ecma INTERNATIONAL

Standard ECMA-376

3rd Edition / June 2011

Office Open XML File Formats – Part 3

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

is the registered trademark of Ecma International

COPYRIGHT PROTECTED DOCUMENT

Table of Contents

Foreword ..... **v**

Introduction ..... **vi**

1. Scope **1**

2. Conformance **2**

2.1 Document Conformance 2

2.2 Application Conformance 2

3. Normative References **3**

4. Terms and Definitions **4**

5. Notational Conventions **6**

6. Acronyms and Abbreviations **7**

7. General Description **8**

8. Overview **9**

9. Markup Compatibility Fundamentals **10**

9.1 Core Concepts 10

9.2 Markup Compatibility Namespace 11

10. Markup Compatibility Attributes and Elements **12**

10.1 Compatibility-Rule Attributes 13

10.1.1 Ignorable Attribute 14

10.1.2 ProcessContent Attribute 16

10.1.3 PreserveElements and PreserveAttributes Attributes 18

10.1.4 MustUnderstand Attribute 20

10.2 Alternate-Content Elements 21

10.2.1 AlternateContent Element 21

10.2.2 Choice Element 22

10.2.3 Fallback Element 23

10.2.4 Alternate-Content Examples 23

11. Namespace Subsumption **26**

11.1 The Subsumption Process 26

11.2 Special Considerations for Attributes 26

12. Application-Defined Extension Elements **29**

13. Preprocessing Model for Markup Consumption **31**

Annex A. (informative) Validation Using NVDL **35**

A.1 Validation Against Requirements of this Part of ECMA-376 35

A.2 Validation Against the Combination of Office Open XML and Extensions 36

Annex B. (informative) Index **40**

Foreword

Changes from the 2nd edition were made to align this 3rd edition Standard with ISO/IEC 29500:2011. Both this 3rd edition and ISO/IEC 29500:2011 refer to the 1st edition. As such, this 3rd edition does not cancel or replace the 1st edition. This 3rd edition does, however, cancel and replace the 2nd edition.

ECMA-376 consists of the following parts:

- *Part 1: Fundamentals and Markup Language Reference*
- *Part 2: Open Packaging Conventions*
- *Part 3: Markup Compatibility and Extensibility*
- *Part 4: Transitional Migration Features*

Annexes A and B are for information only.

Introduction

ECMA-376 specifies 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 existing corpus of Microsoft Office documents.

The following organizations have participated in the creation of ECMA-376 and their contributions are gratefully acknowledged:

Apple, Barclays Capital, BP, The British Library, Essilor, Intel, Microsoft, NextPage, Novell, Statoil, Toshiba, and the United States Library of Congress

1. Scope

This Part of ECMA-376 describes a set of conventions that are used by Office Open XML documents to clearly mark elements and attributes introduced by future versions or extensions of Office Open XML documents, while providing a method by which consumers can obtain a baseline version of the Office Open XML document (a version without extensions) for interoperability.

2. Conformance

The text in this Part of ECMA-376 is divided into *normative* and *informative* categories. Unless documented otherwise, any feature shall be implemented as specified by the normative text describing that feature in this Part of ECMA-376. Text marked informative (using the mechanisms described in §7) is for information purposes only. Unless stated otherwise, all text is normative.

Use of the word “shall” indicates required behavior.

Any behavior that is not explicitly specified by this Part of ECMA-376 is implicitly unspecified (Part 1, §4).

Each Part of this multi-part standard has its own conformance clause. The term *conformance class* is used to disambiguate conformance within different Parts of this multi-part standard. This Part of ECMA-376 has only one conformance class, *MCE* (that is, Markup Compatibility and Extensibility). As such, conformance to that class implies conformance to the whole Part.

2.1 Document Conformance

A document has conformance class MCE if it satisfies the syntax constraints on elements and attributes defined in this Part of ECMA-376. Document conformance to this Part of ECMA-376 is purely syntactic.

2.2 Application Conformance

An application implementing this Part of ECMA-376 has conformance class MCE if any one of the following is true:

- The application is a markup consumer that does not reject any documents of conformance class MCE; or
- The application is a markup producer that is able to produce documents of conformance class MCE; or
- The application is a markup editor that does not reject any documents of conformance class MCE, and is able to produce documents of conformance class MCE.

