Office Open XML

Document Interchange Specification

Ecma TC45

Working Draft 1.4

Part 5: Markup Compatibility

Public Distribution

August 2006

Table of Contents

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 Part is Part 5 of a multi-part Standard covering Open XML-related technology.

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

1. Scope

This Part describes a set of conventions that facilitate future enhancement and extension of XML markup.

Any XML-based document specification can use the markup language described in this Part as the basis of its compatibility with previous and future specification revisions, and to enable the creation of independent extensions of its specification.

2. Conformance

Open XML Markup Compatibility conformance is of interest to the following audiences:

- Those designing, implementing, or maintaining a producer or consumer for an XML-based format that relies on the Open XML Markup Compatibility model to enable versioning or extensibility.
- Governmental or commercial entities wishing to procure a producer or consumer of an XML-based format that relies on the Open XML Markup Compatibility model to enable versioning or extensibility.
- Testing organizations wishing to provide a conformance test suite for an XML-based format that relies on the Open XML Markup Compatibility model to enable versioning or extensibility.
- Programmers wishing to interact programmatically with an XML-based format that relies on the Open XML Markup Compatibility model to enable versioning or extensibility.
- Educators wishing to teach about the Open XML Markup Compatibility model or an XML-based format that uses it.
- Authors wanting to write about the Open XML Markup Compatibility model or an XML-based format that uses it.

As such, conformance is most important, and the bulk of this Part specifies the characteristics that make Open XML Markup Compatibility consumers or producers strictly conforming ones.

The text in this Part is divided into *normative* and *informative* categories. Normative text is further broken into *mandatory* and *optional* subcategories. A mandatory feature shall be implemented as specified by this Part. An optional feature need not be implemented; however, if it is supported, it shall be implemented as specified by this Part. Text marked as informative is for information purposes only. Unless stated otherwise, all features are mandatory. The text in this Part that specifies requirements is considered normative. All other text in this specification is informative; that is, for information purposes only.

To conform to this Part, an implementation shall provide the specified normative elements and meet the criteria of §2.1.

This Part does not contain any unspecified behavior.

2.1 A Conforming Implementation

A *strictly conforming consumer or producer* shall use only those features of Open XML Markup Compatibility specified in this Part as being mandatory. It shall not act in a manner that is dependent on any unspecified or implementation-defined behavior. A strictly conforming consumer shall accept any valid XML document based upon the Open XML Markup Compatibility Part. XML documents containing Open XML Markup Compatibility elements and attributes generated by a strictly conforming producer shall be valid.

A strictly conforming consumer or producer shall interpret characters in conformance with ISO/IEC 10646-1 as required by the XML 1.0 Standard. A strictly conforming consumer or producer shall accept Unicode source files encoded with either the UTF-8 or UTF-16 encoding forms as required by the XML 1.0 Standard.

A strictly conforming consumer shall produce at least one diagnostic message if its input XML document is invalid. An XML document is invalid if any of its contents violate any rule of syntax or any negative requirement in this Part.

A (non-strictly) *conforming consumer or producer* is one having capabilities that are a superset of those described in this Part, provided these capabilities do not alter the behavior that is required by a strictly conforming consumer or producer.

A conforming consumer or producer shall be accompanied by a document that defines all implementation-defined characteristics and all extensions to this Part.

2.2 Verbal Forms for the Expression of Provisions

Specific verbal forms are used in the normative clauses of this Part in order to distinguish among requirements for compliance, provisions allowing a freedom of choice, and recommendations. Those verbal forms are prescribed by ISO/IEC Directives, Part 2, “Rules for the structure and drafting of International Standards.”

The following table summarizes the prescribed verbal forms and equivalent expressions used in this Part.

Table 2–1. Verbal forms

Provision	Verbal form	Alternative expression
A requirement on a producer or consumer, strictly to be followed for compliance to this Part	Shall shall not	is required to is, is, is to is not permitted is not allowed
A permission expressed by the Part	might need not	is permitted to is allowed is not required to
A recommendation expressed by the Part, it need not be followed	should should not	it is recommended that is recommended
A capability or possibility open to a producer or a consumer of the Part	Can cannot	is able to it is possible to is possible

3. Normative References

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this Part. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Part are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 2382.1:1993, *Information technology — Vocabulary — Part 1: Fundamental terms*.

ISO/IEC 10646:2003 (all parts), *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*.

4. Definitions

For the purposes of this Part, the following definitions apply. Other terms are defined where they appear in *italics* type. Terms explicitly defined in this Part are not to be presumed to refer implicitly to similar terms defined elsewhere. [*Note*: This part uses OPC-related terms, which are defined in Part 2: "Open Packaging Conventions". *end note*]

Throughout this Part, the terms *namespace declaration*, *namespace name*, *qualified name*, *expanded name*, *prefixed name*, *unprefixed name*, and *local name* shall have the meanings as defined in the W3C Recommendation, “Namespaces in XML 1.0 (Second Edition).”

alternate content — A set of markup alternatives, of which no more than one shall be processed by a markup consumer. A markup consumer chooses from among the alternatives based upon its set of understood namespaces.

compatibility-rule attribute — An XML attribute described in this Part that expresses rules governing markup consumers’ behavior when encountering XML elements and attributes from non-understood namespaces.

ignore — To disregard the presence of an element or attribute, processing the markup as if that element or attribute did not exist.

markup consumer — A tool that can read and parse a markup document and further conforms to the requirements of a markup specification.

markup document — An XML document that conforms to the requirements of a markup specification.

markup editor — A tool that acts as a markup consumer in reading a markup document, makes changes to that markup, and acts as the producer of the modified markup.

markup preprocessor — A software module, designed for use in the implementation of markup consumers, that follows the rules of this specification to remove or replace all elements and attributes from the Markup Compatibility namespace, all elements and attributes from ignorable non-understood namespaces, and all elements and attributes from subsumed namespaces..

markup producer — A tool that can generate a markup document, and conforms to a markup specification.

markup specification — An XML-based markup format specification that incorporates all of the requirement of this Part.

namespace, ignorable — A namespace, identified in markup, whose elements and attributes shall be ignored by a markup consumer that does not understand that namespace.

namespace, understood — An XML namespace containing any recognized XML elements or attributes.

preserve —To retain an ignored element or attribute during the course of editing.

recognize — To have knowledge of the correct interpretation of an XML element, XML attribute, or attribute-value, as defined in a markup specification.

