INTERNATIONAL STANDARD

ISO/IEC 29500-3

Fourth edition
2015-07-01

Information technology — Document description and processing languages — Office Open XML File Formats —

Part 3: Markup Compatibility and Extensibility

Technologies de l'information — Description des documents et langages de traitement — Formats de fichier "Office Open XML" —

Partie 3: Compatibilité et extensibilité du balisage

Reference number
ISO/IEC 29500-3:2015(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29500-3 was prepared by ISO/IEC JTC 1, Information technology, Subcommittee SC 34, Document description and processing languages.

This fourth edition cancels and replaces the third edition (ISO/IEC 29500-3:2012).

The major changes from the previous edition include:

- Specification of the core semantics in one place, and the interactions among semantic constructs and/or the processing model.
- Removal of the specification of namespace subsumption
- Expansion of examples, in particular, by providing output documents

The intended semantics remains the same as long as namespace subsumption is not used.

The major changes in the third edition included:

- Removed all traces of the concept of *markup editor*
- Removed the attributes PreserveAttributes and PreserveElements

There were no major changes in the second edition.

ISO/IEC 29500 consists of the following parts, under the general title *Information technology — Document description and processing languages — Office Open XML File Formats*:

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

Introduction

ISO/IEC 29500 specifies a family of XML schemas, collectively called *Office Open XML*, that define the XML vocabularies for word-processing, spreadsheet, and presentation office documents, as well as the packaging of office 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.

Information technology — Document description and processing languages — Office Open XML File Formats

Part 3:

Markup Compatibility and Extensibility (MCE)

1 Scope

This Part of ISO/IEC 29500 defines a set of conventions for forward compatibility of markup specifications, applicable not only to Office Open XML specifications as described in Parts 1 and 4 of this Standard, but also to other markup specifications. These conventions allow XML documents created by applications of later versions or extensions to be handled by applications of earlier versions.

2 Normative References

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

XML, Tim Bray, Jean Paoli, Eve Maler, C. M. Sperberg-McQueen, and François Yergeau (editors). Extensible Markup Language (XML) 1.0, Fifth Edition. World Wide Web Consortium. 2008. http://www.w3.org/TR/2008/PER-xml-20080205/. [*Note*: Implementations of this Part of ISO/IEC 29500 are not required to support features of XML introduced by the Fifth Edition. *end note*]

XML Base, Marsh, Jonathan. *XML Base*. World Wide Web Consortium. 2009. http://www.w3.org/TR/2009/REC-xmlbase-20090128/

XML Information Set, John Cowan and Richard Tobin (editors). *XML Information Set (Second Edition)*, 4 February 2004. World Wide Web Consortium. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/

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/

3 Terms and Definitions

For the purposes of this standard, the following terms and definitions apply:

3.1
application configuration
set of names of understood namespaces

3.2
application-defined extension element
element defined by a markup specification, the attributes and content of which are not to be processed by an MCE processor

3.3
markup configuration
set of expanded names of application-defined extension elements

3.4
markup specification
XML-based format specification that allows the use of elements and attributes in the MCE namespace

3.5
MCE processor
software used to process XML documents containing MCE elements and attributes

3.6
mismatch
incompatibility between the constraints specified by MCE elements and attributes, and the namespaces specified by an application configuration

3.7
understood namespace
namespace, the elements and attributes of which a consuming application is able to process

4 Notational Conventions

The following typographical conventions are used in ISO/IEC 29500:

1) The first occurrence of a new term is written in italics. [*Example*: The text in ISO/IEC 29500 is divided into *normative* and *informative* categories. *end example*]
2) The tag name of an XML element is written using a distinct style and typeface. [*Example*: The bookmarkStart and bookmarkEnd elements specify … *end example*]
3) The name of an XML attribute is written using a distinct style and typeface. [*Example*: The dropCap attribute specifies … *end example*]
4) The value of an XML attribute is written using a constant-width style. [*Example*: The attribute value of auto specifies … *end example*]