Application conformance to this Part of ECMA-376 is purely syntactic.

[*Note*: Application conformance to this Part of ECMA-376 cannot be based on semantics, since the semantics depend on the choice of application-defined extension elements. *end note*]

3. Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 2382-1:1993, *Information technology — Vocabulary — Part 1: Fundamental terms*.

ISO/IEC 10646, *Information technology — Universal Coded Character Set (UCS)*.

ISO/IEC 19757-4:2006, *Information technology — Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language (NVDL)*.

RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax,* The Internet Society, Berners-Lee, T., R. Fielding, and L. Masinter, 2005, http://www.ietf.org/rfc/rfc3986.txt.

RFC 4234 *Augmented BNF for Syntax Specifications: ABNF,* The Internet Society, Crocker, D., P. Overell, 2005, http://www.ietf.org/rfc/rfc4234.txt

The Unicode Consortium. The Unicode Standard, http://www.unicode.org/standard/standard.html.

XML, Tim Bray, Jean Paoli, Eve Maler, C. M. Sperberg-McQueen, and François Yergeau (editors). Extensible Markup Language (XML) 1.0, Fourth Edition.1 World Wide Web Consortium. 2006. http://www.w3.org/TR/2006/REC-xml-20060816/ [Implementers should be aware that a further correction of the normative reference to XML to refer to the 5th Edition will be necessary when the related Reference Specifications to which this International Standard also makes normative reference and which also depend upon XML, such as XSLT, XML Namespaces and XML Base, are all aligned with the 5th Edition.]

XML Base, Marsh, Jonathan. *XML Base*. World Wide Web Consortium. 2001. http://www.w3.org/TR/2001/REC-xmlbase-20010627/

XML Namespaces, Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin (editors). *Namespaces in XML 1.0* (Third Edition), 8 December 2009. World Wide Web Consortium. http://www.w3.org/TR/2009/REC-xml-names-20091208/

XML Schema Part 0: Primer (Second Edition), W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-0/

XML Schema Part 1: Structures (Second Edition), W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes (Second Edition), W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-2/

4. Terms and Definitions

For the purposes of this document, the following terms and definitions apply. Other terms are defined where they appear in *italics* typeface. Terms not explicitly defined in this Part of ECMA-376 are not to be presumed to refer implicitly to similar terms defined elsewhere.

Throughout this Part of ECMA-376, 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."

alternate content — A set of alternatives of XML markup and character data, 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 of ECMA-376 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. [*Note*: Because a markup consumer might be implemented as a markup pre-processor, this term is not coalesced with the definition for a consumer, which would process the XML document output by the markup pre-processor. *end note*]

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 character data, and acts as the producer of the modified markup and character data.

markup preprocessor — A software module, designed for use in the implementation of markup consumers, that follows the rules of this Part of ECMA-376 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 format definition that incorporates all of the namespaces, elements, attributes, and requirements specified in this Part of ECMA-376.

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.

qualified attribute name — An attribute's qualified name.

qualified element name — An element's qualified name.

recognize — To identify that an XML element, XML attribute, or attribute-value is defined in this Part of ECMA-376 or in the markup specification against which the containing XML document purports to be conformant.

5. Notational Conventions

The following typographical conventions are used in ECMA-376:

1. The first occurrence of a new term is written in italics. [*Example*: The text in ECMA-376 is divided into *normative* and *informative* categories. *end example*]
2. In each definition of a term in §4 (Terms and Definitions), the term is written in bold. [*Example*: **behavior** — External appearance or action. *end example*]
3. The tag name of an XML element is written using an Element style. [*Example*: The bookmarkStart and bookmarkEnd elements specify … *end example*]
4. The name of an XML attribute is written using an Attribute style. [*Example*: The dropCap attribute specifies … *end example*]
5. The value of an XML attribute is written using a constant-width style. [*Example*: The attribute value of `auto` specifies … *end example*]
6. The qualified or unqualified name of a simple type, complex type, or base datatype is written using a Type style.[*Example*: The possible values for this attribute are defined by the ST_HexColor simple type. *end example*]

6. Acronyms and Abbreviations

This clause is informative

The following acronyms and abbreviations are used throughout this Part of ECMA-376