5. Notational Conventions

Except where otherwise noted, syntax descriptions are expressed in the ABNF format as defined by RFC 4234.

Glossary terms are formatted like *this*.

Syntax descriptions and code are formatted in monospace type.

Attributes names are formatted in arial bold type.

Attribute values are formatted in `courier new` type.

Elements are formatted in arial type.

Examples are delimited by [*Example*: … *end example*] Examples are informative.

Notes are delimited by [*Note*: … *end note*] Notes are informative.

6. Acronyms and Abbreviations

This clause is informative

The following acronyms and abbreviations are used throughout this Part

IEC — the International Electrotechnical Commission

ISO — the International Organization for Standardization

W3C — World Wide Web Consortium

End of informative text

7. General Description

This Part is intended for use by implementers, academics, and application programmers. As such, it contains a considerable amount of explanatory material that, strictly speaking, is not necessary in a formal language specification.

This Part is divided into the following subdivisions:

1. Front matter (clauses 1–7);
2. Overview and introductory material (clause 8);
3. Main body (clauses 9–13);
4. Annexes

Examples are provided to illustrate possible forms of the constructions described. References are used to refer to related clauses. Notes are provided to give advice or guidance to implementers or programmers.

The following clauses form the normative pieces of this Part:

- Clauses -1817705789–5, 7, and 9–12

The following clauses form the informative pieces of this Part:

- Introduction
- Clause 6, 8, and 13
- All annexes
- All notes and examples

Whole clauses that are informative are identified as such. Informative text that is contained within normative text is identified as either an example or a note, as specified in §5.

8. Overview

This clause is informative

This Part describes a set of XML elements and attributes that collectively enable markup producers to explicitly guide markup consumers in their handling of any XML elements and attributes whose expanded names refer to namespaces not understood by the markup consumer.

End of informative text

9. Markup Compatibility Fundamentals

9.1 Terminology

Any XML-based document specification can use the markup language described in this Part as the basis of its compatibility with previous and future specification revisions and to enable the creation of independent extensions of its specification. In this Part, the term *markup specification* is used to refer to a specification that relies on this Open XML Markup Compatibility Part and defines a set of XML namespaces, the elements and attributes within those namespaces, and any processing requirements for those namespaces, elements, and attributes. *Markup document* refers to an XML document that conforms to a markup specification. A *markup producer* is a software application or component that generates a markup document. A *markup consumer* is a software application or component that can process a markup document according to the processing requirements of the markup specification.

This Part is dependent on XML namespace names, expressed as URIs. A markup specification defines a set of elements and attributes within one or more namespaces. A characteristic of a markup consumer is that it can *recognize* or process the elements and attributes within *understood namespaces*, including those containing elements and attributes defined in the markup specification. Markup consumers shall process all recognized elements and attributes of any understood namespace according to the requirements of the markup specifications defining those elements or attributes. A markup specification might require that the presence of unrecognized elements or attributes in an understood namespace be treated as an error condition. Markup consumers shall always treat the presence of an unrecognized element or attribute from the Markup Compatibility namespace as an error condition. If a markup consumer encounters an element or attribute from a non-understood namespace, the markup consumer shall treat the presence of that element or attribute as an error condition, unless the markup producer has embedded in the markup document explicit Markup Compatibility elements or attributes that override that behavior.

Within a markup document, a markup producer might use Markup Compatibility attributes to identify *ignorable namespaces*. Markup consumers shall *ignore* elements and attributes from namespaces that are both non-understood and ignorable, and shall not treat their presence as errors. A markup producer can indicate to the markup consumer whether the content of an ignored element shall be disregarded together with the ignored element, or if the content should be processed as if it were the content of the ignored element's parent.

Within a markup document, a markup producer might also use Markup Compatibility attributes to suggest to a markup editor that the editor attempt to *preserve* some ignored elements or attributes. The markup editor can attempt to persist these ignored elements and attributes when a saving markup document, despite the editor's inability to recognize the purpose of these ignored elements and attributes.

A markup producer, aware of the existence of markup consumers with overlapping but different sets of understood namespaces, might choose to include in a markup document *alternate content* regions, each

holding a set of markup alternatives for use by different markup consumers. A markup consumer shall use rules embedded in the markup document by the markup producer to select no more than one of these alternatives for normal processing, and shall disregard all other alternatives.

Future versions of markup specifications shall specify new namespaces for any markup that is enhanced or modified by the new version, which a markup consumer of that version of the markup specification would include as an understood namespace. Some of the namespaces introduced in the new markup specification might each subsume one of the previous version's understood namespaces. A new understood namespace *subsumes* a previously-understood namespace if it includes all of the elements, attributes, and attribute values of the previously-understood namespace. Regardless of whether a new namespace subsumes a previously defined namespace, markup consumers based on a new version of a markup specification shall support all understood namespaces of the previous version unless the new version makes an explicit statement to the contrary.

This Part can be implemented using a pipelined, preprocessing architecture in the form of a software module called a *markup preprocessor*. A markup preprocessor can use the Markup Compatibility elements and attributes to produce output that is free of all ignorable non-understood content, all Markup Compatibility elements and attributes, and all elements and attributes in subsumed namespaces.

9.2 Markup Compatibility Namespace

The following is the Markup Compatibility namespace:

```
http://schemas.openxmlformats.org/markup-compatibility/2006
```

The namespace includes XML elements and attributes that markup producers can use to express to markup consumers how they shall respond to markup in non-understood namespaces. The elements and attributes described in this Part are contained in the Markup Compatibility namespace.

10. Markup Compatibility Attributes and Elements

This Part defines attributes to express compatibility rules and elements to specify alternate content.

[*Note*: Whitespace characters that can appear in attribute values, as defined in the XML specification, are described in the following table:

Table 10–1. Whitespace characters in attribute values

Character	Syntax
space	#x20
tab	#x9
line feed	#xA
carriage return	#xD

end note]

Whitespace characters that appear in attribute values shall be normalized by markup consumers before processing as follows:

1. Replace each tab, line feed, and carriage return with a space.
2. Collapse contiguous sequences of spaces into a single space.
3. Remove leading and trailing spaces.

[*Note*: The following table, and Table 10–3, summarize the Markup Compatibility attributes and elements, respectively, and are further described in the sub-clauses that follow.