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*]

5 General Description

This clause is informative

This Part of ISO/IEC 29500 is divided into the following subdivisions:

- Front matter (Clauses 1–5);
- Overview (Clause 6);
- Main body (Clauses 7–9);
- 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 ISO/IEC 29500:

- Clauses 1–4, and 7–9

The following make up the informative pieces of this Part of ISO/IEC 29500:

- Foreword
- Introduction
- Clauses 5 and 6
- All annexes
- All notes and examples

End of informative text

6 Overview

This clause is informative

This Part of ISO/IEC 29500 describes a set of XML elements and attributes, called *MCE elements and attributes*, the purpose of which is to enable producing applications to guide consuming applications in their handling of any XML elements and attributes in namespaces not understood by the consuming applications.

MCE elements and attributes are intended to enable producing applications to use features added in new versions or extensions of a markup specification in the production of new documents, which nevertheless remain interoperable with consuming applications that do not understand these features. A producing application includes MCE elements and attributes in documents to indicate to a consuming application how it can adjust the content of the document to exclude those features that are not compatible with the version of the markup specification that it understands, while at the same time allowing consuming applications that do understand these features to make full use of them.

MCE elements and attributes define particular types of compatibility and extension constructs, as summarized below:

- Namespaces can be declared to be ignorable, indicating that all elements and attributes in those namespaces can be disregarded by consuming applications as if they were not present in the input document, enabling graceful degradation of the document functionality. This allows implementations to identify some markup as not core to the document content.
- Elements in ignorable namespaces can be marked for their content to be processed that would otherwise be ignored. This allows producing applications to prevent loss of content nested within an element in an ignorable namespace when processed by consuming applications that do not understand that namespace but do understand the namespace(s) of the nested content.
- Namespaces can be declared that must be understood by consuming applications in order to process the document. This allows producing applications to set minimum compatibility requirements for consuming applications.
- Alternative representations of document content can be specified. This allows producing applications to include content alternatives for consuming applications with differing sets of understood namespaces and corresponding capabilities.
- Application-defined extension elements enable producing applications to introduce additional features scoped to particular elements defined by a markup specification. Consuming applications might preserve application-defined extension elements even if they do not process them in any other way.

Conceptually, a consuming application does not directly process input documents containing MCE elements and attributes but rather uses an MCE processor to produce an output document understood by the consuming application.

See Annex A for examples of input documents using MCE elements and attributes, and output documents that would be produced by an MCE processor.

End of informative text

7 MCE Elements and Attributes

7.1 Introduction

This subclause specifies the syntactic definitions of all MCE elements and attributes, which shall be in the Markup Compatibility namespace:

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

[*Guidance*: External DTD subsets should not specify default values for attributes in the Markup Compatibility namespace, as some non-validating XML processors do not use such default values. *end guidance*]

Attributes within the Markup Compatibility namespace may occur on any XML element, including Markup Compatibility elements.

Elements within the Markup Compatibility namespace shall not contain attributes within the XML namespace `http://www.w3.org/XML/1998/namespace`.

7.2 Ignorable Attribute

An Ignorable attribute shall be an attribute in the Markup Compatibility namespace with local name “Ignorable”. Its value shall be a whitespace-delimited list of zero or more namespace prefixes, optionally having leading and/or trailing whitespace. For each namespace prefix in the list, there shall be an in-scope namespace to which that prefix is bound, and it shall not be the Markup Compatibility namespace. This in-scope namespace is said to be declared as ignorable by this Ignorable attribute.

