and the extent for the group. Note that the following are unaffected: the child offset, the child extents, the offset and extent parameters of the grouped shapes.

The following example illustrates the moving of a group to the right. Let's assume we have the following group, with the following parameters:



Notice that the offset == ChOffset and extent == ChExtent, as this is a group that has just been created with no operations performed on it. We now perform a move on this group to the right.

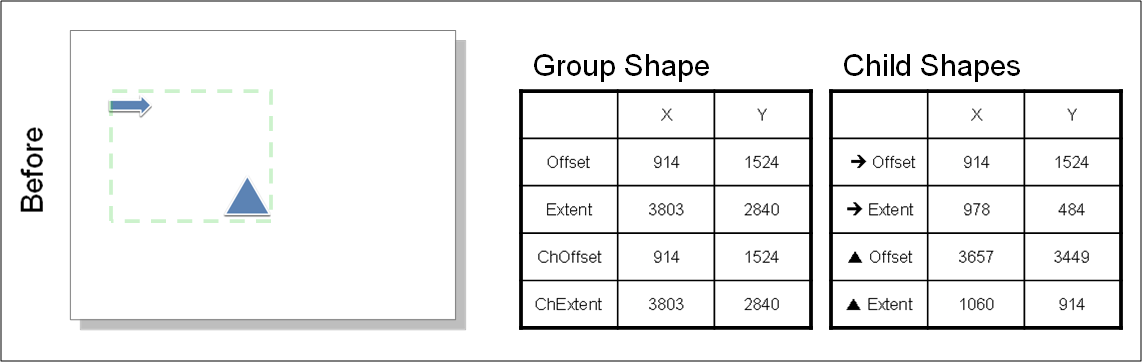


The only variable that gets updated as a result of this translation is the x component of the offset. Had this move also been vertical, the y component of the offset would have been affected as well.

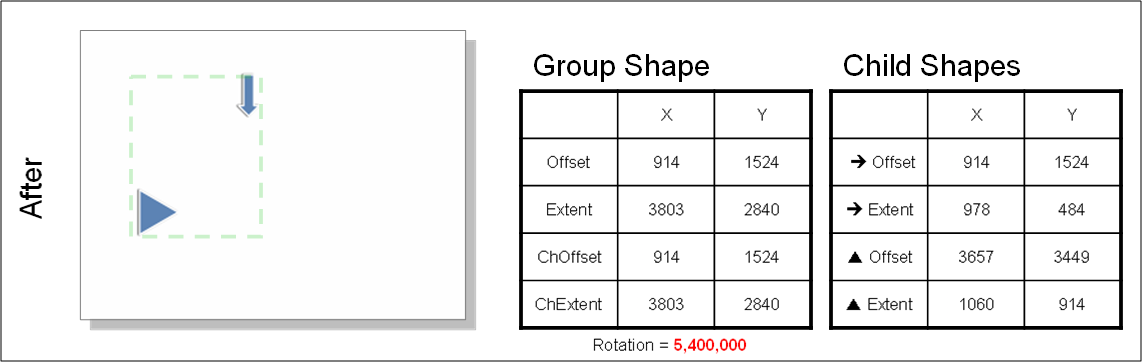
### Group Rotation

As with shapes, there is a rotation attribute on groups. Just like with shapes, rotation of a group has no effect on offset or extent variables. Rotation for groups (as with shapes) is done relative to the center off the bounding box for the group.

A group is rotated 90-degrees clockwise.



After rotation, the variables are as follows:

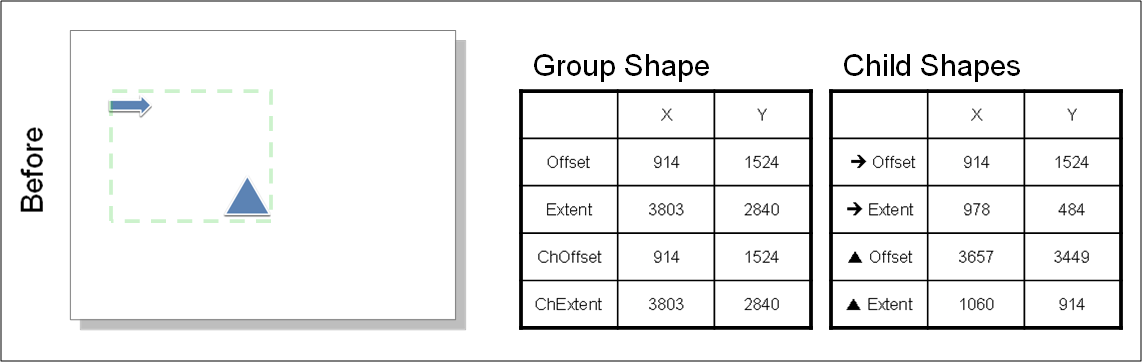


Notice that the rotation attribute is marked with the rotation value and that none of the other variables are affected.

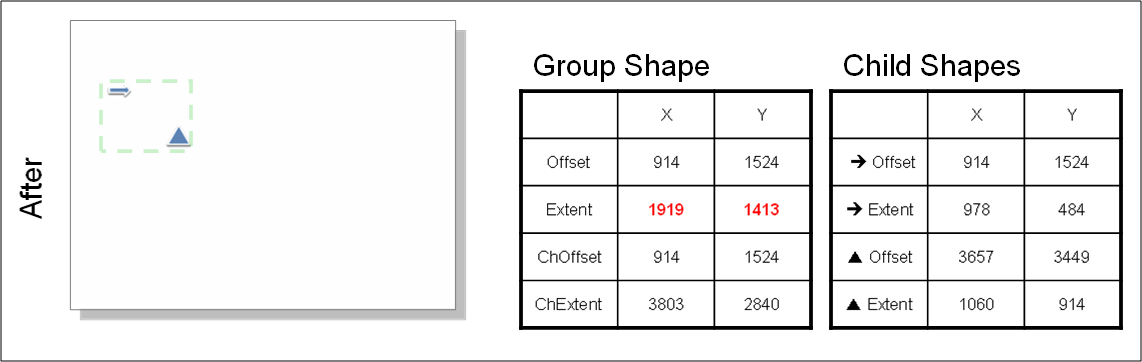
### Group Scaling

Scaling with respect to a group is performed by recording the new extents.

The following group is scaled to 50% of its original size (in both x and y directions)



Notice that in this example, the scaling is done by resizing the bounding box of the bottom right corner and dragging it towards the top left exactly until it is half its original size. So the offset does not change. As the extents are now smaller by half, this is reflected accordingly by the halving of the extents variables.



### Translation, Scaling, and Rotation of Group Members

So far we have considered the cases of translation, scaling, and rotation when applied to groups as a whole. The next level the complexity is when these operations are performed on individual shape elements when they are part of a group.

When individual members of a group are manipulated, the following algorithm gets applied:

1. A new bounding box is computed as a result of the modification (translation, scaling, or rotation). It is possible the new bounding box is identical to the old group's bounding box. If so, we are done, and only the child shape's variables need to be modified. Else, the group shape's variables must be recomputed.

2. The new child offset and child extents are updated to have the values of the new bounding box.

3. The new offset and extents are computed.

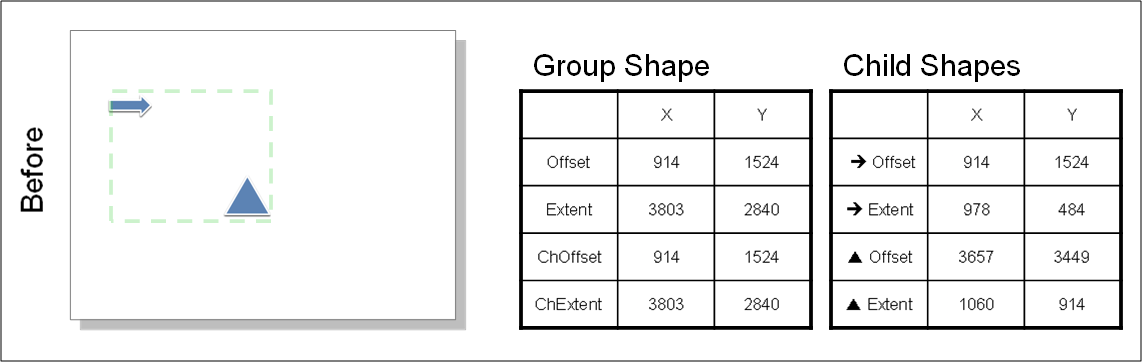
4. As we desire to maintain the same scaling factors as the previous transform, the new extents may be computed given the child extents and the old scaling factors.

5. The flipping and rotation properties are maintained as before.

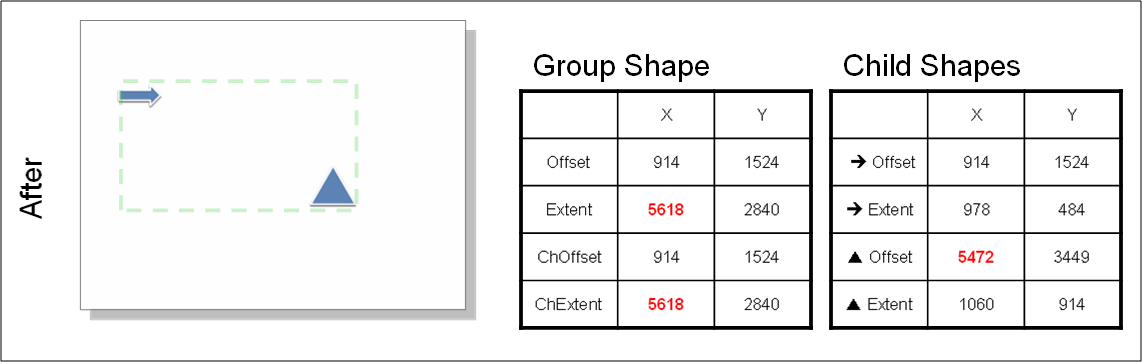
6. This leaves the group offset as the only parameter which must be adjusted for to maintain the relative position of untouched elements the same.

#### Translation

Consider the case of moving a shape within a group:



Let's say that the triangle is moved to the right.

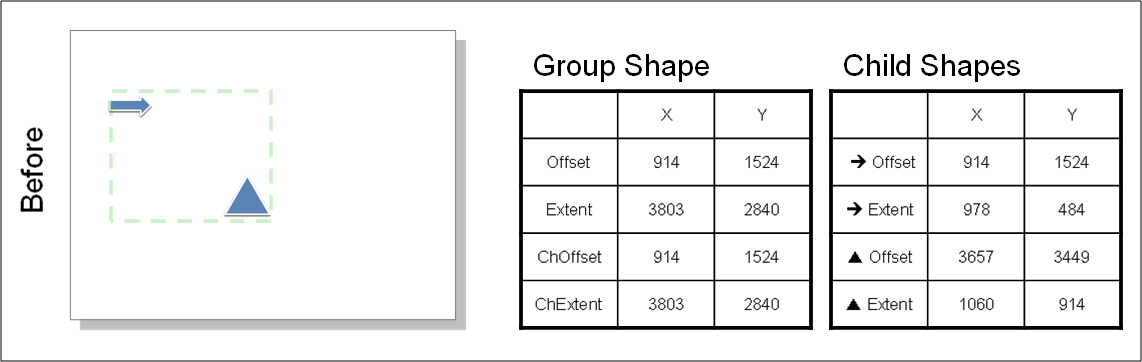


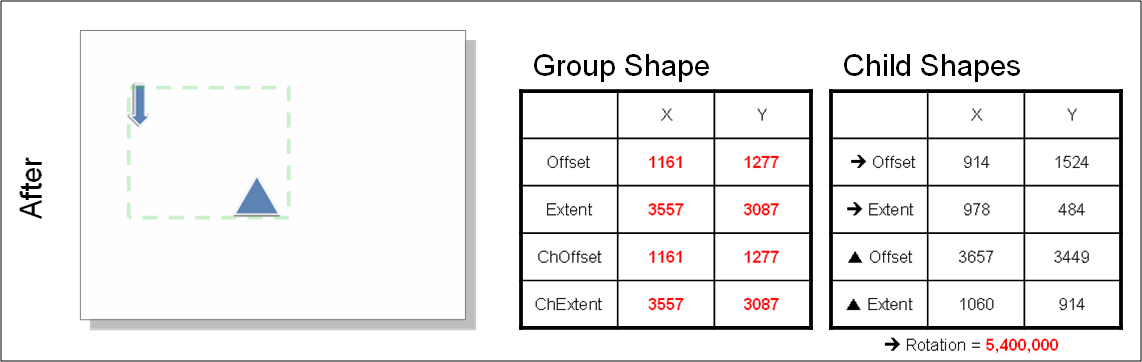
In the fully general case, we would expect all variables to change. However, as the arrow remains at exactly the same location and we have applied no transformations to it, its offset and extents remain the same. As we have moved the triangle to the right, its offset X variable is changes while all other variables remain constant. The result of regrouping the same elements of the old group results in a new offset and extent value for the group offset and extent as well as its child offset and its extent.

#### Rotation of a Child Member

Rotation of a child member is no different.

Consider the impact of rotating the arrow clockwise, 90 degrees.





### Transformation Computations

Assume a point (x,y) on a shape. The following computations are applied to it to derive the location in the global canvas which results due to the translations, rotations and flips applied to it.

### Apply Shape Transforms

First, the local rotations and flips are applied to a local shape. Note that all rotations are performed with respect to the center of the bounding box.

Let's represent our point p as a vector in homogeneous form as: 

We now transform the point to the center of the bounding box in preparation for rotation. Let's define translation matrix, , as this translation matrix: 

Note that the inverse transform to return the point back to its original location is

The rotation matrix R(θ), is 

A flip represented as , where flipH and flipV are -1 when a horizontal or vertical flip respectively is desired, else, they are 1.

Putting the pieces together, the first set of transforms to be applied gives us the result of rotations and flips applied to an ungrouped shape:



### Apply Group Transforms

Local coordinate space refers to the coordinate space of a shape within a group. Global coordinate space refers to the coordinate space of the overall canvas (e.g., a slide).

When a shape is part of a group, the conversion from local to global coordinate space is similar to the equations above, with the exception of two additional transformation steps:

First, the local coordinates must be translated to the position:



The scaling matrix S is defined as:  and scales the

the computation from local to global space is as follows:



## Styles

### Introduction

This piece of DrawingML deals with the definition of the shared aspects contained within a document theme. The shared-style sheet defines an application-independent set of styling that can be applied to objects within a document and which affects the look of the document and the information and objects it contains. For example, in a presentation, shapes can have a certain look, whereas in an e-mail, all of the text can have certain properties, and headings are styled.

A second topic is the definition of a table style as used within DrawingML. A table style defines the look of a table regardless of the data present in that table.

### Shared Style Sheet

The shared-style sheet within DrawingML is responsible for containing different formatting options and style options that can be used within a given document.

#### Theme

The theme is the root-level complex type associated with a shared-style sheet. This complex type holds all of the different formatting options available to a theme, and defines the overall look and feel of a document when themed objects are used within the document. In figure 1 below, we can see an example of two different themes applied to the same slide in a presentation.

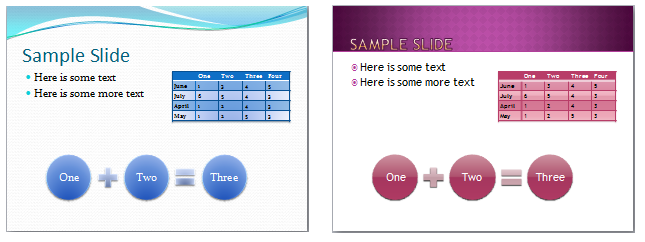


Figure 1: A theme applied to the same slide in a presentation. Not only does the font and colors change, but also the effects applied to the shapes and table.

A theme consists of four main parts, although the themeElements element is the piece that holds the main formatting defined within the theme. The other parts provide overrides, defaults, and additions to the information contained in themeElements. The complex type defining a theme, CT\_OfficeStyleSheet, is defined in the following manner:

<complexType name="CT\_OfficeStyleSheet">  
 <sequence>  
 <element name="themeElements" type="CT\_BaseStyles" minOccurs="1"  
 maxOccurs="1"/>

<element name="objectDefaults" type="CT\_ObjectStyleDefaults"  
 minOccurs="0" maxOccurs="1"/>

<element name="extraClrSchemeLst" type="CT\_ColorSchemeList"  
 minOccurs="0" maxOccurs="1"/>

<element name="custClrLst" type="CT\_CustomColorList" minOccurs="0"  
 maxOccurs="1"/>

<element name="extLst" type="CT\_OfficeArtExtensionList"  
 minOccurs="0" maxOccurs="1"/>  
 </sequence>  
 <attribute name="name" type="xsd:string" use="optional" default=""/>  
</complexType>

This complex type also holds a CT\_OfficeArtExtensionList, which is used for future extensibility of this complex type.