Table 10–2. Compatibility-rule attributes

Name	Description
Ignorable	A whitespace-delimited list of namespace prefixes that identify a set of namespaces whose elements and attributes should be silently ignored by markup consumers that do not understand the namespace of the element or attribute in question.
ProcessContent	A whitespace-delimited list of element-qualified names identifying the expanded names of elements whose content shall be processed, even if the elements themselves are ignored. In any qualified name in the list, the wildcard character “*” can replace the local name to indicate that the content of all elements in the namespace shall be processed.

Name	Description
PreserveElements	A whitespace-delimited list of element qualified names identifying the expanded names of elements that should be preserved when the markup is edited, even if the elements themselves are ignored. In any qualified name in the list, the wildcard character "*" can replace the local name to indicate that all elements in the namespace should be preserved.
PreserveAttributes	A whitespace-delimited list of attribute qualified names identifying the expanded names of attributes that should be preserved when the markup is edited. In any qualified name in the list, the wildcard character "*" can replace the local name to indicate that all attributes in the namespace should be preserved.
MustUnderstand	A whitespace-delimited list of namespace prefixes identifying a set of namespace names. Markup consumers that do not understand these namespaces shall not continue to process the markup document and shall generate an error.

Table 10–3. Alternate-content elements

Name	Description
AlternateContent	Associates a set of possible markup alternatives that a markup consumer might choose based on that markup consumer's understood namespaces. The markup consumer chooses the first alternative, in markup order, requiring only namespaces it understands.
Choice	This child of AlternateContent contains a single markup alternative and identifies the namespaces that the markup consumer needs to understand in order to choose and process that alternative. At least one Choice element is required.
Fallback	This child of AlternateContent specifies the fallback markup alternative a markup consumer chooses if the markup consumer cannot choose any Choice alternative. An AlternateContent element shall hold no more than one Fallback element, which if present, shall follow all Choice elements.

end note]

10.1 Compatibility-Rule Attributes

This Part describes the manner by which compatibility rules can be associated with any XML element, including Markup Compatibility elements. Compatibility rules are associated with an element by means of compatibility-rule attributes. These attributes control how markup consumers, including markup editors, shall react to elements or attributes from non-understood namespaces.

The principal compatibility-rule attribute is the Ignorable attribute. By default, markup consumers should treat the presence of any element or attribute from a non-understood namespace as an error condition. However, elements and attributes from a non-understood namespace identified in an Ignorable attribute shall be ignored without error.

Compatibility-rule attributes shall affect the element to which they are attached, including the element's other attributes and contents. The order in which compatibility-rule attributes occur on an element shall not affect the application of those rules to that element, its attributes, or its contents.

10.1.1 Ignorable Attribute

The Ignorable attribute is declared as a whitespace-delimited list of namespace prefixes, where each namespace prefix identifies an ignorable namespace. During processing, if a markup consumer encounters an element or attribute in a non-understood and ignorable namespace, the markup consumer shall treat that element or attribute as if it did not exist and shall not generate an error.

Markup consumers should treat elements and attributes from non-ignorable and non-understood namespaces as errors.

[*Note*: By default, an ignored element is ignored in its entirety, including its attributes and its content. The processing of an ignored element's contents is enabled through the use of the ProcessContents attribute. The PreserveAttributes and PreserveElements attributes can be used to assist markup editors in preserving ignored elements and ignored attributes. *end note*]

If an Ignorable attribute references an understood namespace, its presence shall not affect the processing of elements and attributes from the understood namespace, regardless of whether or not those elements and attributes are recognized by the markup consumer.

The presence of an Ignorable attribute shall reset a markup consumer's content-processing and preservation behavior for all elements and attributes in the namespaces referenced by the Ignorable attribute value. Once reset, by default the markup consumer shall ignore all content contained by the ignored element and markup editors shall not preserve ignored attributes and elements. This default behavior shall be overridden by the presence of any ProcessContent, PreserveAttributes, and PreserveElements attributes on the element carrying the Ignorable attribute.

The value of the Ignorable attribute can be an empty or blank string. When a markup consumer encounters such a value, it shall proceed as if the Ignorable attribute was not present.

[*Example*: Example 10–1. Processing Ignorable and PreserveAttributes attributes

The example namespace with the name `http://schemas.openxmlformats.org/Circles/v1` defines a Version 1 element, Circle, in its initial version. The subsequent Version 2 of the markup specification introduces the Opacity attribute in a new Version 2 namespace. The subsequent Version 3 of the markup specification introduces the Luminance attribute in a Version 3 namespace. The markup is loadable by markup consumers conforming to any one of these markup specification versions. The PreserveAttributes attribute specifies that the v3:Luminance attribute should be preserved during editing, even when the markup editor does not understand the v3:Luminance attribute.

For a Version 1 markup consumer, Opacity and Luminance are ignored attributes.

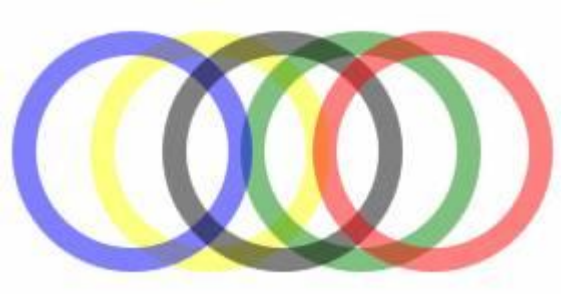
For a Version 2 markup consumer, only Luminance is an ignored attribute.

For a Version 3 markup consumer and beyond, none of the attributes are ignored.

```
<Circles xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
  compatibility/2006"
  xmlns:v2="http://schemas.openxmlformats.org/Circles/v2"
  xmlns:v3="http://schemas.openxmlformats.org/Circles/v3"
  mc:Ignorable="v2 v3"
  mc:PreserveAttributes="v3:Luminance">
  <Circle Center="0,0" Radius="20" Color="Blue"
    v2:Opacity="0.5" v3:Luminance="13" />
  <Circle Center="25,0" Radius="20" Color="Black"
    v2:Opacity="0.5" v3:Luminance="13" />
  <Circle Center="50,0" Radius="20" Color="Red"
    v2:Opacity="0.5" v3:Luminance="13" />
  <Circle Center="13,0" Radius="20" Color="Yellow"
    v2:Opacity="0.5" v3:Luminance="13" />
  <Circle Center="38,0" Radius="20" Color="Green"
    v2:Opacity="0.5" v3:Luminance="13" />
</Circles>
```

The following figure shows an example possible rendering of the markup above.