[*Example*:

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <foo mce:Ignorable="i1"
      xmlns:i1="http://www.example.com/i1"
      xmlns:i2="http://www.example.com/i2">
    <bar mce:Ignorable="i2">…</bar>
  </foo>
</example>
```

The foo element and the bar element each have an Ignorable attribute. The Ignorable attribute of foo specifies the prefix “i1”, which is bound to the in-scope namespace “http://www.example.com/i1”. The Ignorable attribute of bar specifies the prefix “i2”, which is bound to the in-scope namespace “http://www.example.com/i2”. Thus, these namespaces are declared as ignorable. *end example*]

7.3 ProcessContent Attribute

A ProcessContent attribute shall be an attribute in the Markup Compatibility namespace with local name "ProcessContent". Its value shall be a whitespace-delimited list of zero or more tokens, optionally having leading and/or trailing whitespace. Each token shall be a namespace prefix followed by ":" followed either by a local name or by "*". For each token in the list, there shall be an in-scope namespace to which the namespace prefix part of the token is bound. This in-scope namespace shall not be the Markup Compatibility namespace and shall be declared as ignorable by an Ignorable attribute at the same element or an ancestor element. The pair of this in-scope namespace and the local part or "*" in this token is said to be declared as a process-content name pair by this ProcessContent attribute.

If (*n1*, *l1*) is the namespace-name and local-name pair of an element, that element matches a process-content name pair (*n2*, *l2*) if

1) *n1* and *n2* are the same sequence of characters, and
2) Either
    a) *l1* and *l2* are the same sequence of characters, or
    b) *l2* is "*"

[*Example*:

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <foo1 mce:Ignorable="i1"
      mce:ProcessContent="i1:bar1"
      xmlns:i1="http://www.example.com/i1"
      xmlns:i2="http://www.example.com/i2">
    <foo2 mce:Ignorable="i2"
      mce:ProcessContent="i2:*">…</foo2>
    <foo3 mce:ProcessContent="i1:bar2">…</foo3>
  </foo1>
</example>
```

The foo1, foo2, and foo3 elements have ProcessContent attributes. That of the foo1 element has a token "i1:bar1", where "i1" is a namespace prefix bound to an in-scope namespace "http://www.example.com/i1", which is declared as ignorable at this element. That of the foo2 element has a token "i2:*", where i2 is a namespace prefix bound to an in-scope namespace "http://www.example.com/i2", which is declared as ignorable at this element. That of the foo3 element has a token "i1:bar2", where i1 is a namespace prefix bound to an in-scope namespace "http://www.example.com/i1", which is declared as ignorable at the parent foo1 element. *end example*]

7.4 MustUnderstand Attribute

A MustUnderstand attribute shall be an attribute in the Markup Compatibility namespace with local name "MustUnderstand". Its value shall be a whitespace-delimited list of zero or more namespace prefixes

optionally having leading and/or trailing whitespace. For each namespace prefix in the list, there shall be an in-scope namespace name to which that prefix is bound, and this namespace shall not be the Markup Compatibility namespace. This in-scope namespace is said to be declared as a must-understand namespace by this MustUnderstand attribute.

[*Example*:

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:n1="http://www.example.com/n1"
    mce:MustUnderstand="n1">
</example>
```

In this example, the root element has a MustUnderstand attribute. The value contains n1, which is bound to an in-scope namespace name "http://www.example.com/n1". *end example*]

7.5 AlternateContent Element

An AlternateContent element shall be an element in the Markup Compatibility namespace with local name “AlternateContent”. An AlternateContent element shall not have unqualified attributes, but may have qualified attributes. The namespace of each qualified attribute shall be either the Markup Compatibility namespace or a namespace declared as ignorable by the Ignorable attribute of this AlternateContent element or one of its ancestors.

An AlternateContent element shall contain one or more Choice child elements, optionally followed by a single Fallback child element. No other elements in the Markup Compatibility namespace shall appear as child elements. Elements in other namespaces may appear as preceding, intervening, or trailing child elements, but the namespace of such a child element shall be declared as ignorable.

[*Note*: Ignorable elements are allowed as child elements of AlternateContent to allow for future extensions to this construct. If AlternateContent were specified to contain only Choice and Fallback elements from the Markup Compatibility namespace (§7.5), this would prevent the use of other Markup Compatibility elements that would allow extension of AlternateContent in future versions of MCE. Any MCE processor that encounters a child element of AlternateContent that is in the namespace of an intended future extension of MCE will not fail to process the document, provided the namespace of this child element is ignorable, because it will discard all elements in ignorable namespaces that are not understood before making a selection between the remaining Choice and Fallback elements. *end note*]

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

[*Example*:

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:n1="http://www.example.com/n1"
    xmlns:n2="http://www.example.com/n2">
```



```
  <mce:AlternateContent mce:MustUnderstand="n1">
    <mce:Choice Requires="n2">…</mce:Choice>
    <mce:Fallback>…</mce:Fallback>
  </mce:AlternateContent>
</example>
```