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 of ECMA-376 is divided into the following subdivisions:

1. Front matter (clauses 1–7);
2. Overview and introductory material (clause 8–9);
3. Main body (clauses 10–13);
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 form the normative pieces of this Part of ECMA-376:

- Clauses 1–5, 7, and 9–12

The following form the informative pieces of this Part of ECMA-376:

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

Except for whole clauses or annexes that are identified as being informative, informative text that is contained within normative text is indicated in the following ways:

1. [*Example:* code fragment, possibly with some narrative … *end example*]
2. [*Note:* narrative … *end note*]
3. [*Rationale:* narrative … *end rationale*]
4. [*Guidance*: narrative … *end guidance*]

8. Overview

This clause is informative

This Part of ECMA-376 describes a set of XML elements and attributes whose purpose is to collectively enable producers to explicitly guide consumers in their handling of any XML elements and attributes not understood by the consumer.

These elements and attributes are intended to enable the creation of future versions of and extensions to ECMA-376, while enabling these desirable compatibility goals:

- A markup producer can produce markup documents that exploit new features defined by versions and extensions, yet remain interoperable with markup consumers that are unaware of those versions and extensions.
- For any such markup document, a markup consumer whose implementation is aware of the exploited versions and extensions can deliver functionality that is enhanced by the markup document's use of those versions and extensions.
- For any such markup document, the markup producer can enable and precisely control graceful degradation that might occur when the markup document is processed by a markup consumer that is unaware of the exploited versions and extensions.

End of informative text

9. Markup Compatibility Fundamentals

9.1 Core Concepts

Any XML-based document specification can use the markup language described in this Part of ECMA-376 as the basis of its compatibility with previous and future specification revisions, and to enable the creation of independent extensions of its specification.

This Part of ECMA-376 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 the elements and attributes within understood namespaces, including those containing elements and attributes defined in the markup specification. Markup consumers shall treat 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; however, 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 was content of the ignored element's parent, located in the same contextual position as the ignored element.

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 saving a 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 and uses identically the element local names, prefixed and unprefixed attribute names, attribute values, and element contents. 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 of ECMA-376 can be implemented using a 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.

Markup consumers should report errors when processing non-conforming documents.

9.2 Markup Compatibility Namespace

The following is the Markup Compatibility namespace name:

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

The Markup Compatibility namespace includes XML elements and attributes that markup producers can use to express to markup consumers how they shall respond to elements and attributes encountered that belong to non-understood namespaces. The elements and attributes defined in this Part of ECMA-376 are contained in the Markup Compatibility namespace. This namespace contains only those elements and attributes defined in this Part of ECMA-376.

10. Markup Compatibility Attributes and Elements

This Part of ECMA-376 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 values of attributes defined in this Part of ECMA-376 shall be normalized by markup consumers before processing markup compatibility elements and attributes, 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, which are further described in the subclauses that follow.

Table 10–2. Compatibility-rule attributes

Name	Description
Ignorable	A whitespace-delimited list of namespace prefixes identifying 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 qualified element 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 a markup producer suggests for preservation by markup editors, 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 a markup producer suggests for preservation by markup editors. 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 of ECMA-376 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 value contains 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 ProcessContent 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 with 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.

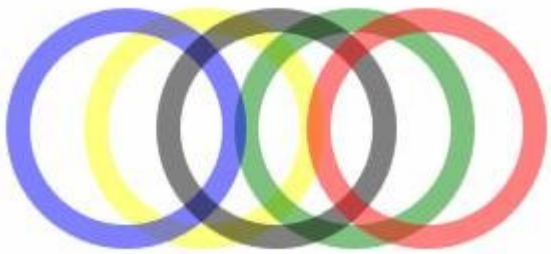
For a Version 2 markup consumer, only Luminance is an ignored attribute.

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 value contains a whitespace-delimited list of qualified element 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.

A markup consumer, when encountering an ignored element whose expanded name matches the expanded name of an element identified in the ProcessContent attribute value, shall consider that element to be a processed element, regardless of whether or not the qualified element name matches the qualified name specified in the ProcessContent attribute value. [*Note*: See Example 10–4 for further explanation of expanded name matching. *end note*]

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

The value of the ProcessContent attribute can be an empty string, although such values should be avoided. When a markup consumer encounters such a value, it shall proceed as if the ProcessContent attribute was not provided.