#### Theme Elements

The complex type CT\_BaseStyles defines the theme elements for a theme, and is the workhorse of the theme. The bulk of the shared theme information that is used by a given document is defined here. Within this complex type is defined a color scheme, a font scheme, and a style matrix (format scheme) that defines different formatting options for different pieces of a document. The complex type CT\_BaseStyles is defined in the following manner:

<complexType name="CT\_BaseStyles">  
 <sequence>  
 <element name="clrScheme" type="CT\_ColorScheme" minOccurs="1"  
 maxOccurs="1"/>

<element name="fontScheme" type="CT\_FontScheme" minOccurs="1"  
 maxOccurs="1"/>

<element name="fmtScheme" type="CT\_StyleMatrix" minOccurs="1"  
 maxOccurs="1"/>

<element name="extLst" type="CT\_OfficeArtExtensionList"  
 minOccurs="0" maxOccurs="1"/>  
 </sequence>  
</complexType>

#### Color Scheme

The complex type CT\_ColorScheme defines a set of colors for the theme. The set of colors consists of twelve color slots that can each hold a color of choice. The colors are organized in the following way:

* Dark 1 (dk1) – This represents a dark color, usually defined as a system text color
* Light 1 (lt1) – This represents a light color, usually defined as the system window color
* Dark 2 (dk2) – This represents a second dark color for use
* Light 2 (lt2) – This represents a second light color for use
* Accents 1 through 6 (accent1 through accent6) – These are six colors which can be used as accent colors in the theme
* Hyperlink (hlink) – The color of hyperlinks
* Followed Hyperlink (folHlink) – The color of a followed hyperlink

These colors define the theme colors that objects can utilize within a document. When an object uses a theme 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.

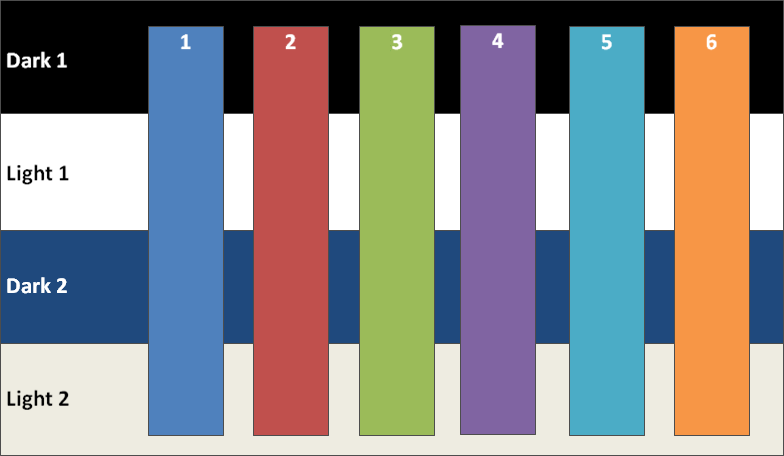


Figure 2: Sample colors defined and used for dark1/2, light1/2, and the six accent colors.

The complex type CT\_ColorScheme is defined in the following manner:

<complexType name="CT\_ColorScheme">  
 <sequence>  
 <element name="dk1" type="CT\_Color" minOccurs="1" maxOccurs="1"/>

<element name="lt1" type="CT\_Color" minOccurs="1" maxOccurs="1"/>

<element name="dk2" type="CT\_Color" minOccurs="1" maxOccurs="1"/>

<element name="lt2" type="CT\_Color" minOccurs="1" maxOccurs="1"/>

<element name="accent1" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="accent2" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="accent3" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="accent4" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="accent5" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="accent6" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="hlink" type="CT\_Color" minOccurs="1" maxOccurs="1"/>

<element name="folHlink" type="CT\_Color" minOccurs="1"  
 maxOccurs="1"/>

<element name="extLst" type="CT\_OfficeArtExtensionList"  
 minOccurs="0" maxOccurs="1"/>  
 </sequence>  
 <attribute name="name" type="xsd:string" use="required"/>  
</complexType>

#### Font Scheme

The complex type CT\_FontScheme defines a font pair. The pair consists of a major font and a minor font. An example of use would be the major font used in headings for a document and the minor font used for the paragraph parts of a document. The major and minor fonts are defined through a collection of font faces defined on a per-language basis. For example, one may define only a Latin-based font, or one can define many different fonts for different locals for a major or minor font. The font used in the document depends on the user’s language.

The complex type CT\_FontScheme is defined in the following manner:

<complexType name="CT\_FontScheme">  
 <sequence>  
 <element name="majorFont" type="CT\_FontCollection" minOccurs="1"  
 maxOccurs="1"/>

<element name="minorFont" type="CT\_FontCollection" minOccurs="1"  
 maxOccurs="1"/>

<element name="extLst" type="CT\_OfficeArtExtensionList"  
 minOccurs="0" maxOccurs="1"/>  
 </sequence>  
 <attribute name="name" type="xsd:string" use="required"/>  
</complexType>

#### Major and Minor Font (Font Collection)

The complex type CT\_FontCollection defines a major and minor font which is used in the font scheme. A font collection consists of a font definition for Latin, East Asian, and complex script. On top of these three definitions, one may also define a font for use in a specific language or languages.

The complex type CT\_FontCollection is defined in the following manner:

<complexType name="CT\_FontCollection">  
 <sequence>  
 <element name="latin" type="CT\_TextFont" minOccurs="1"  
 maxOccurs="1"/>

<element name="ea" type="CT\_TextFont" minOccurs="1" maxOccurs="1"/>

<element name="cs" type="CT\_TextFont" minOccurs="1" maxOccurs="1"/>

<element name="font" type="CT\_SupplementalFont" minOccurs="0"  
 maxOccurs="unbounded"/>

<element name="extLst" type="CT\_OfficeArtExtensionList"  
 minOccurs="0" maxOccurs="1"/>  
 </sequence>  
</complexType>

#### Supplemental Font

The complex type CT\_SupplementalFont defines an additional font that is used for language specific fonts in themes. For example, one can specify a font that gets used only within the Japanese language context.

The complex type CT\_SupplementalFont is defined in the following manner:

<complexType name="CT\_SupplementalFont">  
 <attribute name="script" type="xsd:string" use="required"/>  
 <attribute name="typeface" type="ST\_TextTypeface" use="required"/>  
</complexType>

#### Format Scheme (Style Matrix)

The complex type CT\_StyleMatrix defines a set of formatting options, which can be referenced by documents that apply a certain style to a given part of an object. For example, in a given shape, say a rectangle, one can reference a themed line style, themed effect, and themed fill that would be theme specific and change when the theme is changed. All of these formatting options are defined within this style matrix. Background fills can also be contained within the style matrix. This is most useful to presentations (but not unique to presentations) which reference different background fills as slide backgrounds. Three sets of each type of formatting are defined, corresponding to subtle, moderate, and intense versions of each style. Combinations of styles are used to create, for example a shape style. An example of this would be a shape style utilizing a subtle fill, moderate line, and intense effect to define the overall look of a shape.

The complex type CT\_StyleMatrix is defined in the following manner:

<complexType name="CT\_StyleMatrix">  
 <sequence>  
 <element name="fillStyleLst" type="CT\_FillStyleList" minOccurs="1"  
 maxOccurs="1"/>

<element name="lnStyleLst" type="CT\_LineStyleList" minOccurs="1"  
 maxOccurs="1"/>

<element name="effectStyleLst" type="CT\_EffectStyleList"  
 minOccurs="1" maxOccurs="1"/>

<element name="bgFillStyleLst" type="CT\_BackgroundFillStyleList"  
 minOccurs="1" maxOccurs="1"/>  
 </sequence>  
 <attribute name="name" type="xsd:string" use="optional" default=""/>  
</complexType>

#### Fill Style List

The complex type CT\_FillStyleList defines a set of three fill types. Currently, only three fill types are used, corresponding to subtle, moderate, and intense fills, but the number of fills that can be defined is unbounded. 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 the subtle slot, a gradient fill in the moderate slot, and an image fill in the intense slot.



Figure 3: Three different fills increasing in relative intensity.

The complex type CT\_FillStyleList is defined in the following manner:

<complexType name="CT\_FillStyleList">  
 <sequence>  
 <group ref="EG\_FillProperties" minOccurs="3" maxOccurs="unbounded"/>  
 </sequence>  
</complexType>

#### Line Style List

The complex type CT\_LineStyleList defines a set of three line styles. As with the fill style list, currently only three styles are utilized corresponding to a subtle line, moderate line, and intense line.

The complex type CT\_LineStyleList is defined in the following manner:

<complexType name="CT\_LineStyleList">  
 <sequence>  
 <element name="ln" type="CT\_LineProperties" minOccurs="3"  
 maxOccurs="unbounded"/>  
 </sequence>  
</complexType>

#### Effect Style List

The complex type CT\_EffectStyleList defines a set of three effect styles. As with the previously mentioned style lists, three styles are currently utilized corresponding to subtle, moderate, and intense effect styles, but the list remains unbounded. In figure 4 we see subtle, moderate, and intense effects applied to a given shape with a blue fill. The subtle effect is, basically, no effect, whereas the moderate effect is a glow surrounding the shape, and the intense effect is a 3-D bevel along with a shadow applied to the shape.

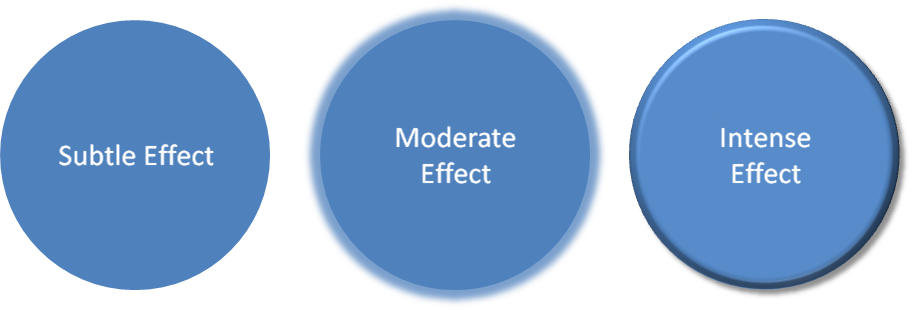


Figure 4: Subtle, moderate, and intense effects applied to a shape that has a blue fill.

The complex type CT\_EffectStyleList is defined in the following manner:

<complexType name="CT\_EffectStyleList">  
 <sequence>  
 <element name="effectStyle" type="CT\_EffectStyleItem" minOccurs="3"  
 maxOccurs="unbounded"/>  
 </sequence>  
</complexType>

#### Effect Style Item

The complex type CT\_EffectStyleItem holds the properties for a given effect style. Within this complex type, one can define a list of effects (blur, shadow, reflection, etc.) along with any 3-D properties that are to be applied to an object. A basic example of how effects can be applied to a shape can be seen in figure 4.

The complex type CT\_EffectStyleItem is defined in the following manner:

<complexType name="CT\_EffectStyleItem">  
 <sequence>  
 <group ref="EG\_EffectProperties" minOccurs="1" maxOccurs="1"/>  
 <element name="scene3d" type="CT\_Scene3D" minOccurs="0"  
 maxOccurs="1"/>  
 <element name="sp3d" type="CT\_Shape3D" minOccurs="0" maxOccurs="1"/>  
 </sequence>  
</complexType>

#### Background Fill Style List

The complex type CT\_BackgroundFillStyleList defines a set of three fill types similar to the fill style list. Again, they define three fill types corresponding to subtle, moderate, and intense background fills but the list itself is unbounded. The background-fills are meant, for example, to be applied to a slide background, or as the background fill in a shape or table.

The complex type CT\_BackgroundFillStyleList is defined in the following manner:

<complexType name="CT\_BackgroundFillStyleList">  
 <sequence>  
 <group ref="EG\_FillProperties" minOccurs="3" maxOccurs="unbounded"/>  
 </sequence>  
</complexType>

#### 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.

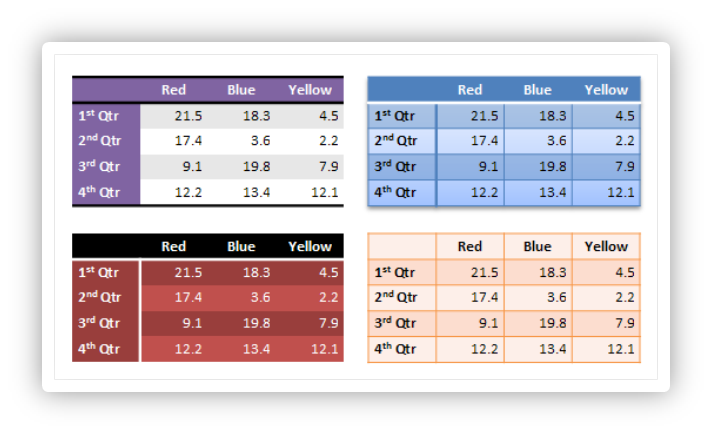


Figure 5: Different table styles in use.

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.

#### 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:

<xsd:complexType name="CT\_Cell3D">  
 <xsd:sequence>  
 <xsd:element name="bevel" type="CT\_Bevel" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="lightRig" type="CT\_LightRig" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>

<xsd:attribute name="prstMaterial" type="ST\_PresetMaterialType"  
 use="optional" default="plastic" />  
</xsd:complexType>

This complex type also holds a CT\_OfficeArtExtension. This complex type is used for future extensibility and will be seen elsewhere throughout the tables area.

#### Themeable Styles

There are three groups and a complex type that 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:

<xsd:group name="EG\_ThemeableFillStyle">  
 <xsd:choice>  
 <xsd:element name="fill" type="CT\_FillProperties" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="fillRef" type="CT\_StyleMatrixReference"  
 minOccurs="1" maxOccurs="1" />  
 /xsd:choice>  
</xsd:group>

<xsd:group name="EG\_ThemeableEffectStyle">  
 <xsd:choice>  
 <xsd:element name="effect" type="CT\_EffectProperties" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="effectRef" type="CT\_StyleMatrixReference"  
 minOccurs="1" maxOccurs="1" />  
 </xsd:choice>  
</xsd:group>

<xsd:group name="EG\_ThemeableFontStyles">  
 <xsd:choice>  
 <xsd:element name="font" type="CT\_FontCollection" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="fontRef" type="CT\_FontReference" minOccurs="1"  
 maxOccurs="1" />  
 </xsd:choice>  
</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:

<xsd:complexType name="CT\_ThemeableLineStyle">  
 <xsd:choice>  
 <xsd:element name="ln" type="CT\_LineProperties" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="lnRef" type="CT\_StyleMatrixReference"  
 minOccurs="1" maxOccurs="1" />  
 </xsd:choice>  
</xsd:complexType>

#### On/Off Property Definition

The simple type ST\_OnOffStyleType defines a type with values of on, off, or default. The default value means to follow the parent settings. This comes into play for a themed property, which means follow what the theme says. For an unthemed property, this means to follow the parent setting in the property inheritance chain.

#### Text Properties

The compelex type CT\_TableStyleTextStyle defines the table text properties that can be styled. The text properties contain a reference to a themeable font style along with bold and italic being enabled or disabled. The CT\_TableStyleTextStyle is defined in the following manner:

<xsd:complexType name="CT\_TableStyleTextStyle">  
 <xsd:sequence>  
 <xsd:group ref="EG\_ThemeableFontStyles" minOccurs="0"  
 maxOccurs="1" />  
 <xsd:group ref="EG\_ColorChoice" minOccurs="0" maxOccurs="1" />  
 <xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"   
 maxOccurs="1" />  
 </xsd:sequence>

<xsd:attribute name="b" type="ST\_OnOffStyleType" use="optional"  
 default="def" />

<xsd:attribute name="i" type="ST\_OnOffStyleType" use="optional"  
 default="def" />  
</xsd:complexType>

#### Cell Border Properties

The complex type CT\_TableCellBorderStyle defines the properties of the borders that can be styled in a table. The border styles can be applied to the following different types of borders in a table:

* left – left border
* right – right border
* top – top border
* bottom – bottom border
* insideH – inner horizontal borders
* insideV – inner vertical borders
* tl2br – diagonal border from top left corner to bottom right corner
* tr2bl – diagonal border from top right corner to bottom left corner

The complex type is defined in the following manner:

<xsd:complexType name="CT\_TableCellBorderStyle">  
 <xsd:sequence>  
 <xsd:element name="left" type="CT\_ThemeableLineStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="right" type="CT\_ThemeableLineStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="top" type="CT\_ThemeableLineStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="bottom" type="CT\_ThemeableLineStyle"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="insideH" type="CT\_ThemeableLineStyle"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="insideV" type="CT\_ThemeableLineStyle"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="tl2br" type="CT\_ThemeableLineStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="tr2bl" type="CT\_ThemeableLineStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

#### Cell Style Properties

The complex type CT\_TableStyleCellStyle contains the definition for cell properties which 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:

<xsd:complexType name="CT\_TableStyleCellStyle">  
 <xsd:sequence>  
 <xsd:element name="tcBdr" type="CT\_TableCellBorderStyle"  
 minOccurs="0" maxOccurs="1" />

<xsd:group ref="EG\_ThemeableFillStyle" minOccurs="0"  
 maxOccurs="1" />  
 <xsd:element name="cell3D" type="CT\_Cell3D" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

#### 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:

<xsd:complexType name="CT\_TableBackgroundStyle">  
 <xsd:sequence>  
 <xsd:group ref="EG\_ThemeableFillStyle" minOccurs="0"  
 maxOccurs="1" />  
 <xsd:group ref="EG\_ThemeableEffectStyle" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

#### 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:

<xsd:complexType name="CT\_TablePartStyle">  
 <xsd:sequence>  
 <xsd:element name="tcTxStyle" type="CT\_TableStyleTextStyle"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="tcStyle" type="CT\_TableStyleCellStyle"   
 minOccurs="0" maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

#### 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
* band1Horizontal – applied when row banding is enabled, this is the first row style, which alternates with band2Horiztonal
* band2Horizontal – applied when row banding is enabled, this is the second row style, which alternates with band1Horizontal
* band1Vertical – applied when column banding is enabled, this is the first column style, which alternates with band2Vertial
* band2Vertical – applied when column banding is enabled, this is the second column style, which alternates with band1Vertical
* lastCol – formatting applied to the last column when last column formatting is enabled
* firstCol – formatting applied to the first column when first column formatting is enabled
* lastRow – formatting applied to the last row when last row formatting is enabled
* firstRow – formatting applied to the first row when first row formatting is enabled
* seCell – formatting applied to the cell in the southeast corner of the table when last column and last row are enabled
* swCell – formatting applied to the cell in the southwest corner of the table when first column and last row are enabled
* neCell – formatting applied to the cell in the northeast corner of the table when the last column and first row are enabled
* nwCell – formatting applied to the cell in the northwest corner of the table when the first column and first row are enabled

The table style is defined in the following manner:

<xsd:complexType name="CT\_TableStyle">  
 <xsd:sequence>  
 <xsd:element name="tblBg" type="CT\_TableBackgroundStyle"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="wholeTbl" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="band1H" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="band2H" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="band1V" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="band2V" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="lastCol" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="firstCol" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="lastRow" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="seCell" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="swCell" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="firstRow" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="neCell" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="nwCell" type="CT\_TablePartStyle" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
 <xsd:attribute name="styleId" type="ST\_Guid" use="required" />  
 <xsd:attribute name="styleName" type="xsd:string" use="required" />"  
</xsd:complexType>

Also contained within the table style are an ID and a name. The name shows up as the name for the table style, and the ID is the unique id (GUID) that is associated with the table style.