Figure 10–1. Rings



end example]

[*Example*: Example 10–2. Processing Ignorable content using namespaces

A markup consumer that does not understand the namespace with the name `http://schemas.openxmlformats.org/MyExtension/v1` shall ignore both the a:IgnoreMe and b:IgnoreMeToo elements. Although the two elements use different namespace prefixes, they draw from the same ignorable namespace.

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
  compatibility/2006"
  xmlns:a="http://schemas.openxmlformats.org/MyExtension/v1"
  xmlns:b="http://schemas.openxmlformats.org/MyExtension/v1"
  mc:Ignorable="a">
  <a:IgnoreMe />
  <b:IgnoreMeToo />
</Circles>
```

end example]

10.1.2 ProcessContent Attribute

The ProcessContent attribute is declared as a whitespace-delimited list of element-qualified names identifying the expanded names of elements whose content shall be processed, even if the elements themselves are ignored. In any qualified name in the list, the wildcard character "*" can replace the local name to indicate that the content all elements in the namespace shall be processed.

A markup consumer that encounters an ignored element whose expanded name matches the expanded name of an element declared in the ProcessContent attribute value, the markup consumer shall consider that element to be a processed element, regardless of whether or not the element's qualified name matches the qualified name declared in the ProcessContent attribute value.

A markup consumer that encounters a processed element shall process the contents of that element as if the contents were directly embedded within the parent element of the ignored element.

The ProcessContent attribute value shall not reference any element name that does not belong to a namespace that is identified by the Ignorable attribute declared on the same element.

The value of the ProcessContent attribute can be an empty or blank string. When a markup consumer encounters such a value, it shall proceed as if the ProcessContent attribute was not declared.

A markup producer shall not generate an element that also declares an xml:lang or xml:space attribute.

[*Example*: Example 10–3. Processing Ignorable and ProcessContent attributes

A Version 1 markup consumer ignores the blue, black, and red circles, but does render the yellow and green circles.

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
  compatibility/2006"
  xmlns:v2="http://schemas.openxmlformats.org/Circles/v2"
  mc:Ignorable="v2"
  mc:ProcessContent="v2:Blink" >
```

```
    <v2:Watermark Opacity="v0.1">
      <Circle Center="0,0" Radius="20" Color="Blue" />
      <Circle Center="25,0" Radius="20" Color="Black" />
      <Circle Center="50,0" Radius="20" Color="Red" />
    </v2:Watermark>
    <v2:Blink>
      <Circle Center="13,0" Radius="20" Color="Yellow" />
      <Circle Center="38,0" Radius="20" Color="Green" />
    </v2:Blink>
  </Circles>
```

The Version 1 markup consumer, unaware of Version 2 markup, renders the above markup as if it had processed the following markup:

```
  <Circles
    xmlns="http://schemas.openxmlformats.org/Circles/v1" >
    <Circle Center="13,0" Radius="20" Color="Yellow" />
    <Circle Center="38,0" Radius="20" Color="Green" />
  </Circles>
```

end example]

10.1.3 PreserveElements and PreserveAttributes Attributes

The Ignorable attribute presents a challenge unique to markup editors in deciding when to *preserve* ignored markup in the face of modification and when to discard ignored markup.

A markup editor might use the presence of PreserveElements and PreserveAttributes attributes as hints for deciding what ignored elements and attributes to preserve when markup is modified. The markup consumer might differ in interpretation of those hints according to the markup specification of the particular markup document being edited.

Markup specifications should specify conditions under which markup editors might preserve ignored markup. If a markup specification lacks such guidance, markup editors for markup documents based on that markup specification should discard all ignored elements or attributes. [*Note:* Even if a markup specification provides such guidance, a markup editor might choose to discard all ignored markup without regard to the presence of any PreserveElements or PreserveAttributes attribute. *end note*]

A markup specification can restrict preservation of elements identified by the PreserveElements attribute to those elements that are descendants of particular elements. Likewise, a markup specification can restrict the set of elements whose ignored attributes can be preserved, as identified by the PreserveAttributes element, to those that are descendants of particular elements. Regardless of any such restrictions, markup consumers shall always accept, but possibly ignore, PreserveElements and PreserveAttributes attributes on any element.

10.1.3.1 PreserveElements Attribute

The PreserveElements attribute is declared as a whitespace-delimited list of element qualified names identifying the expanded names of elements that should be preserved when the markup is edited, even if the elements themselves are ignored. In any qualified name in the list, the wildcard character "*" can replace the local name to indicate that all elements in the namespace should be preserved.

A markup consumer that encounters an ignored element whose expanded name matches the expanded name of an element declared in the PreserveElements attribute value, the markup consumer shall consider that element to be a preserved element, regardless of whether or not the element's qualified name matches the qualified name declared in the PreserveElements attribute value.

When an element is ignored and preserved, all of its unprefixed attributes shall also be preserved along with any preserved attributes declared in the PreserveAttributes attribute value.

The PreserveElements attribute value shall not reference any element name that does not belong to a namespace that is identified by the Ignorable attribute declared on the same element.

The value of the PreserveElements attribute can be an empty or blank string. When a markup consumer encounters such a value, it shall proceed as if the PreserveElements attribute was not declared.

10.1.3.2 PreserveAttributes Attributes

The PreserveAttributes attribute is declared as a whitespace-delimited list of attribute qualified names identifying the expanded names of attributes that should be preserved when the markup is edited. [*Note*: An attribute cannot be preserved unless it appears on a preserved element. *end note*] In any qualified name in the list, the wildcard character "*" can replace the local name to indicate that all attributes in the namespace should be preserved.

A markup consumer that encounters an ignored attribute whose expanded name matches the expanded name of an attribute identified in the PreserveAttributes attribute value, the markup consumer shall consider that attribute to be a preserved attribute, regardless of whether or not the attribute's qualified name matches the qualified name declared in the PreserveAttributes attribute value.

A markup consumer shall not determine preservation of any unprefixed local attributes of an element based on the value of the PreserveAttributes attribute

The ProcessAttributes attribute value shall not reference any attribute name that does not belong to a namespace that is identified by the Ignorable attribute declared on the same element.