Markup producers shall not generate an element that has an xml:lang or xml:space attribute if that element is identified by a ProcessContent attribute value.

[*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]

[*Example*: Example 10–4. ProcessContent and expanded names

In the following example, extB:Blink is ignorable and is identified by the ProcessContent attribute, because extA and extB share the same namespace name and therefore the expanded names match.

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

```
    <extB:Blink>
      <Circle Center="0,0" Radius="20" Color="Blue" />
      <Circle Center="25,0" Radius="20" Color="Black" />
      <Circle Center="50,0" Radius="20" Color="Red" />
    </extB:Blink>
  </Circles>
```

end example]

10.1.3 PreserveElements and PreserveAttributes Attributes

The Ignorable attribute presents a challenge unique to markup editors in deciding when and how to *preserve* ignored markup in the face of modification and when to discard ignored markup. In the absence of explicit knowledge of the specification governing the unrecognized ignored markup, it is difficult for a markup editor to preserve ignored markup while simultaneously maintaining document conformance with whatever specification governs that markup. Similarly, it is difficult for the markup editor to avoid undesired change to the edited document's semantics, as they would be interpreted by a markup consumer whose implementation is based on the specification governing the preserved 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 editor's specific interpretation of those hints shall be governed by the markup specification or specifications that formed the basis for the markup editor's implementation.

Markup specifications should specify conditions under which markup editors should preserve ignored markup. Markup specifications should define the widest possible set of conditions under which markup editors should preserve ignored markup. [*Note*: If preservation conditions are too widely defined, future versions and extensions will be overly constrained in what new semantics they can introduce. *end note*]

If a markup specification lacks such guidance, markup editors for markup documents governed by that markup specification should be conservative in their preservation behavior Before preserving any ignored markup, markup editors should attempt to establish confidence that the preserved markup will be acceptable to, and interpretable with acceptable semantics by, all imaginable markup consumers that understand future versions of extensions. [*Note*: Such confidence could be established by deep understanding of the base specification. *end note*]

Even in the presence of explicit preservation guidance in a markup specification, any markup editor might choose to discard altogether all ignored markup without regard to the presence of any PreserveElements or PreserveAttributes attribute. [*Note:* The explicit presence of the Ignorable attribute indicates that discarding such markup before editing produces a document that conforms to relevant specifications and is self-sufficient in its semantic interpretation. *end note*]

A markup specification might define conditions calling for the preservation of some ignored elements or attributes without requiring the presence of any PreserveElements or PreserveAttributes attribute.

A markup specification might restrict preservation of elements identified by the PreserveElements attribute to those elements that are descendants of particular elements. Likewise, a markup specification might 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 disregard, PreserveElements and PreserveAttributes attributes on any element.

10.1.3.1 PreserveElements Attribute

The PreserveElements attribute value contains a whitespace-delimited list of element-qualified names identifying the expanded names of elements that a markup producer suggests for preservation by markup editors, even if the elements themselves are ignored. In any qualified name in the list, the wildcard character "*" can replace the local name to suggest that all elements in the namespace can be preserved.

If a markup consumer encounters an ignored element whose expanded name matches the expanded name of an element identified in the PreserveElements attribute value, the markup consumer shall consider that element to be a candidate for preservation, regardless of whether or not the element's qualified name matches the qualified name specified 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 identified in a 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 of 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 provided.

10.1.3.2 PreserveAttributes Attributes

The PreserveAttributes attribute value contains a whitespace-delimited list of attribute qualified names identifying the expanded names of attributes that a markup producer suggests for preservation by markup editors. [*Note*: An attribute cannot be preserved if it appears on a non-preserved ignored 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. [*Note*: An unprefixed local attribute cannot be identified by a PreserveAttributes attribute value. *end note*]

If a markup consumer 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 candidate for preservation, regardless of whether or not the attribute's qualified name matches the qualified name specified in the PreserveAttributes attribute value.

The ProcessAttributes attribute value shall not reference any attribute name that does not belong to a namespace that is identified by the Ignorable attribute of 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 provided.