#### Table Style List

The final complex type dealing with table styles is simply a list of table styles. Also contained in this list is the default style which gets applied to the table when the a default is to be used. The complex type CT\_TableStyleList is defined in the following manner:

<xsd:complexType name="CT\_TableStyleList">  
 <xsd:sequence>  
 <xsd:element name="tblStyle" type="CT\_TableStyle" minOccurs="0"  
 maxOccurs="unbounded" />  
 </xsd:sequence>  
 <xsd:attribute name="def" type="ST\_Guid" use="required" />  
</xsd:complexType>

## Future Extensibility

This clause provides a high-level overview of the extensibility model for OOXML documents, and a description of packaging conventions in the context of DrawingML and PresentationML. Two main constructs are described: extensibility lists (extLst/ext) and alternate content blocks (AlternateContent).

To illustrate certain points, a number of examples refer to versions of a (fictitious) PresentationML consumer/producer called PML. The 2003 version is called PML 2003; the 2007 version is called PML 2007; and so on.

### Terminology

Here are some terms useful when discussing future extensibility.

* Round tripping involves the interchange of documents between different consumers/producers, as well as between different versions of the same consumer/producer. The pair of consumers/producers can be on the same or different platforms. Consider the case in which a document is created by the PML 2007. This document is then opened by PML 2003, edited, and saved. The edited document is now opened and used by PML 2007. In this case, the document originally created by PML 2007 has been round-tripped through PML 2003. It is also possible to round-trip a document created by PML 2003 through PML 2007.
* A Downrev (or down-level) version of a consumer/producer refers to one that understands an older version of a given schema. An Uprev (or up-level) version of a consumer/producer refers to one that understands a newer version of a given schema. The terms Downrev and Uprev are typically used in relative reference to one another. As an example, let's consider again, the two consumer/producer PML 2003 and PML 2007, where PML 2007 was released sometime after PML 2003, and, consequently, PML 2007 understands a newer revision of the DrawingML schema then does PML 2003. PML 2003 is referred to as the Downrev version while PML 2007 is referred to as the Uprev version. It is assumed that the Downrev version has less capability than the Uprev version.

### What is Future Extensibility?

The main objective of future extensibility is to design an infrastructure that allows the file format to be extended for representation of data structures in future versions of a given consumer/producer.

On the surface, that isn’t hard to do; there could be a special extension bit bucket allocated across every existing schema element, and any future extension could be placed there. However, the problem is more complex than that. The infrastructure must allow document interoperability between current consumers/producers and future consumers/producers, some which have not yet even been designed or built. That is, future extensibility involves building forward compatibility into the document infrastructure while remaining compatible with the current version.

### Future Extensibility Requirements

There are three design goals to be considered: visual fidelity, editability, and security.

* Visual fidelity involves the desire for users of two consumers/producers to see visually the same thing. This seems like a simple design goal to meet, but, in practice, is not easy to achieve. The difference lies in the capabilities of an Uprev and Downrev consumer/producer. Typically, an Uprev consumer/producer has been extended to have new base capabilities that are not present in the Downrev consumer/producer. As such, the Downrev client does not have the base primitives necessary to express visually the new capability introduced in the Uprev consumer/producer.  
    
  Consider the case in which PML 2007 has the capability to highlight text with a given color, while PML 2003 does not. Given the desire to have PML 2007 documents interoperate with PML 2003, it is necessary for PML 2003 to some way to express visually that text highlight. For example, it might insert a picture of the highlighted text instead of inserting the text itself, since PML 2003 does know how to deal with pictures.
* Editability involves the desire for two consumers/producers to be able to edit the same content. Using the highlighted text example from above, despite the fact that PML 2003 and PML 2007 have different capabilities, one would still desire to edit highlighted text a regardless of which version of PML is in use. Again, this becomes difficult when the underlying capabilities of the consumer/producer versions are different.
* Security involves the desire to have multiple representations of the same data synchronized. This desire is referred to as security for the reason that out-of-sync representation can have dire consequences. For example, there might be multiple representations for a sensitive piece of information such as a Social Security number. If this piece of information were edited, it would be critical to keep all alternate representations in sync. What if that information were deleted altogether? If only one representation was deleted but others remained, it would be possible for one to have sensitive information in a document when the intent was to have it deleted.

One solution to try to solve the visual fidelity and editability goals is to have two equivalent representations for the same construct. In the highlighted text example above, a picture of the highlighted text (also called a rasterized version of the highlighted text) is an equivalent representation of the highlighted text itself. One might use the highlighted text representation when the underlying consumer/producer is capable of understanding it; otherwise, the picture version would be used.

Clearly, these design goals compete with each other. While a picture representation of text is capable of capturing full visual fidelity of how extended text looks, obviously that representation doesn't offer the same editability properties of text. One can't manipulate a picture of text nearly as easily as the text itself.

The competing nature of these design characteristics requires that one choose an extensibility construct that offers the best mix of desired characteristics. It will not always be possible to have visual fidelity, editability, and security, at the same time.

### Future Extensibility Constructs

Here are the two extensibility constructs uses to represent extensions in OOXML schemas.

#### extLst/ext

The extLst construct is used for straight-up extension of existing schemas of a non-visual nature. The term straight up refers to the notion that sometimes extension means refining the semantics of existing constructs. In doing so, an extension sometimes overrides the meaning of previous schemas. extLst and ext were not designed for this scenario. Instead of overriding existing meaning, these two constructs purely augment existing schemas. The nature of the augmentation must be such that it does not overlap any semantics embedded in existing schema constructs.

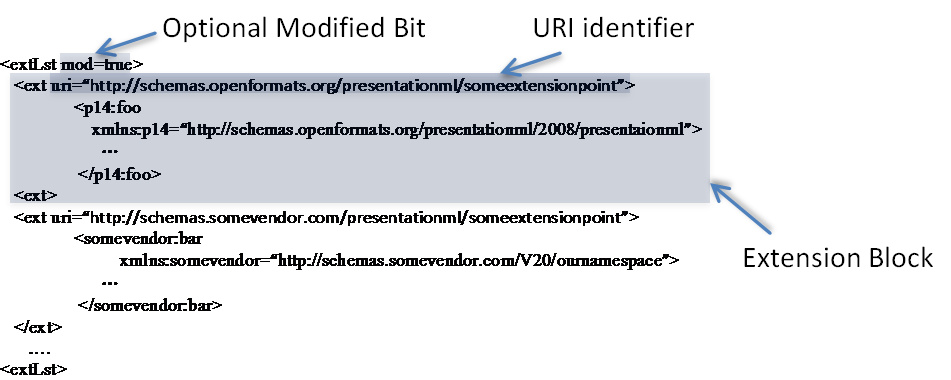
Consider a schema that represents an address, which contain a house number, a street name, a city, a state, and a postal code code. An example of a straight-up extension is the addition of a field that describes whether this address is a business or residential location. This is a straight-up extension because the notion of whether an address is business or residential does not conflict with any information that is embedded in the existing schema. Now let's consider the case in which the Postal Service replaces a purely numeric postal code with one that can contain alphanumeric characters. Such a change would not be a straight-up extension because the new representation conflicts with the old representation of the same data, namely the postal code.

Some extensions are visual in nature. An example would be extending a schema to represent text that has been highlighted. By definition, highlighting text is a visual extension. Contrast that to the case of adding a business or residential classification. The latter does not necessarily involve any visual change to the way data is presented.

The extLst and ext constructs are for extensions of a non-visual nature. The main reason their use is limited to this scenario lies in the fact that they do not offer the capability to create alternative representations of the same data.

##### extLst/ext Syntax

The extLst and ext construct can be placed only at specific locations within the OOXML schemas. Its syntax is as below:



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.

There is no limit to the number of ext extension block constructs. The order of extension blocks can be arbitrary.

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.

##### Round-Trip Behavior of ext Blocks

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.

##### Example

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:



The extLst block is under the non-visual shape properties (i.e., p:nvSpPr). A uri attribute identifies the extension.

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.

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.

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.

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.

Being non-visual in nature, the information in this extension does not directly affect the appearance of the shape.

#### AlternateContent Blocks

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.

##### AlternateContent Syntax

<AlternateContent>  
 <Choice Requires="namespacefoo">  
 <Somemarkup/>  
 <Choice Requires="namespacefoo namespacefoobar">  
 <Somealternatemarkup/>  
 <Fallback>  
 <Choiceoflastresort/>  
</AlternateContent>

The AlternateContent element and its children, Choice and Fallback, are used to provide alternates for specified content. Each Choice element is examined in turn. The Requires attribute specifies a set of space-delimited namespaces that must be understood in order to select that choice. If there is a match between required namespaces and what the consumer understands, the appropriate Choice is returned. If there are multiple possible matches, only the first match is returned. An optional Fallback element can be used, and is utilized as a default when no match occurs.

##### Example

Using PML 2007 and PML 2009, let’s assume that PML 2007 understands some current schema version while PML 2009 will understand some future extended version of the current schema.

PML 2007 knows how to handle connectors:

Now let's suppose that we desire to add a notion of labels on connectors:

Two alternate representations are required: PML 2009's schema has been extended to natively understand how to represent a label on a connector; however, PML 2007 does not. With PML 2007, we approximate this feature by representing the connector and label independently. The label is represented using a textbox. To keep the two elements together, for convenience sake, they are grouped.

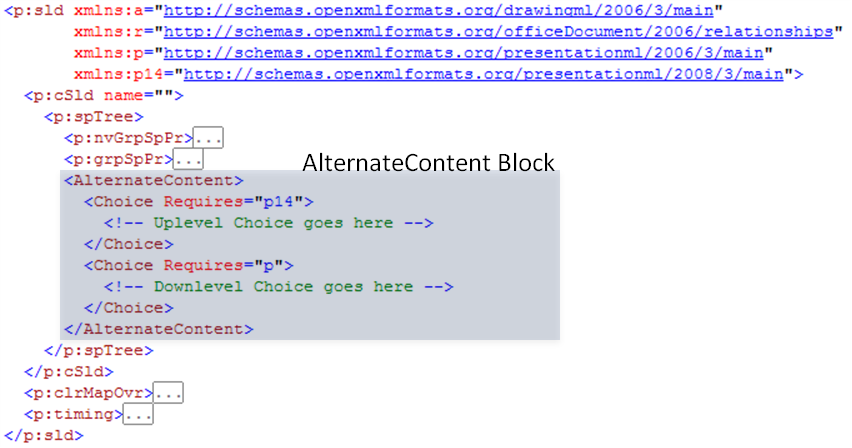
Looking at the corresponding XML, PML 2007's markup looks as follows:



PML 2009 has been extended such that we may represent a label natively:



The final markup putting these choices together is as follows:



##### AlternateContent Round-Trip Behavior

AlternateContent maintains multiple representations for the same content. Consider an extreme case. Using the example above, let's suppose one edited the label using PML 2007. As PML 2007 wouldn't understand future representations, there is no possibility that it could keep PML 2009's markup consistent with the edit performed. Considering a simple case, let's suppose one deleted the label in its entirety. PML 2007 would only know how to delete the corresponding textbox, and would not know how to update the corresponding cntrLblPr.

If this textbox contained sensitive information, one might consider this a security leak. The user's belief is that the information in the textbox was deleted, yet it persists in an alternate representation.

To solve this problem, all AlternateContent choices are discarded when an edit is performed at the location the AlternateContent is placed. It is the consuming client's responsibility to replace the discarded AlternateContent with a new representation.

If an edit to the label occurred in PML 2007, the PML 2009 version is discarded.

If an edit occurs in PML 2009, since PML 2009 understands both PML 2007 and PML 2009 schemas, it is possible for PML 2009 to write an updated AlternateContent Block encompassing an update to both choices.

### Shape Definitions and Attributes

### Introduction

This is a high-level overview of the content described in the dml-shapegeometry.xsd schema.

This aspect of DrawingML deals mainly with the shapes and their attributes, and is broken down into two topics:

* Working with preset shapes
* Defining custom shapes and their properties

### The Coordinate Systems

To specify a shape there are a few high level systems that must first be understood, namely the coordinate systems that will be used. To first specify a shape within a document the document coordinate system must be understood. This system 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 x-coordinate increases, the point will move to the right. As the y-coordinate increases, the point will move downwards. The units of measurement within the document coordinate system are EMUs (91440 EMUs/U.S. inch). In addition to specifying a position for the shape, you must also specify the width and height of the shape, which is called the extent of the shape. This value is again measured in EMUs. To specify these two values, the following transform would be used.

<p:sp>  
 <p:spPr>  
 <a:xfrm>  
 <a:off x="3200400" y="1600200"/>  
 <a:ext cx="1200000" cy="1000000"/>  
 </a:xfrm>  
 </p:spPr>  
</p:sp>

Here we can see that this new shape will be placed at x = 3200400 and y = 1600200 within the document coordinate system. In addition, we see that this shape will have a width of 1200000 EMUs and a height of 1000000 EMUs. This width and height set the bounding box within which the entire shape will be contained.

Now that we have a width and height specified, we can now move into the explanation of the shape coordinate system. The shape coordinate system has both an x and y component and starts with a value of (0,0) in the upper left corner of the shape. The width and height of this coordinate system are specified by the extent of the shape, which was recently specified above, and the units are once again EMUs. This coordinate system is used to define the locations of many of the shape attributes.

The final coordinate system is the path coordinate system which also has both an x and y component and starts with a value of (0,0) in the upper left corner of the shape. Now it must be known that this coordinate system is a unique one in that its units are relative to the specified width and height of the coordinate space. The path coordinate system has exactly the same EMU dimensions as the shape coordinate system but different units. While the shape coordinate system uses EMUs, the path coordinate system uses (1/width) as 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, then the path coordinate (1,1) would be equivalent to (600000,1000000) in the shape coordinate system. The path coordinate system will be better understood later, once the path element is described.

Note that all dimensions and coordinates must be specified using whole numbers.

### Specifying a Preset Shape

Within the Shape Definitions and Attributes section of DrawingML there are many pre-defined shapes that can be used, 187 to be exact. Of course, if the user does not wish to use a preset shape there is always the option of specifying a custom shape which will be described further in section 1.4.

### Defining a Preset Geometry

It is quite easy to specify a preset geometry as that is the whole notion around presets. They are meant to solve the most common cases of shape definition. To specify a heart geometry for instance the following DrawingML code can be used.

<p:sp>  
 <p:spPr>  
 <a:xfrm>  
 <a:off x="1981200" y="533400"/>  
 <a:ext cx="1143000" cy="1066800"/>  
 </a:xfrm>  
 <a:prstGeom prst="heart">  
 </a:prstGeom>  
 </p:spPr>  
</p:sp>



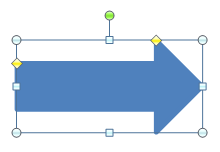
This heart is rendered by the generating application using the custom geometry code for this shape, which is fully documented within ST\_ShapeTypes located in the reference documentation. Thus we see that the user need on specify the preset name to place a shape within their document.

### Adjusting a Preset Geometry

While specifying a preset geometry is convenient and looks good most of the time. There may also be the need for the user to adjust this preset to more closely suit the needs of their document. For this we introduce the notion of adjust values. The preset geometry is built using lines, curves and calculations, just as a custom geometry would be defined. To allow for the adjusting of these preset shapes we have based certain properties of shapes on adjust values rather than concrete dimensions. This means that they can be modified which will in turn modify the geometry of the shape. For instance a simple arrow would be specified using the following DrawingML code.

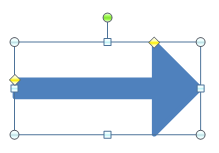
<p:sp>  
 <p:spPr>  
 <a:xfrm>  
 <a:off x="3276600" y="990600"/>  
 <a:ext cx="978408" cy="484632"/>  
 </a:xfrm>

<a:prstGeom prst="rightArrow">  
 <a:avLst>  
 <a:gd name="adj1" fmla="val 50000"/>  
 <a:gd name="adj2" fmla="val 50000"/>  
 </a:avLst>  
 </a:prstGeom>  
 </p:spPr>  
</p:sp>



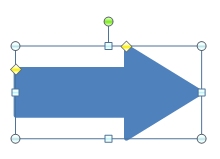
This will specify the basic arrow shown above which might be sufficient for the document needs of the user but it also may not. If this standard arrow is not sufficient then the two adjust values for this shape may be adjusted. For instance, if the body of the arrow is too large then the value for adj1 can be decreased. The following DrawingML code would specify such a case.