In this example, the AlternateContent element has a MustUnderstand attribute and no other attributes. The AlternateContent element has a Choice element followed by a Fallback element. *end example*]

7.6 Choice Element

A Choice element shall be an element in the Markup Compatibility namespace with local name “Choice”. Parent elements of Choice elements shall be AlternateContent elements. A Choice element shall have an unqualified attribute with local name “Requires” and shall have no other unqualified attributes. The value of the Requires attribute shall be a whitespace-delimited list of one or more namespace prefixes, optionally having leading and/or trailing whitespace.

[*Note*: With the exception of empty lists, the syntactical constraints associated with the Requires attribute are the same as those associated with the MustUnderstand attribute. *end note*]

A Choice element may have qualified attributes. The namespace of each qualified attribute shall be either the Markup Compatibility namespace or a namespace declared as ignorable.

[*Example*:

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:i1="http://www.example.com/i1"
    xmlns:n1="http://www.example.com/n1">
  <mce:AlternateContent mce:Ignorable="i1" >
    <mce:Choice Requires="n1" i1:foo="">…</mce:Choice>
  </mce:AlternateContent>
</example>
```

In this example, the Choice element specifies the i1:foo attribute. The namespace of this attribute is declared as ignorable at the parent AlternateContent element. This document is conformant, but would be non-conformant if the i1 namespace was not ignorable. *end example*]

7.7 Fallback Element

A Fallback element shall be an element in the Markup Compatibility namespace with local name “Fallback”. Parent elements of Fallback elements shall be AlternateContent elements.

A Fallback element shall not have unqualified attributes. A Fallback element may have qualified attributes. The namespace of each qualified attribute shall be either the Markup Compatibility namespace or a namespace declared as ignorable.

[*Example*:

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:i1="http://www.example.com/i1"
    xmlns:n1="http://www.example.com/n1">
  <mce:AlternateContent mce:Ignorable="i1" >
    <mce:Choice Requires="n1" >…</mce:Choice>
    <mce:Fallback i1:foo="">…</mce:Fallback>
  </mce:AlternateContent>
</example>
```

In this example, the Fallback element specifies the i1:foo attribute. The namespace of this attribute is declared as ignorable at the parent AlternateContent element. This document is conformant but would be non-conformant if the i1 namespace were not ignorable. *end example*]

8 Application-Defined Extension Elements

A markup specification using MCE elements and attributes might designate one or more elements in the namespaces it defines as *application-defined extension elements*. As described in §9, MCE processing is effectively suspended within the content of these elements and they are passed through to the output document generated by the MCE processor. No elements within the Markup Compatibility namespace shall be designated as application-defined extension elements.

[*Rationale*: This mechanism is intended to, but not limited to, be used by markup specifications to create extensibility points within the markup specification. *end rationale*]

[*Note*: If the markup specification includes a schema, an extension element might be constrained by the schema to occur only in specific markup contexts. *end note*]

[*Note*: The content of an application-defined extension element might contain markup that uses MCE elements and attributes. A consuming application might invoke an MCE processor to process the content of application-defined extension elements contained in an output document constructed by an MCE processor. *end note*]

[*Example*:

```
<example xmlns="http://www.example.com"
    xmlns:n1="http://www.example.com/n1"
    xmlns:unknown="http://www.example.com/unknown">
  <n1:extensionElement>
    <unknown:foo/>
  </n1:extensionElement>
</example>
```