The value of the PreserveAttributes attribute can be an empty or blank string. When a markup consumer encounters such a value, it shall proceed as if the PreserveAttributes attribute was not declared.

10.1.4 MustUnderstand Attribute

The MustUnderstand attribute is declared as a whitespace-delimited list of namespace prefixes identifying a set of namespace names. A markup consumer that does not understand these identified namespaces shall not

continue to process the markup document, regardless of whether the non-understood namespace was declared as an ignorable namespace on an ancestor element. Markup consumers shall generate an error condition if one or more of these identified namespaces is not understood.

The value of the MustUnderstand attribute can be an empty or blank string. When a markup consumer encounters such a value, it shall proceed as if the MustUnderstand attribute was not declared.

[*Note*: §10.2 clarifies the rules for processing the MustUnderstand attribute when it is applied to a Choice or Fallback element, or when it is applied to a descendant element of one of those elements. *end note*]

[*Example*: Example 10–4. Processing an attribute's prefixed qualified name

The declaration of a Version 2 attribute causes a Version 1 markup consumer to trigger an error when processing the last Circle element.

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
  compatibility/2006"
  xmlns:v2="http://schemas.openxmlformats.org/Circles/v2">
  <Circle Center="0,0" Radius="20" Color="Blue" />
  <Circle Center="25,0" Radius="20" Color="Black" />
  <Circle Center="50,0" Radius="20" Color="Red" />
  <Circle Center="13,0" Radius="20" Color="Yellow" />
  <Circle Center="38,0" Radius="20" Color="Green"
    v2:Opacity="0.5" />
</Circles>
```

Example 10–5. Processing a MustUnderstand attribute

The value of the MustUnderstand attribute causes a Version 1 markup consumer to trigger an error when processing the root Circles element.

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc=http://schemas.openxmlformats.org/markup-
  compatibility/2006
  xmlns:v2="http://schemas.openxmlformats.org/Circles/v2"
  mc:MustUnderstand="v2">
  <Circle Center="0,0" Radius="20" Color="Blue" />
  <Circle Center="25,0" Radius="20" Color="Black" />
  <Circle Center="50,0" Radius="20" Color="Red" />
  <Circle Center="13,0" Radius="20" Color="Yellow" />
  <Circle Center="38,0" Radius="20" Color="Green"
    v2:Opacity="0.5" />
</Circles>
```

end example]

10.2 Alternate-Content Elements

[*Note*: Markup producers can generate a markup document that includes multiple markup alternatives, each labeled with the namespaces that needs to be understood by any markup consumer choosing that alternative. A markup consumer shall choose only a single alternative. A particular markup alternative can exploit features introduced in subsequent revisions of the markup specification or in extensions to the markup specification.

In some cases, the Ignorable attribute could provide flexibility sufficient for a markup producer to create an acceptable experience to users of a markup consumer that is unaware of any revisions or extensions. In other cases, it could be desirable or necessary for markup producers to provide different markup alternatives, with one alternative processed by markup consumers implemented according to particular revisions or extensions of the markup specification, and others processed by markup consumers implemented according to different revisions or extensions of the markup specification. *end note*]

10.2.1 AlternateContent Element

The AlternateContent element contains the full set of all possible markup alternatives. Each possible alternative is contained within either a Choice or Fallback child element of the AlternateContent element.

An AlternateContent element shall contain one or more Choice child elements, optionally followed by a Fallback child element. If present, there shall be only one Fallback element, and it shall follow all Choice elements. An AlternateContent element shall not be the child of an AlternateContent element.

More than one Choice child element might be specified, each identifying the namespaces that a markup consumer needs to understand in order to choose the markup alternative contained within the Choice element. Markup consumers shall rely solely on the namespaces identified by the Choice element rather than the alternate content markup itself in order to decide which content to use.

AlternateContent elements may include the attributes Ignorable, MustUnderstand, ProcessContent, PreserveElements, and PreserveAttributes described in this Part. These attributes' qualified names shall be prefixed when associated with an AlternateContent element. A markup consumer shall generate an error if it encounters an unprefixed attribute name associated with an AlternateContent element. [*Note*: A namespace declaration is not considered to be an unprefixed attribute name. *end note*]

AlternateContent elements may contain elements or attributes that are ignored according to the rules in §10.1. Therefore, in addition to Choice and Fallback elements, an element from a non-understood and ignorable namespace may occur as a child of AlternateContent. However, a markup consumer shall generate an error if it encounters any child element from a namespace that is neither understood nor ignorable.

When a markup consumer processes an AlternateContent element, each successive Choice or Fallback element is considered in markup order for selection based on its attributes. If a Choice or Fallback element is not selected for processing, all of the child and descendant elements of that Choice or Fallback element shall be treated as if they did not exist. A markup consumer shall not generate an error based on any MustUnderstand attribute applied to an element contained within the content of the unselected Choice or

Fallback element. A markup consumer shall not generate an error based on any markup that is contained within an unselected Choice or Fallback element, even if that markup is not conformant to this Part.

In processing an AlternateContent element, the attributes of every child Choice or Fallback element shall be processed and checked for conformance to this Part, regardless of whether the Choice or Fallback element precedes or follows the selected alternative in markup order. If the AlternateContent element's child Choice or Fallback elements include an attribute from a namespace that is not understood and is not ignorable, the markup consumer shall generate an error. Likewise, a markup consumer shall generate an error if it encounters a MustUnderstand attribute included on a Choice or Fallback element that identifies a namespace that it does not understand.

The content of the selected Choice or Fallback element is processed as though it replaces the entire AlternateContent element. All namespace declarations and compatibility-rule attributes present on the AlternateContent element or selected Choice or Fallback element shall be processed as though they appeared on every child element of the selected Choice or Fallback element.

[*Note*: The AlternateContent element can appear as the root element of a markup document. *end note*]

AlternateContent elements shall not declare an xml:lang or xml:space attribute.

10.2.2 Choice Element

All Choice elements shall have a Requires attribute whose value contains a whitespace-delimited list of namespace prefixes that identify the namespaces that a markup consumer needs to understand to select that Choice and process the content. If the markup consumer does not understand all of the namespaces identified, it shall not select that Choice element. Markup consumers shall select the first Choice element, in markup order, in which all namespaces identified by the Requires attribute are understood.