<a:gd name="adj1" fmla="val 18553"/>



Similarly if the arrow head itself was too short then the value of adj2 can be increased. The following DrawingML code would specify such a case.

<a:gd name="adj2" fmla="val 81447"/>



Thus it can be seen that while each preset is indeed a preset with a pre-defined geometry, it can be modified. Through the use of adjust values, the user is able to custom fit a preset shape to their document needs without having to specify an entirely custom shape.

Note that the values used here for adjust values have no real units as they are simply input parameters into the equations that make up the shape geometry. More on these equations and their parameters will be discussed in section 1.4.2.

### Specifying a Custom Shape

In addition to the specifying of a preset shape there is also the possibility of a specifying a custom shape. This is accomplished by defining a geometry from a set of construction methods and applying various shape properties to this geometry. This compliments preset shapes, giving the user the opportunity to specify a complete shape with any custom properties that are deemed necessary.

#### Defining the Geometry

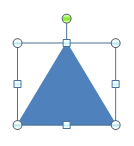
Just like a preset geometry, a custom geometry has a position and a shape bounding box that is specified by the offset and extent transform values. The shape coordinate system is defined by these values as was described in section 1.2 above. The path coordinate system is also partially defined by these in that it has it’s width and height set by these values. The units of the path system however are determined by the specified width and height of the path. A custom geometry with a single path can be specified using the following DrawingML code.

<p:sp>  
 <p:spPr>  
 <a:xfrm>  
 <a:off x="3200400" y="1600200"/>  
 <a:ext cx="1200000" cy="1000000"/>  
 </a:xfrm>

<a:custGeom>  
 <a:pathLst>  
 <a:path w="2" h="2">  
 <a:moveTo>  
 <a:pt x="0" y="2"/>  
 </a:moveTo>

<a:lnTo>  
 <a:pt x="2" y="2"/>  
 </a:lnTo>

<a:lnTo>  
 <a:pt x="1" y="0"/>  
 </a:lnTo>  
 <a:close/>  
 </a:path>  
 </a:pathLst>  
 </a:custGeom>  
 </p:spPr>  
</p:sp>



As can be seen in the above code, the path has a width and height of 2. This means that the path coordinate space will have units of (1/2 \* shape width) for x-coordinate and (1/2 \* shape height) for the y-coordinate. Thus we see that a coordinate of (2,2) in the path coordinate system will be the same as (1200000,1000000) within the shape coordinate system.

To define the shape path above we can see that there are a few different parts to defining this custom shape. 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 multiple paths in it, some filled, some not, some outlined, some not. To define the path we must specify the width, height and thus units for this path via the following DrawingML.

<a:path w="2" h="2">

This sets up the path coordinate system for this path as was previously described. Next we need to move the drawing cursor to the point in this path coordinate system that we wish to start drawing our shape from. The following DrawingML does just that.

<a:moveTo>  
 <a:pt x="0" y="2"/>  
</a:moveTo>

This will move the drawing cursor to the bottom left position (0,2) which is equivalent to (0,1000000) in the shape coordinate system. Following this we can now start by drawing the first line in the shape via the following line.

<a:lnTo>  
 <a:pt x="2" y="2"/>  
</a:lnTo>

This will draw a line from the current drawing cursor position of (0,2) to (2,2) which is the bottom right corner of the path coordinate system. This is equivalent to drawing a line from (0,1000000) to (1200000,1000000) in the shape coordinate system. Now that we have the bottom edge of the triangle drawn we can continue to the final edge in the shape via the following.

<a:lnTo>  
 <a:pt x="1" y="0"/>  
</a:lnTo>

This will draw the final line that will be drawn from the current drawing cursor position of (2,2) to (1,0) which is in the top middle of the path coordinate system. This is equivalent to drawing a line from (1200000,1000000) to (600000,1000000) in the shape coordinate system. Now that most of the triangle has been drawn. It should be noted that since the <close/> element is specified at the end of the path that a line will be drawn from the last point in the path back to the first point in the path. This explains a bit of the existence of the following final element.

<a:close/>

This finalizes the edges of the shape path being specified. Since the fill of this path is set to normal, this path will have a fill no matter if this close tag is specified or not. However, the fact that it is specified determines that there will be a final edge drawn between the final drawing cursor point and the path starting point. Now that the path has been fully specified, this shape can be filled and thus be considered finished.

#### Adjusting the Geometry

Now that we have shown how a custom geometry can be specified we can look at how it might be adjusted. This adjusting is different from the typical resizing that can happen by using the shape transform elements. Using these shape adjusting methods, a shape can be made to have many different resize/adjustment characteristics.

##### Geometry Guides

A guide within a shape is essentially an equation with a set number of inputs and a single output. A guide is used to calculate construction values for a shape and thus can be manipulated to govern the shapes overall geometry. An example of this can be seen in the following DrawingML.

<gdLst>  
 <gd name=”y1” fmla=”\*/ h adj1 100”/>  
</gdLst>

This guide will calculate it’s output based on 3 input parameters and assign this output to a guide named y1. The formula that will be used in the calculation here is the multiply divide formula. The result for this guide will be calculated in the following manner: y1 = ((h \* adj1) / 100). After the result here is calculated, the guide y1 can be used later within the <gdlst> or <path> to calculate further values. That is it can be used as an input for calculating another guide value. These guides then allow for a path to be based off of series of equations rather than static path coordinate values. To use a guide in the defining of a path we would simply specify the following within the path list.

<a:lnTo>  
 <a:pt x="2" y="y1"/>  
</a:lnTo>

This would draw a line to the point (2,y1) where y1 is the calculated result of the guide equation shown above. The drawing of this line will then change based on the input parameters of h and adj1 which are previously calculated guides as well.

Note that while h is a previously calculated guide. It is not calculated for each shape, rather it is a built-in guide that the generating application makes available to the shape.

##### Adjust Handles

To allow for the adjusting UI of a geometry we introduce the notion of an adjust handle. This adjust handle will be linked to adjust values that will then be used as input to the guide equations defined previously. The numerical chain described here will thus directly change the shape of the related geometry. There are two types of adjust handles that can be specified. An XY adjust handle acts in the horizontal/vertical direction and has two related guides, both a horizontal and a vertical respectively. A polar adjust handle acts in a polar manner and has two related guides as well. One guide for the radial width and the other for the radial angle. An adjust handle is specified to have an x and y coordinate as well as these adjust handles. This adjust handle can then be moved around in a generating application’s UI to adjust a pair of guides which will in turn adjust the shape being rendered. An adjust handle can be specified by the following DrawingML.

<ahXY gdRefX="adj1" minX="-2147483647" maxX="2147483647" gdRefY="adj2"  
 minY="-2147483647" maxY="2147483647">  
 <pos x=”x1” y=”y1”/>  
</ahXY>

Above is an XY adjust handle that has two guide references, a min and max allowed position for both the x and y coordinates as well as a position within the shape coordinate system where this adjust handle should be placed.

#### Additional Properties

In addition to specifying the geometry for a shape and all the associated adjustments for it there are also a few other properties that are of special significance. These properties do not act on the geometry of the shape but instead enhance a shape so that it may be used for a more specialized task.

##### Connection Sites

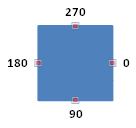
As one may have experienced when trying to draw a diagram with shapes and connections between those shapes, it is quite difficult to move a part of your diagram without entirely redrawing the connections between shapes. For this, there is the notion of connection sites that allow for the specification of specific points within a shape to attach connection shapes to. This allows a user to build a diagram from a set of shapes and connect them together using connection shapes. A connection site is specified within the connection list and consists of an x-coordinate, y-coordinate and an attachment angle. The following DrawingML code defines two connection sites, one at each edge of this triangle.

<a:cxnLst>  
 <a:cxn ang="10800000">  
 <a:pos x="0" y="679622"/>  
 </a:cxn>

<a:cxn ang="0">  
 <a:pos x="1705233" y="679622"/>  
 </a:cxn>  
</a:cxnLst>



The attachment angle works by specifying an angle in 60,000ths of a degree that a connector should attach to. The diagram below shows an actual connection point and the attachment angles that correspond to the sides of this point. This information along with the geometry of the shape is used by the generating applications connector routing algorithm to correctly route connectors around connected shapes.



##### Text Rectangle

Within each shape is a text box that allows for the attaching of text to any given shape. The text rectangle defines where text will reside within the shape. Depending on Auto-fit options that are selected for the body of text attached to this shape the text may intentionally flow outside this text rectangle. It must also be pointed out that this text rectangle will also be the bounding box that is used to compute the geometry of a prstTxWarp. The EMU dimensions of this text rectangle will be used to compute this geometry just like the transform extent element is used to compute the actual shape. The following DrawingML specifies a text rectangle within a shape.

<a:rect l="0" t="0" r="1200000" b="1000000"/>

The text rectangle shown above will have a left edge of 0 x-coordinate, top edge of 0 y-coordinate, right edge of 1200000 x-coordinate and a bottom edge of 1000000 y-coordinate. This will effectively specify a space that is 1200000 EMUs in width and 1000000 EMUs in height.

Note that the edges of this text rectangle can be set so as to allow text to be placed outside the actual geometry of the shape.

## Diagrams

### Introduction

This clause provides a high-level overview of the content described in the following schemas: igxTypes.xsd, igxDataModel.xsd, igxStyleDefinition.xsd, igxLayoutVariables.xsd, igxElementPropertySet.xsd, igxColorTransform.xsd, and igxDiagramDefinition.xsd.

The DrawingML diagram file format is broken down into the following subjects:

* Data Model
* Colors
* Quick Styles
* Layout

The best way to understand the above subjects will be to cover them in the ordering above. The seven schemas can be grouped into the subjects as seen in table 1 below.

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Colors | Quick Styles | Layout |
| igxDataModel.xsd | igxColorTransform.xsd | igxStyleDefinition.xsd | igxTypes.xsd |
| igxElementPropertySet.xsd |  |  | igxDiagramdefinition.xsd |
|  |  |  | igxLayoutVariables.xsd |
|  |  |  | igxElementPropertySet.xsd |

Table 1: IGX schemas grouped by subject.

### Element Property Set

The schema igxElementPropertySet.xsd defines a complex type, CT\_ElemPropSet, which is a catch-all for holding element properties and customizations, and is used throughout certain complex types in DrawingML. This type contains many properties, and these are are explained in subsequent subclauses. The definition of CT\_ElemPropSet is as follows:

<xsd:complexType name="CT\_ElemPropSet">  
 <xsd:sequence>  
 <xsd:element name="presLayoutVars"   
 type="CT\_LayoutVariablePropertySet" minOccurs="0"  
 maxOccurs="1" />  
 <xsd:element name="style" type="a:CT\_ShapeStyle"  
 minOccurs="0" maxOccurs="1" />  
 </xsd:sequence>

<xsd:attribute name="presAssocID" type="ST\_ModelId" use="optional" />  
 <xsd:attribute name="presName" type="xsd:string" use="optional" />  
 <xsd:attribute name="presStyleLbl" type="xsd:string" use="optional" />  
 <xsd:attribute name="presStyleIdx" type="xsd:int" use="optional" />  
 <xsd:attribute name="presStyleCnt" type="xsd:int" use="optional" />

<xsd:attribute name="loTypeId" type="xsd:string" use="optional" />  
 <xsd:attribute name="loCatId" type="xsd:string" use="optional" />  
 <xsd:attribute name="qsTypeId" type="xsd:string" use="optional" />  
 <xsd:attribute name="qsCatId" type="xsd:string" use="optional" />  
 <xsd:attribute name="csTypeId" type="xsd:string" use="optional" />

<xsd:attribute name="csCatId" type="xsd:string" use="optional" />  
 <xsd:attribute name="coherent3DOff" type="xsd:boolean" use="optional" />  
 <xsd:attribute name="phldrT" type="xsd:string" use="optional" />  
 <xsd:attribute name="phldr" type="xsd:boolean" use="optional" />  
 <xsd:attribute name="custAng" type="xsd:int" use="optional" />

<xsd:attribute name="custFlipVert" type="xsd:boolean" use="optional" />  
 <xsd:attribute name="custFlipHor" type="xsd:boolean" use="optional" />  
 <xsd:attribute name="custSzX" type="xsd:int" use="optional" />  
 <xsd:attribute name="custSzY" type="xsd:int" use="optional" />  
 <xsd:attribute name="custScaleX" type="xsd:int" use="optional" />

<xsd:attribute name="custScaleY" type="xsd:int" use="optional" />  
 <xsd:attribute name="custT" type="xsd:boolean" use="optional" />  
 <xsd:attribute name="custLinFactX" type="xsd:int" use="optional" />  
 <xsd:attribute name="custLinFactY" type="xsd:int" use="optional" />  
 <xsd:attribute name="custLinFactNeighborX" type="xsd:int" use="optional" />

<xsd:attribute name="custLinFactNeighborY" type="xsd:int" use="optional" />  
 <xsd:attribute name="custRadScaleRad" type="xsd:int" use="optional" />  
 <xsd:attribute name="custRadScaleInc" type="xsd:int" use="optional" />  
</xsd:complexType>

#### 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.

#### 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.
* qsTypeID – The ID of the current style type.
* qaCatID – The ID of the current style category.
* csTypeID – The ID of the current color transform.
* csCatID – The ID of the current color transform category.
* coherent3Doff – Enables or disables coherent 3D behavior for styles that have such behavior defined.

#### Semantic Element Properties

The following attributes relate to the semantic element properties:

* phldrT – The text used for display in the element if the placeholder flag is set to true. If this field is not set, then the default placeholder text will be used.
* phldr – Indicates that the element is a placeholder or sample item.

#### Customization Properties

The following are customization properties or tweaks:

* custAng – The amount rotation is customized by, in 60,000th of a degree.
* custFlipVert – Vertical flip.
* custFlipHor – Horizontal flip.
* custSzX – Fixed width override for a shape, in emus.
* custSzY – Fixed height override for a shape, in emus.
* custScaleX – Amount that the width is scaled by, in 1,000th of a percent.
* custScaleY – Amount that the height is scaled by, in 1,000th of a percent.
* custT – If text has been customized then layout will no longer change it.
* custLinFactX – A percentage of the shape width that is used for offsetting the shape, in 1,000th of a percent.
* custLinFactY – A percentage of the shape height that is used for offsetting the shape, in 1,000th of a percent.
* custLinFactNeighborX – A percentage of the neighbor’s height used for offsetting the shape, in 1,000th of a percent.
* custLinFactNeighborY – A percentage of the neighbor’s height used for offsetting the shape, in 1,000th of a percent.
* custRadScaleRad – Defines how much the radius has been scaled by, in 1,000th of a percent.
* custRadScaleInc – Defines how much the include angle has been scaled by, in 1,000th of a percent.

#### Data Model

The schema igxDataModel.xsd defines the data model in a diagram. The purpose of the data model is twofold. The first use of the data model is to hold the information contained in a diagram. For example, in figure 1 below, the purpose of the data model would be to hold the information, “one”, “two” and “three” for the diagram.

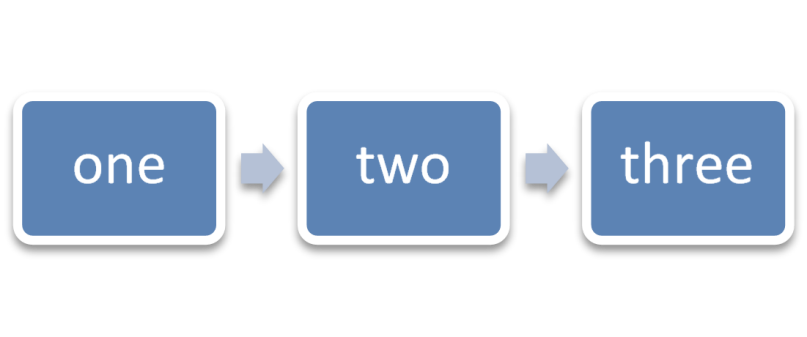


Figure 6: Example diagram with data.

The second use of the data model is to define an initial state of the diagram. This initial state consists of what can be thought of as placeholder data, which an application uses to display a diagram initially before any data has been entered. Figure 2 shows an example of what a diagram might look like in an initial state containing three empty nodes. In this example, the placeholder data consists of three nodes and two connections, which will be explained shortly.

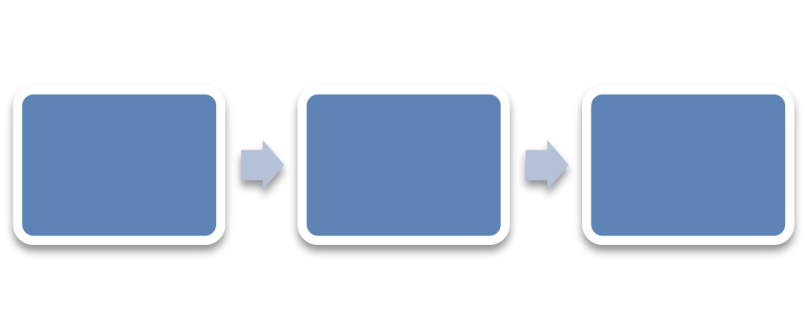


Figure 7: An empty diagram in its initial state.

#### Structural Elements

##### Element Type

There is a single simple type, ST\_PtType, used to define a type of element.; this is defined later. Seven different types of elements are available to the user:

* node – A model element.
* asst – This is used in hierarchy diagrams and represents an assistant element.
* doc – A document element.
* pres – A presentation element.
* parTrans – A parent transition element
* sibTrans – A sibling transition element.
* unknown – An element type which is used to maintain backward compatibility.