10.1.4 MustUnderstand Attribute

The MustUnderstand attribute value contains 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 identified 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 10–5. 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 10–6. 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 labelled with the namespaces that need 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 might include the attributes Ignorable, MustUnderstand, ProcessContent, PreserveElements, and PreserveAttributes described in this Part of ECMA-376. 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 might have ignored attributes or contain ignored child elements. Markup consumers shall not generate an error when encountering such attributes or child elements. However, markup

consumers shall generate an error when encountering an attribute or child element of the AlternateContent element that belongs to a namespace that is neither understood nor ignorable. [*Note*: In addition to Choice and Fallback elements, an ignored element can occur as a child of AlternateContent. *end note*]

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 of ECMA-376.

In processing an AlternateContent element, the attributes of every child Choice or Fallback element shall be processed and checked for conformance to this Part of ECMA-376, regardless of whether the Choice or Fallback element precedes or follows the selected alternative in markup order. [*Note*: Checking the conformance of attributes of all Choice and Fallback elements, including those that follow the selected alternative, ensures that conformance violations detected by older markup consumers are detected by newer markup consumers that understand additional newer namespaces. *end note*] 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. This logical removal of the parent AlternateContent, Choice, and Fallback elements shall not change the expanded name of any element or attribute contained within the selected Choice of Fallback element. Additionally, this logical removal shall not change the set of ignorable namespaces, or their corresponding preservation and content-processing behavior, when processing the contents of the selected Choice or Fallback element.

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

Markup producers shall not generate AlternateContent elements that have the xml:lang or xml:space attributes. Markup consumers shall generate an error if they encounter the xml:lang or xml:space attributes on an AlternateContent element.

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 of ECMA-376. [*Note:* Although the MustUnderstand attribute might seldom appear on a Choice element, it is permitted there to avoid requiring markup consumers to special-case their handling of the MustUnderstand and Ignorable attribute pair. *end note*] These attributes shall be prefixed when 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 might have ignored attributes. Markup consumers shall not generate an error when encountering such attributes. However, markup consumers shall generate an error when encountering an attribute of the Choice element that belongs to a namespace that is neither understood nor ignorable.

Markup producers shall not generate Choice elements that have the xml:lang or xml:space attributes. Markup consumers shall generate an error if they encounter the xml:lang or xml:space attributes on a Choice element, regardless of whether the element is preceded by a selected Choice element.

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 of ECMA-376. These attributes shall be prefixed when 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 might have ignored attributes. Markup consumers shall not generate an error when encountering such attributes. However, markup consumers shall generate an error when encountering an attribute of the Fallback element that belongs to a namespace that is neither understood nor ignorable.

Markup producers shall not generate Fallback elements that have the xml:lang or xml:space attributes. Markup consumers shall generate an error if they encounter the xml:lang or xml:space attributes on a Fallback element, regardless of whether the element is preceded by a selected Choice element.

10.2.4 Alternate-Content Examples

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

[*Example*: Example 10–7. 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 10–8. 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 markup specification defines whether its usage of a namespace *subsumes* another namespace. The markup specification that defines a subsuming namespace shall require that the subsuming namespace include all of the elements, attributes, and attribute values of the subsumed namespace and shall further require that any instance of the following constructs that would be recognized in the subsumed namespace shall also be recognized and interpreted identically by consumers and producers in the subsuming namespace:

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

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” 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.

To illustrate, take the example where a version 2 of a markup specification subsumes all the elements and attributes in a version 1 markup specification. Further, 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:NewAttribute=”value” />
```

How should that markup consumer interpret that markup? Once it has been processed to remove the markup compatibility Ignorable attribute and perform the subsumption, 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” />
```

Note that during subsumption, v1:OldElement is replaced, or subsumed, by v2:OldElement.

According to “Namespaces in XML”, 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 processing 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*: Example 12-1 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 complex/simple 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 complex/simple 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.

[*Note*: the ext and extLst elements defined in Part 1 are examples of such extension elements. *end note*]

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 content 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 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 Part of ECMA-376.

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 trigger validation failures. Where a schema does allow for an open attribute or content model, occurrences of elements and attributes from non-understood namespaces 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 13–1. 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.
(informative)
Validation Using NVDL

This annex is informative.