Choice elements can include the attributes Ignorable, MustUnderstand, ProcessContent, PreserveElements, and PreserveAttributes described in this Part. These attributes shall be prefixed when declared on a Choice element. A markup consumer shall generate an error if it encounters a Choice element having any unprefixed attribute name, with the single exception of the Requires attribute, which shall be unprefixed. [*Note*: A namespace declaration is not considered to be an unprefixed attribute name. *end note*] A markup consumer that encounters a prefixed Requires attribute, when the prefix is associated with the Markup Compatibility namespace, shall generate an error.

Choice elements shall not carry an xml:lang or xml:space attribute.

10.2.3 Fallback Element

If no Choice element can be selected, markup consumers shall use the content provided in a Fallback element, if present. If no Choice element can be selected and no Fallback element is provided, the document content is processed as if the AlternateContent element were absent.

Fallback elements can include the attributes Ignorable, MustUnderstand, ProcessContent, PreserveElements, and PreserveAttributes described in this Part. These attributes shall be prefixed when

declared on a Fallback element. A markup consumer shall generate an error if it encounters a Fallback element having any unprefixed attribute name. [*Note*: A namespace declaration is not considered to be an unprefixed attribute name. *end note*]

Fallback elements can include ignored attributes.

Fallback elements shall not declare an xml:lang or xml:space attribute.

10.2.4 Alternate-Content Examples

The following examples illustrate the usage of alternate-content elements.

[*Example*: Example 10–6. Processing AlternateContent markup

In this example, luminance is expressed as an attribute of a Circle element for markup consumers supporting the Version 3 namespace, identified by the v3 namespace prefix, and as an attribute of a LuminanceFilter element for other markup consumers.

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc=http://schemas.openxmlformats.org/markup-
  compatibility/2006
  xmlns:v2=http://schemas.openxmlformats.org/Circles/v2
  xmlns:v3=http://schemas.openxmlformats.org/Circles/v3
  mc:Ignorable="v2 v3">
  <mc:AlternateContent>
    <mc:Choice Requires="v3">
      <v3:Circle Center="0,0" Radius="20" Color="Blue"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="25,0" Radius="20" Color="Black"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="50,0" Radius="20" Color="Red"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="13,0" Radius="20" Color="Yellow"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="38,0" Radius="20" Color="Green"
         Opacity="0.5" Luminance="13" />
    </mc:Choice>
    <mc:Fallback>
      <LuminanceFilter Luminance="13">
         <Circle Center="0,0" Radius="20" Color="Blue"
           v2:Opacity="0.5" />
         <Circle Center="25,0" Radius="20" Color="Black"
           v2:Opacity="0.5" />
         <Circle Center="50,0" Radius="20" Color="Red"
           v2:Opacity="0.5" />
         <Circle Center="13,0" Radius="20" Color="Yellow"
           v2:Opacity="0.5" />
```

```
            <Circle Center="38,0" Radius="20" Color="Green"
              v2:Opacity="0.5" />
          </LuminanceFilter>
        </mc:Fallback>
      </mc:AlternateContent>
    </Circles>
```

end example]

[*Example*: Example 10–7. Processing AlternateContent markup using namespaces

In this example, if the markup consumer understands the metallic-finishes namespace, the contents of the mc:Choice block are used. If it does not, the contents of mc:Fallback are used instead.

```
    <Circles
      xmlns="http://schemas.openxmlformats.org/Circles/v1"
      xmlns:mc="http://schemas.openxmlformats.org/markup-
      compatibility/2006" >
      <mc:AlternateContent
        xmlns:m="http://schemas.openxmlformats.org/metallic-
        finishes/v1">
        <mc:Choice Requires="m">
          <Circle m:Finish="GoldLeaf" Center="100,100" Radius="50"
            />
        </mc:Choice>
        <mc:Fallback>
          <Circle Fill="Gold" Center="100,100" Radius="50" />
        </mc:Fallback>
      </mc:AlternateContent>
    </Circles>
```

end example]

11. Namespace Subsumption

11.1 The Subsumption Process

A namespace *subsumes* another namespace if it includes all of the elements, attributes, and attribute values of the subsumed namespace.

A markup specification that defines a subsuming namespace shall require that any instance of the following constructs that would be recognized in the subsumed namespace shall also be recognized and interpreted identically in the subsuming namespace.

- Element local names
- Unprefixed attribute names of elements
- Prefixed attribute names of elements
- Attribute values

Subsumption is the process by which a markup consumer recognizes and interprets elements, attributes, and attribute values in a subsumed namespace as if they occurred in the subsuming namespace.

When performing subsumption, markup consumers might confirm that the markup using the old namespace is compatible with the specification of the old namespace

Any markup specification that relies on the Markup Compatibility namespace should define a namespace naming convention for use by future versions of that specification when those future versions introduce new namespaces that subsume older namespaces.

When markup consumers encounter a non-understood namespace name, they might examine it for a naming convention that suggests that an understood namespace is being subsumed. The suggestion of a subsumption relationship shall not suppress error processing triggered by the non-understood namespace name. If error processing is triggered, markup consumers might use the suggested subsumption relationship to choose the most appropriate error message or error code. [*Example*: One error message might encourage upgrading to a newer application version, while another might merely highlight the non-understood namespace name. *end example*]

11.2 Special Considerations for Attributes

§6.3 of the W3C Recommendation “Namespaces in XML 1.0 (Second Edition)” makes a distinction between prefixed attributes and unprefixed attributes sharing the same local name. Given an element whose expanded name refers to namespace *N*, an unprefixed attribute with local name *L* is distinct from a prefixed attribute with local name *L* and namespace *N*. The existence of this distinction is important in defining correct subsumption behavior for markup consumers.

A subsuming namespace might extend a pre-existing element local name with a new unprefixed attribute. Similarly, a subsuming namespace might create a new attribute designed for prefixed use.

[*Note*: The statements above introduce a potential ambiguity in defining the correct behavior of a markup consumer performing subsumption.

Assume that the namespace associated with prefix v2 subsumes the namespace associated with v1. Suppose a markup consumer that understands the v2 namespace encounters markup of the following form:

```
<v1:OldElement mc:Ignorable="v2" v2:NewAttibute="value" />
```

How should that markup consumer interpret that markup? Should it be considered equivalent to the following markup?