In this example, the extensionElement element contains the unknown:foo element. Suppose that an MCE processor's markup configuration contains the expanded name ("http://www.example.com/n1", "extensionElement") and its application configuration does not contain "http://www.example.com/unknown". Then, the element n1:extensionElement is an application-defined extension element. Although the unknown:foo element does not belong to an understood or ignorable namespace, according to the semantic definitions in §9, the MCE processor does not report the existence of that element as an error. *end example*]

[*Example*:

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:i1="http://www.example.com/i1"
    xmlns:n1="http://www.example.com/n1">
  <extensionElement>
```

```
      <foo1 mce:Ignorable="i1"
          mce:ProcessContent="i1:bar1"
          mce:MustUnderstand="n1">
      </foo1>
      <mce:AlternateContent mce:Ignorable="i1" >
        <mce:Choice Requires="n1" >…</mce:Choice>
        <mce:Fallback i1:foo="">…</mce:Fallback>
      </mce:AlternateContent>
    </extensionElement>
  </example>
```

In this example, MCE elements and attributes occur within the extensionElement element, which is the only child of the root element example. Suppose that an MCE processor is configured to preserve extension elements of an expanded name ("http://www.example.com/n1", "extensionElement"). Then, the MCE processor preserves the extensionElement element. Therefore, MCE elements and attributes within it, `mce:Ignorable="i1"`, `mce:ProcessContent="i1:bar1"`, `mce:MustUnderstand="n1"`, `mce:AlternateContent`, `mce:Choice`, and `mce:Fallback`, appear in the output document. *end example*]

9 Semantic Definitions and Reference Processing Model

9.1 Overview

This clause defines the output document that shall be created by the MCE processor given an input document, markup configuration, and application configuration. The MCE processor is initialized with the markup and application configurations. The markup configuration represents the set of expanded names that are to be treated as those of application-defined extension elements. The application configuration represents the set of namespaces that are to be treated as understood.

This clause further specifies the mismatch conditions among a given input document, markup configuration, and application configuration that shall be signaled by the MCE processor; however, it does not specify exactly when or how such mismatches are to be signaled. If an MCE processor detects a mismatch, it shall signal this mismatch to the consuming application and it may continue normal MCE processing. [*Note*: If a consuming application is able to process elements and attributes that are in no namespace, this must be specified by the application configuration. Implementers are reminded that there is no namespace name for elements and attributes that are in no namespace, and are advised that they should take this into account when designing MCE processors. *end note*]

This clause defines the semantics through the description of an abstract processing model, which has the following three steps:

1) Step 1 defines which elements and attributes are marked as ignored or unwrapped. This definition takes into consideration Ignorable and ProcessContent attributes, but does not take into consideration MustUnderstand attributes or AlternateContent, Choice, or Fallback elements. Elements or attributes inside application-defined extension elements are not marked as ignored or unwrapped.
2) Step 2 defines the semantics of AlternateContent elements. It specifies which Choice or Fallback element of each AlternateContent element is selected. This definition takes into consideration AlternateContent, Choice, and Fallback elements, but does not take into consideration Ignorable, ProcessContent, or MustUnderstand attributes. Choice or Fallback elements inside application-defined extension elements are not marked as selected.
3) Step 3 applies the results from Steps 1 and 2 to construct the output document and also examines MustUnderstand attributes unless they are inside application-defined extension elements.

However, MCE processors are not required to carry out these steps. MCE processors are conformant as long as output documents created and mismatches signaled from given input documents, markup configurations, and application configurations are consistent with those created and signaled by these steps.

[*Note*: Because Markup Compatibility processing is not performed inside application-defined extension elements, an output document might still contain elements and attributes in the Markup Compatibility namespace after Markup Compatibility processing has been applied. *end note*]

If an MCE processor detects that a document is non-conformant, the MCE processor should indicate this non-conformance to the consuming application.

9.2 Step 1: Processing the Ignorable and ProcessContent Attributes

An element shall be marked as ignored if all of the following conditions are satisfied:

1) The namespace of this element is declared as ignorable by an Ignorable attribute of this element or of an ancestor element;
2) The namespace of this element is not included in the given application configuration;
3) This element does not match any process-content name pairs declared by this element or some ancestor; and
4) This element is neither an application-defined extension element nor a descendant of an application-defined extension element.

An attribute shall be marked as ignored if all of the following conditions are satisfied:

1) The namespace of this attribute is declared as ignorable by an Ignorable attribute of the element having this attribute or of an ancestor element;
2) The namespace of this attribute is not included in the given application configuration; and
3) This attribute does not belong to an application-defined extension element or a descendant of an application-defined extension element

An element shall be marked as unwrapped if all the following conditions are satisfied:

1) The namespace of this element is declared as ignorable by an Ignorable attribute of this element or of some ancestor element;
2) The namespace of this element is not included in the given application configuration;
3) This element matches a process-content name pair declared by this element or some ancestor; and
4) This element is neither an application-defined extension element nor a descendant of an application-defined extension element.

An element marked as unwrapped shall not have an xml:base, xml:lang or xml:space attribute.

[*Example*:

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:i1="http://www.example.com/i1"
    mce:Ignorable="i1"
    mce:ProcessContent="i1:bar"
    i1:foo="">
```



```
  <i1:foo><fooChild/></i1:foo>
  <i1:bar><barChild/></i1:bar>
  <i1:baz i1:baz=""><i1:bazChild/></i1:baz>
</example>
```