Namespace-based Validation Dispatching Language (NVDL) allows documents to be decomposed into validation candidates, each of which can be validated independently.

NVDL can be used for validation against the normative requirements of this Part of ECMA-376. It can also be used for validation against the combination of Office Open XML documents (including the elements and attributes defined in this Part of ECMA-376) and any extensions.

A.1 Validation Against Requirements of this Part of ECMA-376

A markup document can satisfy requirements of this Annex without being an Office Open XML document. The following NVDL script examines whether a given document correctly uses the attributes and elements as defined by this Part of ECMA-376.

This NVDL script first extracts elements and attributes in the Markup Compatibility namespace, and then validates them against the appropriate RELAX NG schemas.

Note that AlternateContent, Choice and Fallback elements are allowed to have foreign elements and attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
  <namespace match="attributes" ns="http://schemas.openxmlformats.org/markup-
    compatibility/2006">
    <validate schemaType="application/relax-ng-compact-syntax">
      <schema>
        namespace mc="http://schemas.openxmlformats.org/markup-
         compatibility/2006"
        nsList = list { xsd:NCName* }
        qnameList = list { (xsd:QName | xsd:string {pattern = "\i\c*:\*" })*}
        start = element * {
          attribute mc:Ignorable { nsList }?,
          attribute mc:ProcessContent { qnameList }?,
          attribute mc:PreserveElements { qnameList }?,
          attribute mc:PreserveAttributes { qnameList }?,
```

```
            attribute mc:MustUnderstand { nsList }?
          }
        </schema>
      </validate>
    </namespace>
    <namespace match="elements" ns="http://schemas.openxmlformats.org/markup-
    compatibility/2006">
      <validate schemaType="application/relax-ng-compact-syntax">
        <schema>
          default namespace ="http://schemas.openxmlformats.org/markup-
            compatibility/2006"
          nsList = list { xsd:NCName* }
          qnameList = list { (xsd:QName | xsd:string {pattern = "\i\c*:\*" })*}
          start = element AlternateContent {choice+,fallback?}
          choice = element Choice {attribute Requires { nsList }, text}
          fallback = element Fallback {text}
        </schema>
      </validate>
    </namespace>
    <namespace ns="" match="attributes">
      <attach/>
    </namespace>
    <anyNamespace match="elements attributes">
      <allow/>
    </anyNamespace>
  </rules>
```

The two RELAX NG schemas embedded in the above NVDL script can be rewritten in the analogous XML Schema form.

A.2 Validation Against the Combination of Office Open XML and Extensions

An extension of Office Open XML specified using the mechanisms defined in this Part of ECMA-376 can be captured by an NVDL script that invokes the Office Open XML schema and schemas for the extension.

The following schema allows two extensions. They have the namespaces http://www.example.com/myExtensionWithFallback and http://www.example.com/myExtensionWithoutFallback. The first extension is accompanied with a parent AlternateContent element and a sibling Fallback element, while the second one can appear anywhere in the document without AlternateContent or Fallback elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0" startMode="top">
    <mode name="top">
```

```
    <namespace
      ns="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
        <validate schema="wml.xsd" useMode="nested"/>
    </namespace>
</mode>
<mode name="nested">
    <namespace match="attributes elements"
        ns="http://schemas.openxmlformats.org/drawingml/2006/*">
        <attach/>
    </namespace>
    <namespace match="attributes elements"
        ns="http://schemas.openxmlformats.org/officeDocument/2006/*">
        <attach/>
    </namespace>
    <namespace match="attributes elements"
      ns="http://schemas.openxmlformats.org/package/2006/*">
        <attach/>
    </namespace>
    <namespace match="attributes elements"
        ns="http://schemas.openxmlformats.org/presentationml/2006/main">
        <attach/>
    </namespace>
    <namespace match="attributes elements"
        ns="http://schemas.openxmlformats.org/schemaLibrary/2006/main">
        <attach/>
    </namespace>
    <namespace match="attributes elements"
        ns="http://schemas.openxmlformats.org/spreadsheetml/2006/7/main">
        <attach/>
    </namespace>
    <namespace match="attributes elements"
      ns="urn:schemas-microsoft-com:*">
        <attach/>
    </namespace>
    <namespace match="attributes"
        ns="http://schemas.openxmlformats.org/markup-compatibility/2006">
        <validate schemaType="application/relax-ng-compact-syntax">
            <schema>
                namespace mc =
                "http://schemas.openxmlformats.org/markup-
                  compatibility/2006"
                nsList = list { xsd:NCName* }
```

```
                qnameList = list { (xsd:QName | xsd:string {pattern =
                  "\i\c*:\*" })*}
                start = element * {
                  attribute mc:Ignorable { nsList }?,
                  attribute mc:ProcessContent { qnameList }?,
                  attribute mc:PreserveElements { qnameList }?,
                  attribute mc:PreserveAttributes { qnameList }?,
                  attribute mc:MustUnderstand { nsList }?
                }
              </schema>
            </validate>
          </namespace>
          <namespace match="elements"
            ns="http://schemas.openxmlformats.org/markup-compatibility/2006">
              <validate schemaType="application/relax-ng-compact-syntax">
                  <schema>
                      default namespace =
                      "http://schemas.openxmlformats.org/markup-
                        compatibility/2006"
                      nsList = list { xsd:NCName* }
                      qnameList = list { (xsd:QName | xsd:string {pattern =
                        "\i\c*:\*" })*}
                      start = element AlternateContent {choice,fallback}
                      choice = element Choice {attribute Requires { nsList },
                        text}
                      fallback = element Fallback {text}
                  </schema>
                  <mode>
                      <anyNamespace>
                          <allow/>
                      </anyNamespace>
                  </mode>
                  <context path="Choice">
                      <mode>
                          <namespace
                            ns="http://www.example.com/myExtenstionWithFallback">
                              <validate schema="myExtensionWithFallback.rng">
                                  <mode>
                                      <anyNamespace>
                                          <attach/>
                                      </anyNamespace>
                                  </mode>
                              </validate>
```

```
                    </namespace>
                </mode>
            </context>
        </validate>
        <unwrap>
            <mode>
                <anyNamespace>
                    <allow/>
                </anyNamespace>
            </mode>
            <context path="Fallback">
                <mode>
                    <anyNamespace>
                        <attach/>
                    </anyNamespace>
                </mode>
            </context>
        </unwrap>
    </namespace>
    <namespace ns="http://www.example.com/myExtensionWithoutFallback">
        <validate schema="myExtensionWithoutFallback.rng">
            <mode>
                <anyNamespace>
                    <attach/>
                </anyNamespace>
            </mode>
        </validate>
    </namespace>
  </mode>
</rules>
```