```
<v2:OldElement NewAttribute="value" />
```

Or should it be considered equivalent to the following markup?

```
<v2:OldElement v2:NewAttribute="value" />
```

According to "Namespaces in XML 1.0 (Second Edition)", these two potential pieces of markup are not equivalent. Additionally, the XML Schema specification allows for the construction of different XSD schemas that validate one, the other, or both of these constructs. *end note*]

When subsuming an element from an older namespace that carries a prefixed attribute from a newer, subsuming namespace, a markup consumer shall decide whether to treat the new attribute as if its expanded name refers to the new namespace or as if its expanded name refers to no namespace. If a subsuming namespace adds a new attribute or permissible attribute value to an element that was present in the subsumed namespace, the markup specification that defines the subsuming namespace shall state which of the two subsumption behaviors shall be used by markup consumers and assumed by markup producers.

[*Note*: 13-1 illustrates how a markup preprocessor handles each of the two possible behaviors. *end note*]

In order to support a preprocessing model for Markup Compatibility elements and attributes, specifications should restrict the use of any combination of prefixed and unprefixed attributes with the same local name.

A namespace should not be subsumed by a newer namespace if the older namespace includes both a prefixed attribute and an unprefixed attribute sharing its local name but having a different type or different semantics.

A subsuming namespace should not include both a prefixed attribute and an unprefixed attribute sharing its local name but having a different type or different semantics.

12. Application-Defined Extension Elements

A markup specification using Markup Compatibility elements and attributes might define one or more specific extension elements in the namespaces it defines. *Extension elements* suspend Markup Compatibility processing within their content. Except as noted below, within the content of an extension element, markup consumers shall not treat elements and attributes from non-understood namespaces as Markup Compatibility errors. Similarly, under the same conditions, markup consumers shall disregard elements and attributes from the Markup Compatibility namespace.

A specification for an element nested somewhere within an extension-element might require a markup consumer to re-establish Markup Compatibility processing. Within the scope of such a nested element and its content, a markup consumer shall disregard all Markup Compatibility attributes that were encountered on elements outside of the element that re-establishes Markup Compatibility processing. Within the scope of such a nested element, a markup consumer might understand a set of namespaces that is different from the set of namespaces understood at the point in the markup where the extension element was encountered.

The following examples illustrate two uses of application-defined extension elements:

[*Example*: Example 12–1. An application-defined XML island

An extension element can be used to introduce an “island” of unprocessed XML whose markup is otherwise unconstrained by the application's specification. The specification of the island element can further require preservation of the contents of the island by markup processors, without requiring the use of the PreserveElements and PreserveAttributes Markup Compatibility attributes. *end example*]

[*Example*: Example 12–2. An application-defined add-in element

Some markup specifications and markup consumers can use an extension element to implement an add-in model. In an add-in model, the specification for the contents of the extension element is separate from the specification for the extension element itself.

The specification for some particular nested content can include support for Markup Compatibility elements and attributes, while the specification for other nested could omit such support. If the specification for the nested content does include support for Markup Compatibility elements and attributes, the Markup Compatibility processing state is reset temporarily for processing of the nested content. Any Ignorable attribute-value associated with an extension element or any of its ancestor elements is “forgotten” during the processing of content nested within that extension element. In an add-in model the set of namespaces assumed to be understood when processing descendant elements of an extension element is completely unrelated to the set of understood namespaces when that extension element itself is processed.

In this example, the "Circles" specification includes an extension AddIn element, allowing for nested markup to be handled by a markup consumer that does not process Circle markup. The specification for the nested "TextFlow" markup does not provide for the processing of Markup Compatibility elements and attributes.

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc=http://schemas.openxmlformats.org/markup-
  compatibility/2006
  xmlns:v2="http://schemas.openxmlformats.org/Circles/v2"
  mc:Ignorable="v2 ">
  <Circle Center="0,0" Radius="20" Color="Blue"
    v2:Opacity="0.5" />
  <Circle Center="25,0" Radius="20" Color="Black"
    v2:Opacity="0.5" />
  <Circle Center="50,0" Radius="20" Color="Red"
    v2:Opacity="0.5" />
  <Circle Center="13,0" Radius="20" Color="Yellow"
    v2:Opacity="0.5" />
  <Circle Center="38,0" Radius="20" Color="Green"
    v2:Opacity="0.5" />
  <AddIn Center="25,10" Radius="10" CodeBase=
    "http://www.openxmlformats.org/code/TextFlowAddin.jar">
    <TextFlow
      xmlns="http://schemas.openxmlformats.org/TextFlow/v1">
      <!--
        Because the TextFlow specification does not make use
        of Markup Compatibility elements and attributes,
        the TextFlow processor would consider the presence
        of an mc:Ignorable attribute to be an error condition.
        Because the TextFlow specification is completely
        unaware of all versions of the Circles specification,
        the TextFlow processor would also consider the
        presence of a Circle or v2:Ellipse element to be an
        error condition.
      -->
      <Paragraph>How are <Bold>you</Bold>?</Paragraph>
    </TextFlow>
  </AddIn>
</Circles>
```

end example]

13. Preprocessing Model for Markup Consumption

This clause is informative

Markup consumers might rely on a preprocessing model to support Markup Compatibility. Such a model can simplify the markup consumer's implementation and allow it to rely on XML schema validation for the preprocessed markup document. Furthermore, a single preprocessing implementation can be shared by markup consumers that target various markup specifications.

A markup preprocessor accepts as input XML markup containing a variety of elements and attributes from the following namespace categories:

- The Markup Compatibility namespace.
- *Current namespaces*: Understood namespaces that have not been subsumed by newer namespaces.
- *Obsolete namespaces*: Understood namespaces known to have been subsumed by newer namespaces).
- Non-understood namespaces.

The markup preprocessor transforms its input markup into output markup that contains elements and attributes drawn only from current and non-understood namespaces. This transformation is disabled for input markup nested within an application-defined extension element. The markup preprocessor accomplishes its transformation using the following rules:

The markup preprocessor checks for the correct usage of all elements and attributes in the Markup Compatibility namespace, and triggers error processing if an element or attribute in that namespace violates the requirements of this Markup Compatibility Part.

The markup preprocessor interprets occurrences of the MustUnderstand attribute, and triggers error processing when appropriate.