The namespace "http://www.example.com/i1" is declared as ignorable. A pair ("http://www.example.com/i1", bar) is declared as a process-content name pair. Suppose that a given markup configuration is a singleton containing ("http://www.example.com/i1", baz) and that a given application configuration does not contain "http://www.example.com/i1". Then, the i1:foo attribute and the i1:foo element are marked as ignored, and the i1:bar element is marked as unwrapped. However, the i1:baz element is not marked as either ignored or unwrapped, as it is an application-defined extension element. Likewise, neither the i1:baz attribute nor the i1:bazChild element are marked. Neither fooChild nor barChild are marked as ignored or unwrapped although their parents are marked as ignored or unwrapped. *end example*]

9.3 Step 2: Processing the AlternateContent, Choice and Fallback Elements

A Choice element shall be marked as selected if the following conditions are satisfied:

1) Each of the namespaces specified by the Requires attribute of this element is included in the given application configuration;
2) No preceding sibling Choice element is marked as selected; and
3) The element is not a descendant of an application-defined extension element.

A Fallback element shall be marked as selected if the following conditions are satisfied:

1) No preceding sibling Choice element is marked as selected; and
2) The element is not a descendant of an application-defined extension element.

[*Example*:

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:n1="http://www.example.com/n1"
    xmlns:n2="http://www.example.com/n2"
    xmlns:n3="http://www.example.com/n3">
  <mce:AlternateContent>
    <mce:Choice Requires="n1 n2">                    <!-- Choice #1 -->
      <mce:AlternateContent>
        <mce:Choice Requires="n3">...</mce:Choice>     <!-- Choice #1-1 -->
        <mce:Fallback>...</mce:Fallback>               <!-- Fallback #1-1 -->
      </mce:AlternateContent>
    </mce:Choice>
    <mce:Choice Requires="n1">                       <!-- Choice #2 -->
      <mce:AlternateContent>
```