End of informative text.

Annex B. (informative) Index

This annex is informative.

alternate content **4**, 10
AlternateContent 13, **21**
attribute
  compatibility-rule **13**
  Ignorable ..... See Ignorable
  MustUnderstand ..... See MustUnderstand
  PreserveAttributes ..... See PreserveAttributes
  PreserveElements ..... See PreserveElements
  ProcessContent ..... See ProcessContent
Choice 13, **22**
compatibility-rule attribute **4**
conformance class 2
element
  alternate content **21**
  AlternateContent ..... See AlternateContent
  Choice ..... See Choice
  Fallback ..... See Fallback
extension element **29**
Fallback 13, **23**
IEC.. See International Electrotechnical Commission
Ignorable 12, **14**
ignore **4**
informative text **2**
International Electrotechnical Commission 7
International Organization for Standardization 7
island 29
ISO ..... See International Organization for Standardization
ISO/IEC 10646 3
markup consumer **4**
markup document **4**
markup editor **4**
markup preprocessor **4**, 11
markup producer **4**
markup specification **4**
MustUnderstand 13, **20**
name
  expanded 4
  local 4
  prefixed 4
  qualified 4
  unprefixed 4
namespace
  current **31**
  ignorable **4**
  obsolete **31**
  understood **5**
namespace declaration 4
namespace name 4
namespace subsumption **26**
normative text **2**
Office Open XML ..... **vi**
preserve **5**, 10, 18
PreserveAttributes 13, **19**
PreserveElements 13, 18, **19**
ProcessContent 12, **16**
qualified attribute name **5**
qualified element name **5**
recognize **5**
subsume 11
subsumption **26**
W3C ..... See World Wide Web Consortium
World Wide Web Consortium 7

End of informative text.