##### 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.

##### 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:

<xsd:complexType name="CT\_Pt">  
 <xsd:sequence>  
 <xsd:element name="prSet" type="CT\_ElemPropSet" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="spPr" type="a:CT\_ShapeProperties"   
 minOccurs="0" maxOccurs="1 />

<xsd:element name="style" type="a:CT\_ShapeStyle"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="t" type="a:CT\_TextBody" minOccurs="0"   
 maxOccurs="1" />  
 </xsd:sequence>

<xsd:attribute name="modelId" type="ST\_ModelId" use="required" />

<xsd:attribute name="type" type="ST\_PtType" use="optional"   
 default="node" />

<xsd:attribute name="cxnId" type="ST\_ModelId" use="optional"   
 default="0" />  
</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.

##### Relationship

A relationship is a connection between any two model elements. An example of where a relationship would be used can be seen in figures 1 and 2. In each of those examples, the arrows between the nodes have relationships defined. A relationship is defined as follows:

<xsd:complexType name="CT\_Cxn>  
 <xsd:attribute name="modelId" type="ST\_ModelId" use="required" />  
 <xsd:attribute name="type" type="ST\_CxnType" use="optional"   
 default="parOf" />  
 <xsd:attribute name="srcId" type="ST\_ModelId" use="required" />  
 <xsd:attribute name="destId" type="ST\_ModelId" use="required" />

<xsd:attribute name="srcOrd" type="xsd:unsignedInt" use="required" />  
 <xsd:attribute name="destOrd" type="xsd:unsignedInt" use="required" />  
 <xsd:attribute name="parTransId" type="ST\_ModelId" use="optional"   
 default="0" />  
 <xsd:attribute name="sibTransId" type="ST\_ModelId" use="optional"   
 default="0" />  
 <xsd:attribute name="presId" type="xsd:string" use="optional"   
 default="" />  
</xsd:complexType>

The relationship, as with the element, has a unique id associated with it referred to as the modelID. The srcId and destId attributes refer to ids of the source element and destination element, respectively, that this relationship is defined between.

The srcOrd and destOrd refer to the ordinality of siblings for a given connection. For example, if a node had 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, and then A.

The presId attribute contains the presentation that is associated with this particular relationship.

##### Element List

The complex type CT\_PtList is simply a sequence of elements. Its definition is as follows:

<xsd:complexType name="CT\_PtList">  
 <xsd:sequence>  
 <xsd:element name="pt" type="CT\_Pt" minOccurs="0" maxOccurs="unbounded"/>  
 </xsd:sequence>  
</xsd:complexType>

##### Relationship List

This complex type, CT\_CxnList, is simply a sequence of connections. Its definition is as follows:

<xsd:complexType name="CT\_CxnList" oxsd:cname="Relationships">  
 <xsd:sequence>  
 <xsd:element name="cxn" type="CT\_Cxn" minOccurs="0"   
 maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

##### Data Model

The complex type CT\_DataModel defines the data model and contains a sequence of elements. It is defined as follows:

<xsd:complexType name="CT\_DataModel">  
 <xsd:sequence oxsd:emitArgs="flattenSequence">  
 <xsd:element name="ptLst" type="CT\_PtList" />  
 <xsd:element name="cxnLst" type="CT\_CxnList" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="bg" type="a:CT\_BackgroundFormatting"   
 minOccurs="0" />

<xsd:element name="whole" type="a:CT\_WholeE2oFormatting"   
 minOccurs="0" />  
 </xsd:sequence>  
</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.

### 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.

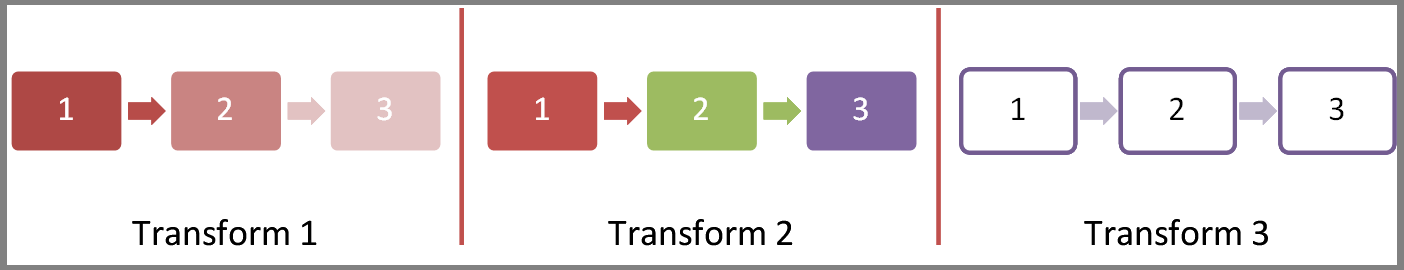


Figure 8: Different examples of a color transform applied to a diagram

#### Structural Elements

The structural elements which come together to create a color transform, or rather, the complex type CT\_ColorTransform, are as follows:

* CT\_CTName
* CT\_CTDescription
* CT\_CTCategory
* CT\_CTCategories
* ST\_ClrAppMethod
* ST\_HueDir
* CT\_Colors
* CT\_CTStyleLabel
* CT\_CTVersion
* CT\_ColorTransformHeader
* CT\_ColorTransformHeaderLst

The complex types CT\_CTName (name), CT\_CTDescription (description), CT\_CTCategory (category), and CT\_CTCategories (list of categories) work together to name, describe and categorize the particular color transform. These types are mirrored elsewhere throughout DrawingML in the different subjects in order to perform the same tasks of naming, describing, and categorizing.

The name consists simply of two strings, one of a name for the color transform, which is required, and an optional language tag. The language allows someone to specify a language for a given title. It is possible to specify multiple titles that are language dependant. The description also has the optional language attribute as in the name, along with a second required string attribute which holds the actual description. The usage of this is exactly the same as within CT\_CTName. CT\_CTName and CT\_CTDescription are defined in the following way:

<xsd:complexType name="CT\_CTName">  
 <xsd:attribute name="lang type="xsd:string" use="optional" />  
 <xsd:attribute name="val" type="xsd:string" use="required" />  
</xsd:complexType>

<xsd:complexType name="CT\_CTDescription">  
 <xsd:attribute name="lang" type="xsd:string" use="optional" />  
 <xsd:attribute name="val" type="xsd:string" use="required" />  
</xsd:complexType>

The category and categories complex types , CT\_CTCategory and CT\_CTCategories, respectively, define how the color transform is categorized within the user interface of the application. The category contains a name, or type, along with a priority that defines the ordering of the color transform. The lower the priority, the earlier in the category it will display. If there is a tie, the unique id associated with the color transform will decide the order alphabetically. CT\_CTCategories is simply a list of CT\_CTCategory. The two complex types are defined as follows:

<xsd:complexType name="CT\_CTCategory">  
 <xsd:attribute name="type" type="xsd:anyURI" use="required" />  
 <xsd:attribute name="pri" type="xsd:unsignedInt" use="required"/>  
</xsd:complexType>

<xsd:complexType name="T\_CTCategories">  
 <xsd:sequence minOccurs="0" maxOccurs="unbounded">  
 <xsd:element name="cat" type="CT\_CTCategory" minOccurs="0"   
 maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

##### 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. 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.

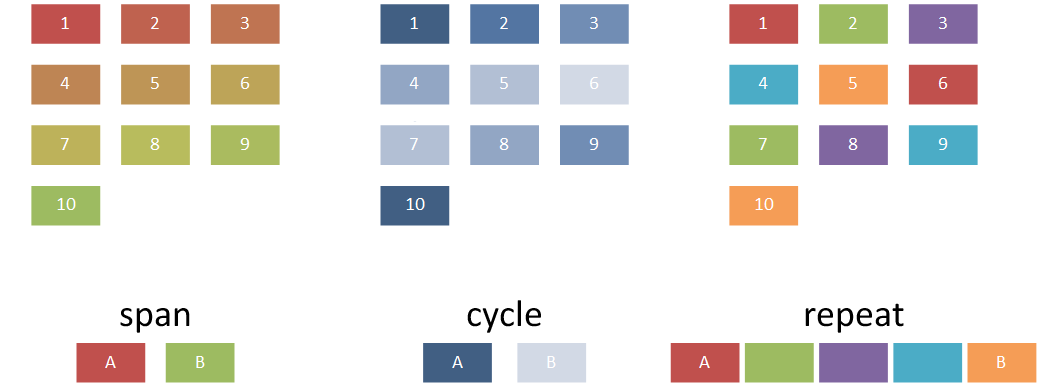


Figure 9: Examples of the three different ways a color transform is applied to a diagram.

##### Hue Direction

The simple type ST\_HueDir defines the direction of a hue color shift around the 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.

##### Colors

The complex type CT\_Colors holds the actual color values that are to be applied to a given diagram and how those colors are to be applied. It contains the color application method and hue shift direction, and is defined as follows:

<xsd:complexType name="CT\_Colors>  
 <xsd:sequence>  
 <xsd:group ref="a:EG\_ColorChoice" minOccurs="0" maxOccurs="unbounded" />  
 </xsd:sequence>

<xsd:attribute name="meth" type="ST\_ClrAppMethod" use="optional"   
 default="span" />

<xsd:attribute name="hueDir" type="ST\_HueDir" use="optional"   
 default="cw" />  
</xsd:complexType>

The sequence of colors is defined via the sequence of EG\_ColorChoices.

##### Style Label

The complex type CT\_CTStyleLabel packages together colors for the different pieces of a diagram. There are six different aspects to a diagram that can be colored independently of one another. Each of the six parts is of type CT\_Colors. They are:

* Fill Colors – The colors that actually fill the shapes in the diagram
* Line Colors – The colors of the lines on the shapes in the diagram.
* Effect Colors – The colors of the effects applied to the shapes within the diagram (eg. Glow).
* Text Line Colors – The colors of the lines on the text within the diagram.
* Text Fill Colors – The color of the text within the diagram.
* Text Effect Colors – The colors of the effects applied to the text within the diagram.

The final piece of a style label is simply its name, which is a string. CT\_CTStyleLabel is defined as follows:

<xsd:complexType name="CT\_CTStyleLabel">  
 <xsd:sequence>  
 <xsd:element name="fillClrLst" type="CT\_Colors"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="linClrLst" type="CT\_Colors"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="effectClrLst" type="CT\_Colors"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="txLinClrLst" type="CT\_Colors"  
 minOccurs="0" maxOccurs="1" />

<xsd:element name="txFillClrLst" type="CT\_Colors"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="txEffectClrLst" type="CT\_Colors"   
 minOccurs="0" maxOccurs="1" />  
 </xsd:sequence>  
 <xsd:attribute name="name" type="xsd:string" use="required" />  
</xsd:complexType>

##### Version

The simple type ST\_CTVersion defines the minimum version of an application that the color transform will work with. The version corresponds to build numbers in the major.minor.build.revision format and is defined as follows:

[0-9]?[0-9])?(\.[0-9]?[0-9])?(\.[0-9]{4})?(\.[0-9]{4}

##### Color Transform

The complex type CT\_ColorTransform brings together all of the pieces into one cohesive structure. This is the actual definition of a color transform, which can be applied to any diagram; it is defined as follows:

<xsd:complexType name="CT\_ColorTransform">  
 <xsd:sequence>  
 <xsd:element name="title" type="CT\_CTName" minOccurs="0"   
 maxOccurs="unbounded" />

<xsd:element name="desc" type="CT\_CTDescription"   
 minOccurs="0" maxOccurs="unbounded" />

<xsd:element name="catLst" type="CT\_CTCategories"   
 minOccurs="0" />

<xsd:element name="styleLbl type="CT\_CTStyleLabel"   
 minOccurs="0" maxOccurs="unbounded" odoc />  
 </xsd:sequence>

<xsd:attribute name="uniqueId" type="xsd:anyURI" use="optional"/>  
 <xsd:attribute name="minVer" type="ST\_CTVersion" use="optional"   
 default="12.0" />  
</xsd:complexType>

A color transform contains a title, description, category list, and style label in a sequence along with a unique id and a minimum version.

#### Color Transform Header

Two complex types, CT\_ColorTransformHeader and CT\_ColortransformHeaderLst, help alleviate potential performance concerns with initially loading in a large number of color transforms. The header information contains the minimum information required to load a color transform into the application. Because of this, color transforms themselves can be loaded only when needed, and other initialization work can progress quickly without loading unneeded information.

CT\_ColorTransformHeader contains information about the title of the color transform, the description, how it is categorized, the unique id, minimum version, and resId. It is defined in the following way:

<xsd:complexType name="CT\_ColorTransformHeader">  
 <xsd:sequence>  
 <xsd:element name="title" type="CT\_CTName" minOccurs="1"   
 maxOccurs="unbounded" />

<xsd:element name="desc" type="CT\_CTDescription"   
 minOccurs="1" maxOccurs="unbounded" />

<xsd:element name="catLst" type="T\_CTCategories"   
 minOccurs="0" o:cname="categories" />  
 </xsd:sequence>

<xsd:attribute name="uniqueId" type="xsd:anyURI" use="required"/>  
 <xsd:attribute name="minVer" type="ST\_CTVersion" use="optional"   
 default="12.0" />   
 <xsd:attribute name="resId" type="xsd:int" use="optional"   
 default="0" />  
</xsd:complexType>

The complex type CT\_ColorTransformHeaderLst simply contains a list of color transform headers. It is defined as follows:

<xsd:complexType name="CT\_ColorTransformHeaderLst">  
 <xsd:sequence>  
 <xsd:element name="colorsDefHdr" type="CT\_ColorTransformHeader"   
 minOccurs="0" maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

### Style Definition

A style definition is similar to a color transform. A style definition defines items such as the font used, the thickness of a contour line, 3-D properties on the diagram, among other things. Style definitions work in combination with color transforms to give an overall look and feel for the diagram. Some examples of quick styles in use are in figure 5.

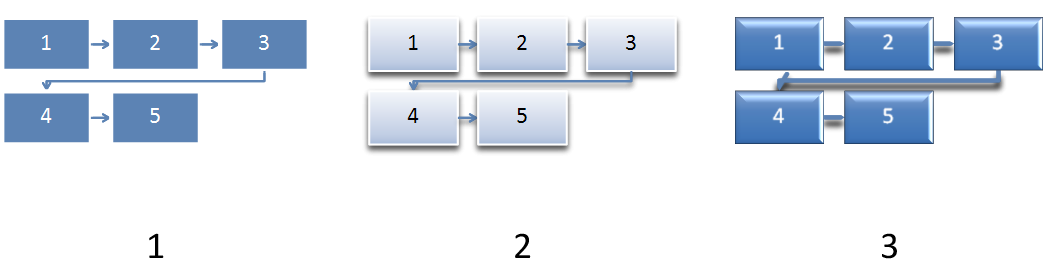


Figure 10: Examples of using three different style definitions on a diagram.

#### Structural Elements

The structural elements which come together to create a style definition are as follows:

* CT\_SDName
* CT\_SDDescription
* CT\_SDCategory
* CT\_SDCategories
* CT\_TextProps
* CT\_StyleLabel
* ST\_SDVersion
* CT\_StyleDefinition
* CT\_StyleDefinitionHeader
* CT\_StyleDefinitionHeaderLst

CT\_SDName, CT\_SDDescription, CT\_SDCategory, CT\_SDCategories, and ST\_SDVersion are all defined exactly as they are within color transforms. These types were recreated within the style definition area to allow slight differentiations to be made, although, at this time they are defined exactly the same.

The text properties, style label, and then the style definition combine together to create a style definition, which is applied to a diagram.

##### Text Properties

The complex type CT\_TextProps holds any 3-D associated properties of the text that is to be held in the diagram. A CT\_TextProps is defined as follows:

<xsd:complexType name="CT\_TextProps">  
 <xsd:sequence>  
 <xsd:group ref="a:EG\_Text3D" minOccurs="0" maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

All that is contained within the text properties is an EG\_Text3D complex type. The usage of the text properties complex type allows 3-D text properties to be defined for a diagram style.

##### Style Label

The style label contains information pertaining to the styleable elements within a diagram. These elements include the shape properties and text properties along with any references to the style matrix or document theme. The shape properties are defined in two different ways: the scene3d element that pertains to the scene on a whole (and includes lighting effects, rotations, and the like), and any 3-D and material settings are held in the sp3d element. CT\_StyleLabel is defined as follows:

<xsd:complexType name="CT\_StyleLabel">  
 <xsd:sequence>  
 <xsd:element name="scene3d" type="a:CT\_Scene3D"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="sp3d" type="a:CT\_Shape3D" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="txPr" type="CT\_TextProps" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="style" type="a:CT\_ShapeStyle"   
 minOccurs="0" maxOccurs="1" />  
 </xsd:sequence>  
 <xsd:attribute name="name" type="xsd:string" use="required" />  
</xsd:complexType>

As with many other complex types, the style label has an attribute reserved for a name. This is simply a string that names the particular style label. This style label can then be referenced from within a diagram definition, as we shall see below.

Note that the scene3d element contained within a style label acts on the level of an individual shape, rather than the diagram as a whole. A second scene3d element is defined within the style definition; that acts on the diagram level and allows for scene coherent 3-D to be applied to the diagram. A style definition contains a style label.

##### Style Definition

The style definition complex type, CT\_StyleDefinition, is the root element used to define a style definition. As with the root element of a color transform, within the style definition there exists a title, description, category, unique id, and minimum version number. These elements serve the same purpose as they do within the color transform.