```
        <mce:Choice Requires="n3">...</mce:Choice>       <!-- Choice #2-1 -->
        <mce:Fallback>...</mce:Fallback>                 <!-- Fallback #2-1 -->
      </mce:AlternateContent>
    </mce:Choice>
    <mce:Fallback>...</mce:Fallback>                  <!-- Fallback #1 -->
  </mce:AlternateContent>
</example>
```

The child of the root element is an AlternateContent element. Suppose that an application configuration contains three namespaces, “http://www.example.com/n1”, “http://www.example.com/n2”, and http://www.example.com/n3. Then Choice #1, Choice #1-1, and Choice #2-1 are marked as selected, and Choice #2, Fallback #1, Fallback #1-1, and Fallback #2-1 are not. Note that Choice #2-1, which is marked as selected, appears under Choice #2, which is not.

Suppose that an application configuration contains two namespaces, “http://www.example.com/n1” and “http://www.example.com/n2”, but not “http://www.example.com/n3”. Then, Choice #1, Fallback #1-1, and Fallback #2-1 are marked as selected. *end example*]

9.4 Step 3: Processing the MustUnderstand Attribute and Creating the Output Document

The output document shall be constructed by modifying the input document with the following procedure, beginning with the root element.

For each element in the input document, if the element is:

1) Marked as ignored:

    a) Remove this element and its attributes and contents.

2) Marked as unwrapped:

    a) Examine each MustUnderstand attribute of this element. If any of the namespaces declared by this attribute are not in the application configuration, signal a mismatch.
    b) Replace this element with the content of this element. [*Note*: The attributes of this element will be lost. *end note*.]
    c) Recursively apply the Step 3 procedure to each child of this element.

3) An AlternateContent element that is not a descendant of an application-defined extension element:

    a) If any child element of this AlternateContent element is neither a Choice nor a Fallback element, and is not marked as ignored, signal a mismatch.
    b) If none of the child Choice or Fallback elements is marked as selected, remove this AlternateContent element and its contents. If a child Choice or Fallback element is marked as selected, replace this AlternateContent element with the content of the Choice or Fallback element marked as selected.

c) Examine each MustUnderstand attribute of this AlternateContent element and of the Choice or Fallback element marked as selected, if any. If any of the namespaces declared by these attributes are not in the application configuration, signal a mismatch.
d) Recursively apply the Step 3 procedure to each child element of the Choice or Fallback element marked as selected, if any.

4) An application-defined extension element:

a) Make no changes to the element or its contents, and include them in the output document.

5) Otherwise:

a) Remove Ignorable, ProcessContent, and MustUnderstand attributes as well as attributes marked as ignored.
b) Recursively apply the Step 3 procedure to each child of this element.

[*Note*: The content of a selected Choice or Fallback element does not appear in the output document if any ancestor of this element is ignored or any ancestor Choice or Fallback element is not selected. *end note*]

[*Note*: Output documents might contain attributes or elements in namespaces that are not contained in the application configuration. *end note*]

[*Note*: With the exception of those within application-defined extension elements, elements and attributes in the Markup Compatibility namespace do not appear in the output document. *end note*]

[*Example*:

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:foo="http://www.example.com/foo"
    xmlns:bar="http://www.example.com/bar"
    mce:Ignorable="foo bar"
    mce:ProcessContent="foo:unwrapped">
  <mce:AlternateContent>
    <mce:Choice Requires="foo">            <!-- Choice #1 -->
      <foo:foo/>                           <!-- Foo #1 -->
      <bar:bar>                            <!-- Bar #1 -->
        <mce:AlternateContent>
          <mce:Choice Requires="bar">      <!-- Choice #1-1 -->
            <Choice1-1/>
          </mce:Choice>
          <mce:Fallback>                   <!-- Fallback #1-1 -->
            <Fallback1-1/>
          </mce:Fallback>
        </mce:AlternateContent>
      </bar:bar>
```

```
      </mce:Choice>
      <mce:Choice Requires="bar">             <!-- Choice #2 -->
        <bar:bar/>                            <!-- Bar #2 -->
        <foo:unwrapped>                       <!-- Foo #2 -->
          <mce:AlternateContent>
            <mce:Choice Requires="foo">       <!-- Choice #2-1 -->
              <Choice2-1/>
            </mce:Choice>
            <mce:Fallback>        <!-- Fallback #2-1 -->
              <Fallback2-1/>
            </mce:Fallback>
          </mce:AlternateContent>
        </foo:unwrapped>
      </mce:Choice>
    </mce:AlternateContent>
  </example>
```