The markup preprocessor removes all elements and attributes in the Markup Compatibility namespace, removing the contents of unselected Choice and Fallback elements.

The markup preprocessor removes all elements and attributes in obsolete namespaces, and replaces them with identically named elements and attributes in current namespaces.

Guided by the Ignorable attribute, the markup preprocessor removes some elements and attributes in non-understood namespaces, leaving others undisturbed.

Guided by the ProcessContent attribute, the markup preprocessor removes all nested content within some removed elements from non-understood namespaces.

In addition to producing such transformed output, the markup preprocessor might also implement mechanisms to optionally provide to a markup editor the additional information necessary to preserve some ignored content

[*Note*: The output of the markup preprocessor can be validated against a schema written entirely in terms of current namespaces. Where a schema does not allow for an open attribute or content model, occurrences of elements and attributes from non-understood namespaces in the markup preprocessor's output will trigger validation failures. Where a schema does allow for an open attribute or content model, occurrences of elements and attributes from non-understood namespaces will validate successfully.

If a markup preprocessor is used in conjunction with XSD schema validation, the schema should set the value of the elementFormDefault attribute of the root schema element to the value `qualified`. *end note*]

[*Example*: Example 13–3. Preprocessing using Markup Compatibility elements and attributes

Given the following input document:

```
<Circles
  xmlns="http://schemas.openxmlformats.org/Circles/v1"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
  compatibility/2006"
  xmlns:v2="http://schemas.openxmlformats.org/Circles/v2"
  xmlns:v3="http://schemas.openxmlformats.org/Circles/v3"
  mc:Ignorable="v3"
  mc:ProcessContent="v3:Blink">
  <mc:AlternateContent>
    <mc:Choice Requires="v3">
      <v3:Circle Center="0,0" Radius="20" Color="Blue"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="25,0" Radius="20" Color="Black"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="50,0" Radius="20" Color="Red"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="13,0" Radius="20" Color="Yellow"
        Opacity="0.5" Luminance="13" />
      <v3:Circle Center="38,0" Radius="20" Color="Green"
        Opacity="0.5" Luminance="13" />
    </mc:Choice>
    <mc:Fallback>
      <LuminanceFilter Luminance="13">
        <Circle Center="0,0" Radius="20" Color="Blue"
          v2:Opacity="0.5" />
        <Circle Center="25,0" Radius="20" Color="Black"
          v2:Opacity="0.5" />
        <Circle Center="50,0" Radius="20" Color="Red"
          v2:Opacity="0.5" />
```

```
            <Circle Center="13,0" Radius="20" Color="Yellow"
              v2:Opacity="0.5" />
            <Circle Center="38,0" Radius="20" Color="Green"
              v2:Opacity="0.5" />
          </LuminanceFilter>
        </mc:Fallback>
      </mc:AlternateContent>
      <v3:Blink>
        <Circle Center="13,0" Radius="20" Color="Yellow" />
        <Circle Center="38,0" Radius="20" Color="Green" />
      </v3:Blink>
      <v3:Watermark>
        <Circle Center="50,0" Radius="20" Color="Red" />
      </v3:Watermark>
    </Circles>
```

According to §11.2, this markup document's markup specification, when it defines the namespace name `http://schemas.openxmlformats.org/Circles/v2`, shall state the intended subsumption behavior when processing a Version 2 Opacity attribute that is applied to a Circle element from an earlier version. Depending on that statement, a Version 2 markup consumer renders the above markup as if it had processed one of the following pieces of preprocessed markup:

```
    <v2:Circles
      v2:xmlns="http://schemas.openxmlformats.org/Circles/v2">
      <v2:LuminanceFilter Luminance="13">
        <v2:Circle Center="0,0" Radius="20" Color="Blue"
          Opacity="0.5" />
        <v2:Circle Center="25,0" Radius="20" Color="Black"
          Opacity="0.5" />
        <v2:Circle Center="50,0" Radius="20" Color="Red"
          Opacity="0.5" />
        <v2:Circle Center="13,0" Radius="20" Color="Yellow"
          Opacity="0.5" />
        <v2:Circle Center="38,0" Radius="20" Color="Green"
          Opacity="0.5" />
      </v2:LuminanceFilter>
      <v2:Circle Center="13,0" Radius="20" Color="Yellow" />
      <v2:Circle Center="38,0" Radius="20" Color="Green" />
    </v2:Circles>
```

Or:

```
    <v2:Circles
      v2:xmlns="http://schemas.openxmlformats.org/Circles/v2">
      <v2:LuminanceFilter Luminance="13">
        <v2:Circle Center="0,0" Radius="20" Color="Blue"
          v2:Opacity="0.5" />
        <v2:Circle Center="25,0" Radius="20" Color="Black"
          v2:Opacity="0.5" />
```

```
        <v2:Circle Center="50,0" Radius="20" Color="Red"
          v2:Opacity="0.5" />
        <v2:Circle Center="13,0" Radius="20" Color="Yellow"
          v2:Opacity="0.5" />
        <v2:Circle Center="38,0" Radius="20" Color="Green"
          v2:Opacity="0.5" />
      </v2:LuminanceFilter>
      <v2:Circle Center="13,0" Radius="20" Color="Yellow" />
      <v2:Circle Center="38,0" Radius="20" Color="Green" />
    </v2:Circles>
```

end example]

End of informative text

Annex A. Bibliography

This annex is informative.

The following documents are useful references for implementers and users of this Part, in addition to the normative references:

ISO/IEC Directives Part 2, Rules for the structure and drafting of International Standards, Fourth edition, 2001, ISBN 92-67-01070-0.

Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation, 16 August 2006.

Namespaces in XML 1.0 (Second Edition), W3C Recommendation, 16 August 2006.

RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax, The Internet Society*, Berners-Lee, T., R. Fielding, and L. Masinter, 2005, http://www.rfc-editor.org.

RFC 4234 *Augmented BNF for Syntax Specifications: ABNF, The Internet Society*, Crocker, D., P. Overell, 2005

XML Base, W3C Recommendation, 27 June 2001.

XML Schema Part 1: Structures Second Edition, W3C Recommendation, 28 October 2004.

XML Schema Part 2: Datatypes Second Edition, W3C Recommendation, 28 October 2004.

End of informative text.

Annex B. Index

This annex is informative.

End of informative text