The interesting aspects of a style definition complex type are that it holds a style label, another style index (which is contained within the style label as well), and another scene3d (which is also contained within the style label as has been previously mentioned). The scene3d element applies to the diagram on a whole and allows for scene coherent 3-D to be applied to the diagram. The duplication of the style index is two-fold. If a style index is not defined within the style label, then the default style index, or rather, the index defined in this complex typem is used. Since a diagram definition can reference a style label, and not a style definition, the style index is also required within the style label. A CT\_StyleDefniition is defined as follows:

<xsd:complexType name="CT\_StyleDefinition">  
 <xsd:sequence>  
 <xsd:element name="title" type="CT\_SDName" minOccurs="0"   
 maxOccurs="unbounded" />

<xsd:element name="desc" type="CT\_SDDescription"   
 minOccurs="0" maxOccurs="unbounded" />

<xsd:element name="catLst" type="CT\_SDCategories"   
 minOccurs="0" />

<xsd:element name="scene3d" type="a:CT\_Scene3D"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="style" type="a:CT\_ShapeStyle"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="styleLbl" type="CT\_StyleLabel"   
 minOccurs="1" maxOccurs="unbounded" />  
 </xsd:sequence>

<xsd:attribute name="uniqueId" type="xsd:anyURI" use="optional"/>  
 <xsd:attribute name="minVer" type="ST\_SDVersion" use="optional" `   
 default="12.0" />   
</xsd:complexType>

#### Style Definition Header

The complex types CT\_StyleDefinitionHeader and CT\_StyleDefinitionHeaderLst perform the same function as the two complex header types in color transforms. They are used to pre-load required information so the actual loading of the style definition can happen only when needed.

### Layout

The single largest aspect of DrawingML is Layout. Ultimately, layout is responsible for defining all aspects of the diagram outside of color and style. It defines how the diagram looks, how it behaves, and how the data is to be mapped. Figure 6 shows different examples of how layout works to create a diagram holding the data ‘1’, ‘2’, and ‘3’.

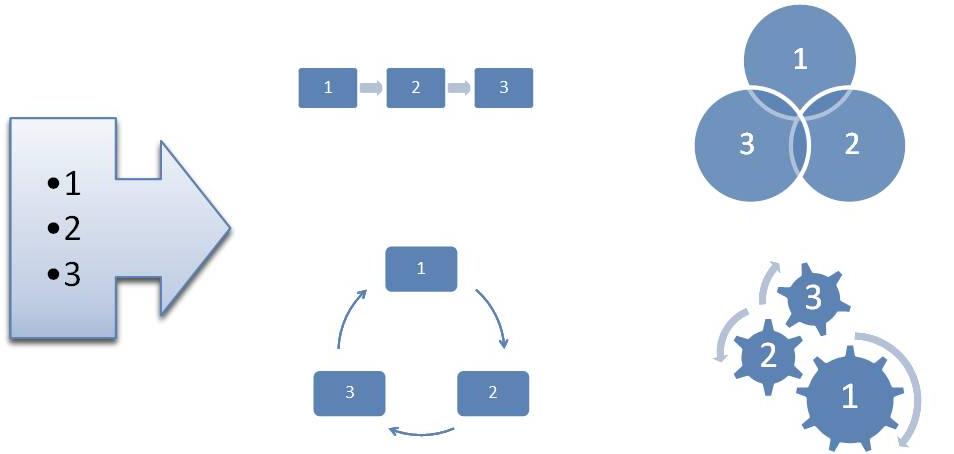


Figure 11: Different layouts mapping the data 1, 2, and 3 to four different diagrams

There are two aspects to layout: defining the numerous complex types utilized by diagram definitions, and the diagram definitions themselves. Diagram definitions are a fundamental part of layout, and utilize everything in order to define new diagrams.

#### Basic Layout Types

There are a very large number of simple types defined in this subject. These simple types are all associated with properties of a diagram that can be modified to create a desired behavior. These simple types, along with an explanation of what each does, follow.

##### Algorithm Type

ST\_AlgorithmType is responsible for which algorithm will be used to layout the diagram. The algorithm layout chosen determines if the diagram behaves as if it were a simple list, a circular cycle, or some other type of diagram. The algorithms available are:

* unknown – an unknown algorithm type
* 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

##### 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

##### Boolean Operators

Boolean type operators, defined by ST\_BoolOperator, are for layout.

* none – no operator defined
* equ – ‘equal to’ operator, returns true if the two compared values are equal, false otherwise
* gte – ‘greater than or equal to’ operator
* lte – ‘less than or equal to’ operator

##### Child Order Type

The simple type ST\_ChildOrderType specifices the child order type for a layout node.

* b – bottom
* t – top

##### Constraint Types

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:

* unknown – an unknown constraint
* alignOff – specifies the alignment offset
* begMarg – specifies the beginning margin
* bendDist – specifies the distance at which a connector will bend
* begPad – specifies the distance between the edge of the transition node and the connector shape
* b – bottom alignment
* bMarg – the bottom margin
* bOff – specifies the amount of offset relative to the bottom of the node
* ctrX – specifies the center of the node in the X-direction
* ctrXOff – specifies the amount of offset of the center of the node in the X-direction
* ctrY – specifies the center of the node in the Y-direction
* ctrYOff – specifies the amount of offset of the center of the node in the Y-direction
* connDist – specifies the distance between connectors. Intended for use with boolean and reference constraints. Overrides values specified in the layout definition part
* diam – specifies the diameter, and is used within the cycle algorithm type
* endMarg – specifies the end margins
* endPad – specifies the distance between the edge of the transition node and the connector shape
* h – specifies the height
* hArH – specifies the arrowhead height
* hOff – specifies the height offset
* l – specifies the left
* lMarg – specifies the amount of left margin
* lOff – specifies the amount of left offset
* r – specifies right
* rMarg – specifies the amount of right margin
* rOff – specifies the amount of right offset
* primFontSz – specifies the primary font size
* pyraAcctRatio - specifies the fraction of the width of the diagram that is reserved for the fly-outs at their shortest distance
* secFontSiz – specifies the secondary font size
* stemThick – specifies the thickness of the shaft on an arrow
* t – specifies the top
* tMarg – specifies the amount of top margin
* tOff – specifies the amount of top offset
* userA through userZ – these are a group of user defined variables which can store values for later use within the diagram definition
* w – specifies the width
* wArH – specifies the width of an arrowhead
* wOff – specifies the amount of width offset

##### Constraint Relationships

The simple type ST\_ConstraintRelationship defines the application of retrieval data the constraint is applied to. The following relationships are available:

* self – the constraint is applicable to the current point
* ch – the constraint is applicable to a child of the current point
* des – the constraint is applicable to a descendant of the current point

##### Element Type

The simple type ST\_ElementType defines the type of element, or point which get created and how they are created from the data at hand. The different ways to pull from the data to create points are as follows:

* all – use all of the data points, nodes and transitions
* doc – use the document level, or root data point
* node – use only data nodes input by the user
* norm – in place for extensibility and behaves exactly opposite of the asst element type
* nonNorm – in place for extensibility and behaves exactly opposite of the nonAsst element type
* asst – use assistant data nodes within hierarchy algorithm
* nonAsst – use non-assistant nodes within the hierarchy algorithm
* parTrans – Use only parent transitions between nodes. Parent transitions are similar to sibling transitions, except that they represent parent/child relationships. Parent transitions are most commonly used in hierarchy diagrams, such as organization charts, to draw lines between parent and child nodes.
* pres – specifies that the node is related to the presentation level
* sibTrans – Use only sibling transitions between data nodes. These transitions represent sibling relationships between nodes, and are frequently mapped to arrows between shapes in the drawing. A sibTrans value is sometimes used to create white space between nodes.

##### Parameter ID

The simple type ST\_ParameterId defines parameters that can be used to modify the behavior or algorithms. The modifications are as follows:

* horzAlign – specifies the horizontal alignment
* vertAlign – specifies the vertical alignment
* chDir – specifies the child direction
* chAlign – specifies the alignment of the children
* secChAlign – specifies a secondary child alignment
* linDir – specifies whether children are arranged from left to right, right to left, top to bottom, or bottom to top
* secLinDir – specifies a secondary linear direction in which children are
* arranged from left to right, right to left, top to bottom, or bottom to top
* stElem – specifies the point type of the layout node to use as the first shape in the cycle
* bendPt – specifies where the bend point is to be located on connectors
* connRout – specifies whether the connector is drawn as a single straight line, orthogonal lines with a single bend, or a curve that uses the diam constraint
* begSty – specifies whether the beginning of the connector has an arrowhead
* endSty – specifies whether the end of the connector has an arrowhead
* dim – specifies the connector dimension, 2-D, 3-D, or custom
* rotPath - if rotPath=alongPath, the algorithm rotates all children perpendicular to the line from the cycle’s center to the child node; otherwise they are not rotated. The alongPath value does not take rotation into account when determining if shapes overlap
* ctrShpMap - None specifies to place nodes around a circle. First node (fNode) specifies to place the first node in the center and the remaining nodes around the circle
* nodeHorzAlign – specifies the horizontal node alignment
* nodeVertAlign – specifies the vertical node alignment
* fallback – specifies if the fallback occurs in a single dimension (e.g. vertically) or if it occurs in two dimensions
* txDir – specifies where the text of the first node starts
* pyraAcctPos – specifies the placement of the fly-out grandchildren
* pyraAcctTxMar – specifies the placement of one edge of the child text
* txBlDir – Specifies the text block direction, vertical or horizontal
* txAnchorHorz – Specifies the horizontal text anchor position.
* txAnchorVert – Specifies the vertical text anchor position.
* txAnchorHorzCh – Specifies the horizontal text anchor position for child text.
* txAnchorVertCh – Specifies the vertical text anchor position for child text.
* parTxLTRAlign – Specifies the paragraph alignment of parent text when the shape has only parent text. This parameter applies when the text direction is left to right.
* partTxRTLAlign – Specifies the paragraph alignment of parent text when the shape has only parent text. This parameter applies when the text direction is right to left.
* shpTxLTRAlignCh – Specifies the paragraph alignment of all text within the shape when the shape contains both parent and child text. This parameter applies when the text direction is left to right.
* shpTxRTLAlignCh – Specifies the paragraph alignment of all text within the shape when the shape contains both parent and child text. This parameter applies when the text direction is right to left.
* autoTxRot – specifies how text is oriented relative to the shape
* grDir – Specifies the direction of growth for the snake algorithm.
* flowDir – specifies weather nodes are arranged in rows or columns for the snake algorithm.
* contDir – Specifies the direction of subsequent rows or columns in the snake algorithm.
* bkpt – specifies the point at which the diagram starts to snake
* off – specifies whether each row and column is centered or offset from the previous row or column
* hierAlign – specifies the alignment of the hierarchy
* bkPtFixedVal – specifies where the sname should break if bkpt is set to fixed
* stBulletLvl – Specifies the level at which to start using bullets for incoming text.
* stAng – Specifies the angle at which the first shape is placed. Angles are in degrees, measured clockwise from a line pointing straight upward from the center of the cycle.
* spanAng – Specifies the angle the cycle spans. Final shapealign text is placed at stAng+spanAng, unless spanAng=360. In that case, the algorithm places the text so that shapes do not overlap
* ar – Specifies the aspect ratio (width to height) of the composite node to use when determining child constraints. A value of 0 means leave the width and height constraints as is. The algorithm may temporarily shrink one dimension to achieve that ratio
* lnSpPar – specifies the line spacing of the parent
* lnSpAfParP – specifies the line spacing after the parent paragraph
* LnSpCh specifies the line spacing of a child
* lnSpAfChP – specifies the line spacing after the child paragraph
* rtShortDist – Specifies the routing to use the shortest distance for connectors.
* alignTx – specifies if to hold text or not
* pyraLvlNode – If pyramid has a composite child node, specifies the name of the node that is a child of the composite that makes up the pyramid itself. If the node specifies a trapezoid shape, it modifies the adjustment handles to construct a pyramid.
* pyraAcctBkgdNode – If pyramid has a composite child node, specifies the child node that should hold the child text.
* pyraAcctTxNode – If pyramid has a composite child node, specifies the name of the node that is a child of the composite that makes up the child flyout shape.
* srcNode – Specifies the name of the layout node from which to start the connection.
* dstNode – Specifies the name of the layout node from which to end the connection from.
* begPts – Specifies the point type for the beginning of a connector.
* endPts – Specifies the point type for the end of a connector.
* unknown – an unkown parameter id

##### Function Type

The simple type ST\_FunctionType defines different types of conditional expressions that can be utilized. The different types of expressions are:

* cnt – Specifies the count of items.
* pos
* revPos
* posEven
* posOdd
* var
* depth
* maxDepth

##### Function Operator

The simple type ST\_FunctionOperator defines the different condition expression operators that can be used. The different operators are as follows:

* equ – equal
* neq – not equal
* gt – greater than
* lt – less than
* gte – greater than or equal to
* lte – less than or equal to

##### Horizontal Alignment

The simple type, ST\_HorizontalAlignment, specifies the different options available for alignment horizontally. The options are:

* l – left
* ctr – center
* r – right
* none – none

##### Vertical Alignment

The simple type, ST\_VerticalAlignment, specifies the different options available for alignment vertically. The options are:

* t – top
* mid – middle
* b – bottom
* none – none

##### Child Direction

The simple type ST\_ChildDirection is used to specify the direction the children are laid out. The different options are:

* horz – horizontally
* vert – vertically

##### Child Alignment

The simple type ST\_ChildAlignment defines the alignment parameter types for children. The different types are:

* t – top
* b – bottom
* l – left
* r – right

##### Secondary Child Alignment

The simple type ST\_SecondaryChildAlignment defines secondary alignment parameter types for children. The simple type ST\_ChildAlignment is mirrored here with the addition of the none type.

##### Linear Direction

The simple type ST\_LinearDirection defines the linear direction parameter types. The types are as follows:

* fromL – from left
* fromR – from right
* fromT – from top
* fromB – from bottom

##### Secondary Linear Direction

The simple type ST\_SecondaryLinearDirection defines a secondary linear direction parameter. This simple type mirrors exactly the simple type ST\_LinearDirection with the addition of the none type.

##### Starting Element

The simple type ST\_StartingElement specifies the first node point type for a cycle diagram. The different starting elements are:

* node – node
* trans – transition

##### Rotation Path

The simple type ST\_RotationPath specifies the way in which the algorithm rotates children. The different rotation types are:

* none – no rotation is performed
* alongPath – the children are rotated perpendicular to the line from the cycle’s center to the child node

##### Center Shape Mapping

The simple type ST\_CenterShapeMapping specifies how the first node of a cycle diagram is laid out within the diagram. The different places to put the first node are:

* none – the node is laid out around the circle
* fNode – the node is placed in the center of the circle and the remaining nodes along the outside of the circle

##### Bend Point

The simple type ST\_BendPoint specifies where the bend point is to be located along elbow connectors. The different options are:

* beg – beginning
* def – default
* end – end

##### Connector Routing

The simple type ST\_ConnectorRouting defines how the routing of a connector happens within the diagram. The different routing options are:

* stra – straight
* bend – an elbow connection
* curve – a curved connection
* longCurve – a curved connection with a larger radius than simple curve

##### Arrowhead Style

The simple type ST\_ArrowheadStyle defines the style of the arrowhead used on a connector. The different options are:

* auto – automatic
* arr – an arrowhead is used
* noArr – no arrowhead is used

##### Connector Dimension

The simple type ST\_ConnectorDimension defines the dimension of a connector used in a diagram. The different dimension types are:

* 1D – a single dimension connector, for example, a line
* 2D – a two dimensional connector, for example, an arrow
* cust – custom

##### Connector Point

The simple type ST\_ConnectorPoint defines the point at which the connector starts and ends. The different beginning and ending types are:

* auto – automatic
* bCtr – bottom center
* ctr – center
* midL – middle left
* midR – middle right
* tCtr – top center
* bL – bottom left
* bR – bottom right
* tL – top left
* tR – top right
* radial – radial

##### Node Horizontal Alignment

The simple type ST\_NodeHorizontalAlignment defines the alignment of a node in the horizontal direction. The different alignments are:

* l – left
* ctr – center
* r – right

##### Node Vertical Alignment

The simple type ST\_NodeVerticalAlignment defines the alignment of a node in the vertical direction. The different alignments are:

* t – top
* mid – mid
* b – bottom

##### Fallback Dimension

The simple type ST\_FallbackDimension defines how many dimensions fallback will resize the diagram in. The different options for fallback dimension are:

* 1D – fallback occurs in a single dimension (X or Y)
* 2D – fallback occurs in two dimensions (X and Y)

##### Text Direction

The simple type ST\_TextDirection specifies where the text on the first node starts. The different text directions are:

* fromT – from top
* fromB – from bottom

##### Pyramid Accent Position

The simple type ST\_PyramidAccentPosition defines where the position of the fly-out grandchildren. The possible positions are:

* bef – before
* after – after

##### Pyramid Text Margin

The simple type ST\_PyramidAccentTextMargin specifies the alignment of the text in the fly-out grandchildren. The different alignments are:

* step – the text is against the edge of the pyramid
* stack – the text aligns

##### Text Block Direction

The simple type ST\_TextBlockDirection defines the text block direction. The different directions the text can have are:

* vert – vertical
* horz – horizontal

##### Text Anchor Horizontal

The simple type ST\_AnchorHorizontal is responsible for anchoring text horizontally. The available options are:

* none – no anchor set
* ctr – the text is anchored the center

##### Text Anchor Vertical

The simple type ST\_AnchorVertical is responsible for anchoring the text vertically. The available options are:

* t – top anchor
* mid – middle anchor
* b – bottom anchor

##### Text Alignment

The simple type ST\_TextAlignment defines the text alignment. The available options are:

* l – left
* ctr – ctr
* r – right

##### Auto Text Rotation

The simple type ST\_AutoTextRotation defines the behavior of the text as the containing shape is rotated. The following options are available:

* none – no rotation, the text rotates with the shape
* upr – upright
* grav – gravity

##### Grow Direction

The simple type ST\_GrowDirection defines the growing behavior of the snake algorithm. The following options are available:

* tL – grow from the top left
* tR – grow from the top right
* bL – grow from the bottom left
* gR – grow from the bottom right

##### Flow Direction

The simple type ST\_FlowDirection Specifies whether nodes are arranged in rows or columns for the snake algorithm. The following options are available:

* row – Row
* col – column

##### Continue Direction

The simple type ST\_ContinueDirection specifies the direction of the subsequent row or column in the snake algorithm. The following options are available:

* revDir – reverse direction
* sameDir – same direction

##### Breakpoint

The simple type ST\_Breakpoint defines the behavior of a snake diagram’s breaking behavior. The available options are:

* endCnv – end of canvas
* bal – balanced
* fixed – fixed

##### Offset

The simple type ST\_Offset defines the behavior of whether each row or column in the snake algorithm is offset from the previous row or column. The available options are:

* ctr – the offset is center based
* off – there is an offset defined

##### Hierarchy Alignment

The simple type ST\_HierarchyAlignment specifies the relationship between parent and children in a hierarchy diagram. The following options exist:

* tL – top left
* tR – top right
* tCtrCh – top center children
* tCtrDes – top center descendants
* bL – bottom left
* bR – bottom right
* bCtrCh – bottom center children
* bCtrDes – bottom center descendants
* lT – left top
* lB – left bottom
* lCtrCh – left center children
* lCtrDes – left center descendants
* rT – right top
* rB – right bottom
* rCtrCh – right center children
* rCtrDes – right center descendants

##### Variable Type

The simple type ST\_VariableType defines the type of the conditional expression. These variables turn user interface options on and off. The available variable types are:

* unknown – unknown
* orgChart – organizational chart command
* chMax – used for the insert shape dropdown commands
* chPref – used for the insert shape button
* bulEnabled – used for the insert bullet command
* dir – diagram direction, RTL or LTR
* hierBranch – stores the different layouts for org chart
* animOne – exposes options for animation
* animLvl – exposes options for animation

##### Output Shape Type

The simple type ST\_OutputShapeType defines special shape types which are unique to diagrams. The unique types are:

* none – none
* conn – connector

#### Diagram Definitions

Diagram definitions define the look of a diagram. They utilize almost all the aspects of the file format discussed thus far in order to create layout properties which get translated into visual diagrams.

There are a few more simple types which need to be defined before talking about the larger aspects of what it takes to create a diagram definition. Many of these simple types are provided as wrappers or lists of the above mentioned simple types.

##### Lists

There are a group of simple types which act as lists of the simple types already mentioned. They are all defined in the following general way:

<xsd:simpleType name= NAME OF TYPE >  
 <xsd:list itemType= NAME OF SIMPLE TYPE />  
</xsd:simpleType>

The list of these list types are the following simple types:

* ST\_AxisTypes – list of ST\_AxisType
* ST\_ElementTypes – list of ST\_ElementType
* ST\_Ints – list of xsd:int
* ST\_UnsignedInts – list of xsd:unsignedInt
* ST\_Booleans – list of xsd:boolean

##### Function Value

The simple type ST\_FunctionValue is a value for a condition expression. It is defined as:

<xsd:simpleType name="ST\_FunctionValue" final="restriction">  
 <xsd:union memberTypes="xsd:int xsd:boolean ST\_Direction  
 ST\_HierBranchStyle ST\_AnimOneStr ST\_AnimLvlStr" />  
</xsd:simpleType>

##### Direction

The simple type ST\_Direction defines the direction the diagram is to be laid out. The directions available are:

* norm – normal
* rev – reversed

##### Hierarchy Branch Style

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:

* l – left
* r – right
* hang – hanging
* std – standard
* init – initial

##### One by One Animation

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:

* none – nothing is displayed
* one – the term one-by-one is used
* branch – the term branch one-by-one is used to distinguish a hierarchy diagram

##### Level Animation

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:

* none – nothing
* lvl – normal depth first traversal
* ctr – allows a radial diagram to be shown with the center node first

##### Org Chart Flag

The complex type CT\_OrgChart defines that the diagram will be an organizational chart. Organizational charts contain special behavior in that assistants can now be utilized correctly. The complex type is defined as follows:

<xsd:complexType name="CT\_OrgChart">  
 <xsd:attribute name="val" type="xsd:boolean" default="false"   
 use="optional" />  
</xsd:complexType>

##### Node Count

The simple type ST\_NodeCount holds a value that is used by the complex types CT\_ChildMax and CT\_ChildPref.

##### Child Max

This complex type defines when the user interface for inserting a child shape is to become disabled for a given node, or rather the max number of children that the user interface will be enabled for. This complex type is defined as:

<xsd:complexType name="CT\_ChildMax">  
 <xsd:attribute name="val" type="ST\_NodeCount" default="-1"   
 use="optional" />  
</xsd:complexType>

##### Child Preference

This complex type defines how many children are inserted with a single action through the user interface to add a child. This is useful in hierarchy diagrams in which one would like to specify that every shape should have three children. A single click of the add shape button would add three children. The complex type is defined as follows:

<xsd:complexType name="CT\_ChildPref">  
 <xsd:attribute name="val" type="ST\_NodeCount" default="-1"   
 use="optional" />  
</xsd:complexType>

##### Bullets Enabled

This complex type defines if the user interface for inserting a bullet into a shape is enabled or disabled for a given node. The complex type is defined as follows:

<xsd:complexType name="CT\_BulletEnabled">  
 <xsd:attribute name="val" type="xsd:boolean" default="false"   
 use="optional" />  
</xsd:complexType>

##### Direction

This complex type defines the direction of the diagram, be it normal or reversed. The complex type is defined as:

<xsd:complexType name="CT\_Direction">  
 <xsd:attribute name="val" type="ST\_Direction" default="norm"   
 use="optional" />  
</xsd:complexType>

##### Hierarchy Branch Style

This complex type defines the hierarchy branch style for a diagram. The complex type is defined as:

<xsd:complexType name="CT\_HierBranchStyle">  
 <xsd:attribute name="val" type="ST\_HierBranchStyle" default="std"   
 use="optional" />  
</xsd:complexType>

##### Animate as One

This complex type defines the animate as one value for a diagram. The complex type is defined as:

<xsd:complexType name="CT\_AnimOne" >  
 <xsd:attribute name="val" type="ST\_AnimOneStr" default="one"   
 use="optional" />  
</xsd:complexType>

##### Animate by Level

This complex type defines the animate by level value for a diagram. The complex type is defined as:

<xsd:complexType name="CT\_AnimLvl">  
 <xsd:attribute name="val" type="ST\_AnimLvlStr" default="none"   
 use="optional" />  
</xsd:complexType>

##### Layout Property Set

The complex type CT\_LayoutPropertySet holds all of the layout properties for a given diagram. The layout property set is a single structure which contains most of what has been talked about thus far in a diagram definition. The layout property set is defined as:

<xsd:complexType name="CT\_LayoutVariablePropertySet">  
 <xsd:sequence>  
 <xsd:element name="orgChart" type="CT\_OrgChart"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="chMax" type="CT\_ChildMax" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="chPref" type="CT\_ChildPref"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="bulletEnabled" type="CT\_BulletEnabled"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="dir" type="CT\_Direction" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="hierBranch" type="CT\_HierBranchStyle"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="animOne" type="CT\_AnimOne" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="animLvltype="CT\_AnimLvl" minOccurs="0"   
 maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

Because all of the contents of this complex type have already been discussed, no further detail on this complex type needs to be given.

##### Iterators

The attribute group AG\_IteratorAttributes defines the attributes used by the iterators forEach, presOf, and if. The attribute group is defined as follows:

<xsd:attributeGroup name="AG\_IteratorAttributes">  
 <xsd:attribute name="axis" type="ST\_AxisTypes" use="optional"   
 default="none" />  
 <xsd:attribute name="ptType" type="ST\_ElementTypes"   
 use="optional" default="all" />  
 <xsd:attribute name="hideLastTrans" type="ST\_Booleans"   
 use="optional" default="true" />

<xsd:attribute name="st" type="ST\_Ints" use="optional" default="1" />  
 <xsd:attribute name="cnt" type="ST\_UnsignedInts" use="optional"   
 default="0" />  
 <xsd:attribute name="step" type="ST\_Ints" use="optional" default="1" />  
</xsd:attributeGroup>

##### Constraints

The attribute group AG\_ConstraintAttributes defines the attributes used to specify a constraint. The attribute group is defined as:

<xsd:attributeGroup name="AG\_ConstraintAttributes">  
 <xsd:attribute name="type" type="ST\_ConstraintType" use="required" />

<xsd:attribute name="for" type="ST\_ConstraintRelationship"   
 use="optional" default="self" />

<xsd:attribute name="forName" type="xsd:IDREF" use="optional" />

<xsd:attribute name="ptType" type="ST\_ElementType" use="optional"   
 default="all" />  
</xsd:attributeGroup>

##### Constraint References

The attribute group AG\_ConstraintRefAttributes defines the attributes used to specify a constraint reference. The attribute group is defined as:

<xsd:attributeGroup name="AG\_ConstraintRefAttributes">  
 <xsd:attribute name="refType" type="ST\_ConstraintType"   
 use="optional" default="unknown" />

<xsd:attribute name="refFor" type="ST\_ConstraintRelationship"   
 use="optional" default="self" />

<xsd:attribute name="refForName" type="xsd:IDREF"   
 use="optional" />

<xsd:attribute name="refPtType" type="ST\_ElementType"   
 use="optional" default="all" />  
</xsd:attributeGroup>

##### Constraint

The complex type CT\_Constraint define a constraint within the layout framework. A constraint acts as a limit or sets a value to a given parameter in a diagram definition, for example, it can be used to specify that all nodes of a give point type are the same size. A constraint is defined as:

<xsd:complexType name="CT\_Constraint">  
 <xsd:attributeGroup ref="AG\_ConstraintAttributes" />  
 <xsd:attributeGroup ref="AG\_ConstraintRefAttributes" />

<xsd:attribute name="op" type="ST\_BoolOperator" use="optional"   
 default="none" />

<xsd:attribute name="val" type="xsd:double" use="optional"   
 default="0" />

<xsd:attribute name="fact" type="xsd:double" use="optional"   
 default="1" />  
</xsd:complexType>

##### Constraint List

The complex type CT\_Constraints is a sequence of CT\_Constraint complex types. It is defined as:

<xsd:complexType name="CT\_Constraints">  
 <xsd:sequence>  
 <xsd:element name="constr" type="CT\_Constraint" minOccurs="0"   
 maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

##### Rule

The complex type CT\_NumericRule defines a layout framework constraint rule. Rules are run after the diagram is created in order to specify what happens when the diagram doesn’t fully fit within the bounds. This allows for specific behavior to be defined rather than using default rules for fitting the diagram. A rule is defined in the following way:

<xsd:complexType name="CT\_NumericRule" >  
 <xsd:attributeGroup ref="AG\_ConstraintAttributes" />

<xsd:attribute name="val" type="xsd:double" use="optional"   
 default="NaN" />

<xsd:attribute name="fact" type="xsd:double" use="optional"   
 default="NaN" />

<xsd:attribute name="max" type="xsd:double" use="optional"   
 default="NaN" />  
</xsd:complexType>

##### Rule List

The complex type CT\_Rules is simply a list of CT\_NumericRule complex types. It is defined in the following manner:

<xsd:complexType name="CT\_Rules">  
 <xsd:sequence>  
 <xsd:element name="rule" type="CT\_NumericRule"   
 minOccurs="0" maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

##### Presentation Of

The complex type CT\_PresentationOf defines the mapping between data and the diagram. The complex type is defined in the following manner:

<xsd:complexType name="CT\_PresentationOf">  
 <xsd:attributeGroup ref="AG\_IteratorAttributes" />  
</xsd:complexType>

##### Layout Shape

The simple type ST\_LayoutShapeType is a simple type that contains all of the shapes available which can be used within a diagram. The simple type is defined as a union of ST\_OutputShapeType and an externally defined ST\_ShapeType.

##### Index1

The simple type ST\_Index1 defines a 1-based index that is used to index values elsewhere. The simple type is defined as:

<xsd:simpleType name="ST\_Index1">  
 <xsd:restriction base="xsd:unsignedInt">  
 <xsd:minInclusive value="1" />  
 </xsd:restriction>  
</xsd:simpleType>

##### Adjust Handle

The complex type CT\_Adj specifies a shape adjust handle modification. The shapes within a diagram can be modified based on their adjust handles, for example, the radius of the corner rounding in a rounded rectangle can be adjusted using this complex type. The complex type is defined in the following manner:

<xsd:complexType name="CT\_Adj">  
 <xsd:attribute name="idx" type="ST\_Index1" use="required" />  
 <xsd:attribute name="val" type="xsd:double" use="required" />  
</xsd:complexType>

##### Adjust Handle List

The complex type CT\_AdjLst holds all of the adjust handles for a given shape. The number of adjust handles accessible varies shape by shape, but there are usually less than four for a given shape. The complex type is defined in the following way:

<xsd:complexType name="CT\_AdjLst" o:cname="CAdjList">  
 <xsd:sequence>  
 <xsd:element name="adj" type="CT\_Adj" minOccurs="0"   
 maxOccurs="unbounded" />  
 </xsd:sequence>  
</xsd:complexType>

##### Shape

The complex type CT\_Shape specifies a shape for a layout node. The shape complex type holds all of the information associated with the particular layout node and all of the adjustments or modifications that can be made to the shape. The rot attribute specifies a rotation on the shape. The blip attribute specifies an image that is used as a background fill for the shape and the blipPhldr attribute specifies whether or not the shape shows up with an image placeholder. The zOrderOff attribute specifies an offset to be used for the z-ordering of this shape, while the lkTxEntry attribute prevents text editing within the shape. A shape is defined in the following manner:

<xsd:complexType name="CT\_Shape">  
 <xsd:sequence>  
 <xsd:element name="adjLst" type="CT\_AdjLst" minOccurs="0"   
 maxOccurs="1" />  
 </xsd:sequence>

<xsd:attribute name="rot" type="xsd:double" use="optional" default="0" />

<xsd:attribute name="type" type="ST\_LayoutShapeType"   
 use="optional" default="none" />

<xsd:attribute ref="r:blip" use="optional" />

<xsd:attribute name="zOrderOff" type="xsd:int" use="optional"   
 default="0" />

<xsd:attribute name="hideGeom" type="xsd:boolean" use="optional"   
 default="false" />

<xsd:attribute name="lkTxEntry" type="xsd:boolean" use="optional"   
 default="false" />

<xsd:attribute name="blipPhldr" type="xsd:boolean" use="optional"   
 default="false" />  
</xsd:complexType>

##### Parameter

The complex type CT\_Parameter holds the information regarding an algorithm parameter. The complex type is defined as:

<xsd:complexType name="CT\_Parameter">  
 <xsd:attribute name="type" type="ST\_ParameterId" use="required" />  
 <xsd:attribute name="val" type="xsd:string" use="required" />  
</xsd:complexType>

##### 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:

<xsd:complexType name="CT\_Algorithm" >  
 <xsd:sequence>  
 <xsd:element name="param" type="CT\_Parameter" minOccurs="0"   
 maxOccurs="unbounded" />  
 </xsd:sequence>

<xsd:attribute name="type" type="ST\_AlgorithmType" use="required" />  
 <xsd:attribute name="rev" type="xsd:unsignedInt" use="optional"   
 default="0" />   
</xsd:complexType>

##### 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:

<xsd:complexType name="CT\_LayoutNode">  
 <xsd:choice minOccurs="0" maxOccurs="unbounded">  
 <xsd:element name="alg" type="CT\_Algorithm" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="shape" type="CT\_Shape" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="presOf" type="CT\_PresentationOf"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="constrLst" type="CT\_Constraints"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="ruleLst" type="CT\_Rules" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="varLst"   
 type="CT\_LayoutVariablePropertySet" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="forEach" type="CT\_ForEach" />  
 <xsd:element name="layoutNode" type="CT\_LayoutNode" />  
 <xsd:element name="choose" type="CT\_Choose" />  
 </xsd:choice>

<xsd:attribute name="name" type="xsd:ID" use="optional" />  
 <xsd:attribute name="styleLbl" type="xsd:string" use="optional" />  
 <xsd:attribute name="chOrder" type="ST\_ChildOrderType"   
 use="optional" default="b" />  
 <xsd:attribute name="moveWith" type="xsd:IDREF" use="optional" />  
</xsd:complexType>

##### For Each

The complex type CT\_ForEach defines a for each iterator. The iteration behaves as if it were a for each loop. The complex type is defined as:

<xsd:complexType name="CT\_ForEach">  
 <xsd:choice minOccurs="0" maxOccurs="unbounded">  
 <xsd:element name="alg" type="CT\_Algorithm" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="shape" type="CT\_Shape" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="presOf" type="CT\_PresentationOf"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="constrLst" type="CT\_Constraints"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="ruleLst" type="CT\_Rules" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="forEach" type="CT\_ForEach" />  
 <xsd:element name="layoutNode" type="CT\_LayoutNode" />  
 <xsd:element name="choose" type="CT\_Choose" />  
 </xsd:choice>

<xsd:attribute name="name" type="xsd:ID" use="optional" />  
 <xsd:attribute name="ref" type="xsd:IDREF" use="optional" />  
 <xsd:attributeGroup ref="AG\_IteratorAttributes" />  
</xsd:complexType>

##### When

The complex type CT\_When defines an if conditional expression. The complex type is usually used in conjunction with the else counterpart which is defined next. The CT\_When complex type is defined in the following manner:

<xsd:complexType name="CT\_When">  
 <xsd:choice minOccurs="0" maxOccurs="unbounded">  
 <xsd:element name="alg" type="CT\_Algorithm" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="shape" type="CT\_Shape" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="presOf" type="CT\_PresentationOf"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="constrLst" type="CT\_Constraints"   
 minOccurs="0" maxOccurs="1” />

<xsd:element name="ruleLst" type="CT\_Rules" minOccurs="0"   
 maxOccurs="1" o:cname="Rules" />

<xsd:element name="forEach" type="CT\_ForEach" />  
 <xsd:element name="layoutNode" type="CT\_LayoutNode" />  
 <xsd:element name="choose" type="CT\_Choose" />  
 </xsd:choice>

<xsd:attribute name="name" type="xsd:ID" use="optional" />  
 <xsd:attributeGroup ref="AG\_IteratorAttributes" />

<xsd:attribute name="func" type="ST\_FunctionType"   
 use="required" />

<xsd:attribute name="arg" type="ST\_FunctionArgument"   
 use="optional" />

<xsd:attribute name="op" type="ST\_FunctionOperator"   
 use="required” />

<xsd:attribute name="val" type="ST\_FunctionValue"   
 use="required" />  
</xsd:complexType>

##### Otherwise

The complex type CT\_Otherwise is the else counterpart to the already defined if conditional expression. The complex type is defined as:

<xsd:complexType name="CT\_Otherwise" o:cname="DDOtherwise">  
 <xsd:choice minOccurs="0" maxOccurs="unbounded">  
 <xsd:element name="alg" type="CT\_Algorithm" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="shape" type="CT\_Shape" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="presOf" type="CT\_PresentationOf"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="constrLst" type="CT\_Constraints"   
 minOccurs="0" maxOccurs="1" />

<xsd:element name="ruleLst" type="CT\_Rules" minOccurs="0"   
 maxOccurs="1" />

<xsd:element name="forEach" type="CT\_ForEach" />  
 <xsd:element name="layoutNode" type="CT\_LayoutNode" />  
 <xsd:element name="choose" type="CT\_Choose" />  
 </xsd:choice>

<xsd:attribute name="name" type="xsd:ID" use="optional" />  
</xsd:complexType>

##### Choose Statement

The complex type CT\_Choose packages together the if and else conditions into an actual if/else statement. The complex type is defined in the following manner:

<xsd:complexType name="CT\_Choose" o:cname="DDChoose">  
 <xsd:sequence>  
 <xsd:element name="if" type="CT\_When" maxOccurs="unbounded" />  
 <xsd:element name="else" type="CT\_Otherwise" minOccurs="0" />  
 </xsd:sequence>  
 <xsd:attribute name="name" type="xsd:ID" use="optional" />  
</xsd:complexType>

##### Sample Data

The complex type CT\_SampleData defines how the data model is to be populated in an initial manner. The complex type holds a temporary data model when there is no data model present in order to display a diagram on an initial insert. The complex type is defined by:

<xsd:complexType name="CT\_SampleData">  
 <xsd:sequence>  
 <xsd:element name="dataModel" type="CT\_DataModel" minOccurs="0" />  
 </xsd:sequence>

<xsd:attribute name="useDef" type="xsd:boolean" use="optional"   
 default="false" />  
</xsd:complexType>

##### Common Structures

CT\_Category, CT\_Categories, CT\_Name, CT\_Description, and ST\_Version are defined just as their counterparts in the subclauses above, and they perform the same tasks.

##### Diagram Definition

The complex type T\_DiagramDefinition is the root element for a diagram definition. It is defined in the following manner:

<xsd:complexType name="CT\_DiagramDefinition">  
 <xsd:sequence>  
 <xsd:element name="title" type="CT\_Name" minOccurs="0"   
 maxOccurs="unbounded" />

<xsd:element name="desc" type="CT\_Description"   
 minOccurs="0" maxOccurs="unbounded" />

<xsd:element name="catLst" type="CT\_Categories" minOccurs="0 />  
 <xsd:element name="sampData" type="CT\_SampleData" minOccurs="0" />  
 <xsd:element name="styleData" type="CT\_SampleData" minOccurs="0" />  
 <xsd:element name="clrData" type="CT\_SampleData" minOccurs="0" />  
 <xsd:element name="layoutNode" type="CT\_LayoutNode" />  
 </xsd:sequence>

<xsd:attribute name="uniqueId" type="xsd:anyURI" use="optional" />  
 <xsd:attribute name="minVer" type="ST\_Version" use="optional"   
 default="12.0" />  
 <xsd:attribute name="defStyle" type="xsd:anyURI" use="optional" />  
</xsd:complexType>

### Drawing Anchors

Yet to be supplied

## 3-D

### Introduction

This subclause provides a high-level overview of the content described in the following schemas: dml-sp3dstyles.xsd, dml-sp3dscene.xsd, dml-sp3dsceneplane.xsd, dml-sp3dlight.xsd, and dml-sp3dcamera.xsd.

This aspect of DrawingML deals mainly with the 3-D aspects, and can be broken down into two topics. The first topic is the actual 3-D properties associated with an object, and the second is the styling information associated with an object. The above-mentioned schemas fall into the grouping in the following way:

|  |  |
| --- | --- |
| 3-D | Styles |
| dml-sp3dscene.xsd  dml-sp3dsceneplane.xsd  dml-sp3dlight.xsd  dml-sp3dcamera.xsd | dml-sp3dstyles.xsd |

### 3-D

Here we'll explain the 3-D definitions contained in DrawingML. The goal here is to define a 3-D scene so that lighting calculations can be made on the geometry within the scene.

#### 3-D Scene

Every 3-D scene consists of a camera, a light, and a backdrop, that define the associated properties of the scene. The complex type, CT\_Scene3D, defines the scene as follows:

<xsd:complexType name="CT\_Scene3D" oxsd:cname="Scene3D">  
 <xsd:sequence>  
 <xsd:element name="camera" type="CT\_Camera" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="lightRig" type="CT\_LightRig" minOccurs="1"  
 maxOccurs="1/>

<xsd:element name="backdrop" type="CT\_Backdrop" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

As was stated above, the complex type, CT\_Scene3D, contains a camera, a set of lights (the light rig), and a backdrop. Those familiar with 3-D rendering techniques understand the usage of a camera and set of lights, or light rig. The backdrop, however, is a special structure (which will be defined below) that allows for a special plane to render certain effects that need to be rendered together in a single plane. The final element of a CT\_Scene3D is the ext element. This is a DrawingML structure used for future extensibility. This element will be seen in other complex types dealing with the 3-D scene as well.

#### Camera

The complex type, CT\_Camera, defines a camera within the 3-D scene. A camera is based on a preset, with an optional rotation, field-of-view, and zoom, which all act as overrides for the preset values. A camera is defined in the following way:

<xsd:complexType name="CT\_Camera">  
 <xsd:sequence>  
 <xsd:element name="rot" type="CT\_SphereCoords" minOccurs="0"   
 maxOccurs="1" oxsd:dataStructure="optional" />  
 </xsd:sequence>

<xsd:attribute name="prst" type="ST\_PresetCameraType" use="required"  
 />

<xsd:attribute name="fov" type="ST\_FOVAngle" use="optional" />

<xsd:attribute name="zoom" type="ST\_PositivePercentage"  
 use="optional" default="100000" />  
</xsd:complexType>

The only complex type contained in the camera, CT\_SphereCoords, is a type defined elsewhere within the DrawingML. There are three simple types associated with a camera:

* ST\_FOVAngle (field of view angle), which is a positive angle between 0 and 180 in 60,000th of a degree.
* ST\_PositivePercentage (zoom), which is defined as a percentage in 1,000th of a percent.
* ST\_PresentCameraType (preset camera)

Figure 1 below shows some different presets applied to a shape.

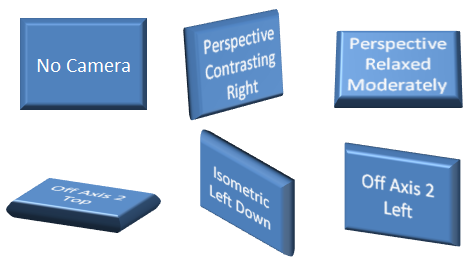


Figure 12: Different default cameras applied to a shape

The available options for ST\_PresetCameraType are as follows:

legacyObliqueTopLeft  
legacyObliqueTop  
legacyObliqueLeft  
legacyObliqueFront  
legacyObliqueRight

legacyObliqueBottomLeft  
legacyObliqueBottom  
legacyObliqueBottomRight  
legacyPerspectiveTopLeft  
legacyPerspectiveTop

legacyPerspectiveTopRight  
legacyPerspectiveLeft  
legacyPerspectiveFront  
legacyPerspectiveRight  
legacyPerspectiveBottomLeft

legacyPerspectiveBottom  
legacyPerspectiveBottomRight  
orthographicFront  
isomentricTopUp  
isometricTopDown

isomentricBottomDown  
isometricLeftUp  
isometricLeftDown  
isometricRightUp  
isometricRightDown

isometricOffAxis1Left  
isometricOffAxis1Right  
isometricOffAxis1Top  
isometricOffAxis2Left  
isometricOffAxis2Right

isometricOffAxis2Top  
isometricOffAxis3Left  
isometricOffAxis3Right  
isometricOffAxis3Bottom  
isometricOffAxis4Left

isometricOffAxis4Right  
isometricOffAxis4Bottom  
obliqueTopLeft  
obliqueTopRight  
obliqueLeft

obliqueRight  
obliqueBottomLeft  
obliqueBottom  
obliqueBottomRight  
perspectiveFront

perspectiveLeft  
perspectiveRight  
perspectiveAbove  
perspectiveBelow  
perspectiveAboveLeftFacing

perspectiveAboveRightFacing  
perspectiveContrastingRightFacing  
perspectiveContrastingLeftFacing  
perspectiveHeroicLeftFacing  
perspectiveHeroicRightFacing

perspectiveHeroicExtremeLeftFacing  
perspectiveHeroicExtremeRightFacing  
perspectiveRelaxed  
perspectiveRelaxedModerately

#### Light

The complex type, CT\_LightRig, defines the lighting of the scene. A light rig consists of a preset direction, preset rig type, and a rotation that serves as an override for the direction. The complex type is defined as:

<xsd:complexType name="CT\_LightRig">  
 <xsd:sequence>  
 <xsd:element name="rot" type="CT\_SphereCoords" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
 <xsd:attribute name="rig" type="ST\_LightRigType" use="required" />  
 <xsd:attribute name="dir" type="ST\_LightRigDirection" use="required"}  
 />  
</xsd:complexType>

Just as with the camera, the complex type, CT\_SphereCoords, is defined elsewhere in the DrawingML. This element, however, serves as an override for the default light right direction. Figure 2 below shows some of the different preset lights applied to a shape.



Figure 13: Some preset lights applied to a shape.

The types of available light rigs are:

legacyFlat1  
legacyFlat2  
legacyFlat3  
legacyFlat4  
legacyNormal1

legacyNormal2  
legacyNormal3  
legacyNormal4  
legacyHarsh1  
legacyHarsh2

legacyHarsh3  
legacyHarsh4  
threePoint  
balanced  
soft

harsh  
flood  
contrasting  
morning  
sunrise

sunset  
chilly  
freezing  
flat  
twoPoint  
glow  
brightRoom

The types of available present directions are:

tl – top left  
t – top   
tr – top right  
l – left

r – right   
bl – bottom left  
b – bottom   
br – bottom right

#### Backdrop

The complex type, CT\_Backdrop, defines a unique place in the 3-D scene. The backdrop is a flat 2-D plane that can hold effects, such as shadows, oriented in 3-D space. The points and vectors contained within the backdrop are relative to world space. The complex type is defined as:

<xsd:complexType name="CT\_Backdrop">  
 <xsd:sequence>  
 <xsd:element name="anchor" type="CT\_Point3D" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="norm" type="CT\_Vector3D" minOccurs="1"  
 maxOccurs="1" />

<xsd:element name="up" type="CT\_Vector3D" minOccurs="1"  
 maxOccurs="1"/>

<xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>  
</xsd:complexType>

All of the complex types defined within this backdrop are defined elsewhere in DrawingML. As with other complex types, the backdrop also contains an element reserved for future extensibility.

### Styles

The 3-D styles section encompasses the properties for 3-D shapes. These properties are those that get applied 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 simple types used within the complex types of this group are defined below.

#### Simple Types

The simple types defined here outline the different presets available to the user. These presets are applied to the shapes through the complex type definitions that are outlined later.

##### Bevel Type

The simple type, ST\_BevelPresetType, defines a preset bevel for a shape and some examples can be seen in figure 3.

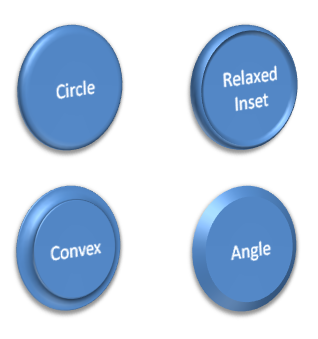


Figure 14: Different bevel types applied to a shape

The different types of bevels available are:

relaxedInset  
circle  
slope  
cross  
angle  
softRound

convex  
coolSlant  
divot  
riblet  
hardedge  
artDeco

##### Preset Material Type

The simple type, ST\_PresetMaterialType, defines a material for the shape. The material properties describe the surface appearance of the shape, and are used in lighting calculations to define exactly how the light interacts with the shape. Some example material types can be seen in figure 4.



Figure 15: Different material types on a shape.

The different preset material types are:

legacyMatte  
legacyPlastic  
legacyMetal  
legacyWireframe  
matte

plastic  
metal  
warmMatte  
translucentPowder  
powder

dkEdge  
softEdge  
clear  
flat  
softMetal

#### 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.

##### 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:

<xsd:complexType name="CT\_Bevel">  
 <xsd:attribute name="w" type="ST\_PositiveCoordinate" use="optional"  
 default="76200" />

<xsd:attribute name="h" type="ST\_PositiveCoordinate" use="optional"  
 default="76200" />

<xsd:attribute name="prst" type="ST\_BevelPresetType" use="optional"  
 default="circle" />  
</xsd:complexType>

##### 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:

<xsd:complexType name="CT\_Shape3D">  
 <xsd:sequence>  
 <xsd:element name="bevelT" type="CT\_Bevel" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="bevelB" type="CT\_Bevel" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="extrusionClr" type="CT\_Color" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="contourClr" type="CT\_Color" minOccurs="0"  
 maxOccurs="1" />

<xsd:element name="ext" type="CT\_OfficeArtExtension" minOccurs="0"  
 maxOccurs="1" />  
 </xsd:sequence>

<xsd:attribute name="z" type="ST\_Coordinate" use="optional"   
 default="0" />

<xsd:attribute name="extrusionH" type="ST\_PositiveCoordinate"   
 use="optional" default="0" />

<xsd:attribute name="contourW" type="ST\_PositiveCoordinate"   
 use="optional" default="0" />

<xsd:attribute name="prstMaterial" type="ST\_PresetMaterialType"   
 use="optional" default="warmMatte" />  
</xsd:complexType>

##### Flat Text

The complex type, CT\_FlatText, defines a text object in a 3-D scene that should be rendered as a normal, flat, text overlay outside of the 3-D scene. The complex type is defined in the following manner:

<xsd:complexType name="CT\_FlatText">  
 <xsd:attribute name="z" type="ST\_Coordinate" use="optional"   
 default="0" />  
</xsd:complexType>

##### Group, Text 3-D

The final structure to be defined is a group, EG\_Text3D, which describes how text should be applied in the 3-D scene. If the text object is a member of the 3-D scene, then there are three different ways it can be displayed:

* If no EG\_Text3D choice is provided, the text will be rendered in a scene coherent manner and will be rendered in perspective inside of the 3D scene as a planer shape inside the 3-D.
* If CT\_Shape3D is provided then the text will be scene coherent and fully 3-D.
* If CT\_FlatText is provided then the text will be drawn as normal 2-D text rendered on top of the 3-D scene.

An EG\_Text3D is defined in the following manner:

<xsd:group name="EG\_Text3D">  
 <xsd:choice oxsd:cname="Text3DChoice"   
 oxsd:cnameMember="text3DChoice">  
 <xsd:element name="sp3dtype="CT\_Shape3D" minOccurs="1"   
 maxOccurs="1/>  
 <xsd:element name="flatTx" type="CT\_FlatText" minOccurs="1"   
 maxOccurs="1" />  
 </xsd:choice>  
</xsd:group>

## Tables

Yet to be supplied

## Shape Properties and Effects

Yet to be supplied

## Chart

Yet to be supplied

## Pictures

### Introduction

This subclause provides a high-level overview of the content described in the dml-picture.xsd schema.

The DrawingML Picture file format is broken down into the following subjects:

* Specifying a basic picture
* Attaching properties to this picture
* Transforming this picture
* The best way to understand the above subjects will be to cover them in the ordering above.

### Specifying a Basic Picture

A picture can be inserted into a presentation slide by use of the picture element, pic, which is similar to the shape element but contains some key differences that enable more complete storage of picture information. This basic picture element should contain a blipfill and some basic non-visual picture properties.