Suppose that an application configuration contains the namespace http://www.example.com/foo. In Step 1, Bar #1 and Bar #2 are marked as ignored. In Step 2, Choice #1, Fallback #1-1, and Choice #2-1 are marked as selected. However, in Step 3, the content of Fallback #1-1 is discarded, since Bar #1 is marked as ignored. Likewise, the content of Choice #2-1 is discarded, since Choice #2 is not marked as selected. In Step 3, Bar #1 and its contents, including Fallback #1-1, are discarded. Thus, the following output document is constructed.

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:foo="http://www.example.com/foo"
    xmlns:bar="http://www.example.com/bar">
  <foo:foo/>
</example>
```

Suppose that an application configuration contains the namespace http://www.example.com/bar. In Step 1, Foo #1 is marked as ignored and Foo #2 is marked as unwrapped. In Step 2, Choice #1-1, Choice #2, and Fallback #2-1 are marked as selected. However, in Step 3, the content of Choice #1-1 is discarded, since Choice #1 is not marked as selected. Since Foo #2 is marked as unwrapped, the content of Fallback #2-1 is not discarded. Thus, the following output document is constructed.

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:foo="http://www.example.com/foo"
    xmlns:bar="http://www.example.com/bar">
  <bar:bar/>
  <Fallback2-1/>
</example>
```


Suppose that an application configuration contains two namespaces, http://www.example.com/foo and http://www.example.com/bar. In Step 1, no elements or attributes are marked as ignored or unwrapped. In Step 2, Choice #1, Choice #1-1, and Choice #2-1 are marked as selected. However, in Step 3, the content of Choice #2-1 is discarded, since Choice #2 is not marked as selected. Thus, the following output document is constructed.

```
<example xmlns="http://www.example.com"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:foo="http://www.example.com/foo"
    xmlns:bar="http://www.example.com/bar">
  <foo:foo/>
  <bar:bar>
    <Choice1-1/>
  </bar:bar>
</example>
```

end example]

Annex A (informative) Examples

This annex is informative.

A.1 Syntactic Examples

A.1.1 General

This clause provides examples further demonstrating the syntax of MCE elements and attributes.

A.1.2 Ignorable Attribute: Multiple Prefixes Bound to a Namespace

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <foo mce:Ignorable="i1"
      xmlns:i1="http://www.example.com/i1"
      xmlns:i2="http://www.example.com/i2">
    <bar mce:Ignorable="i2">…</bar>
    <bar mce:Ignorable="i1 i2">…</bar>
    <bar mce:Ignorable="i1alias i2"
        xmlns:i1alias="http://www.example.com/i1">…</bar>
  </foo>
</example>
```

The Ignorable attribute of the first bar element declares the namespace “http://www.example.com/i2” as ignorable. The Ignorable attribute of the second bar element declares both “http://www.example.com/i1” and “http://www.example.com/i2” as ignorable, but the former is already declared by the Ignorable attribute of the parent foo element. The Ignorable attribute of the third bar element also declares these two namespaces as ignorable, although the namespace prefix is i1alias rather than i1. Therefore, although the lexical values are different, these three Ignorable attributes cause the content of each bar element to be treated equally as far as MCE processing is concerned.

A.1.3 Ignorable Attribute: Non-conformant Use

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <foo1 mce:Ignorable="i1">
    <foo2 xmlns:i1="http://www.example.com/i1">…</foo2>
  </foo1>
```



```
  <foo3 mce:Ignorable="i2">…</foo3>
  <foo4 xmlns:i2="http://www.example.com/i2"/>
</example>
```

This document is non-conformant for two reasons: First, the foo1 element has an Ignorable attribute, but the value i1 is not bound to an in-scope namespace. Second, the foo3 element also has an Ignorable attribute, but the value i2 is not bound to an in-scope namespace either.

A.1.4 ProcessContent Attribute: Multiple Prefixes Bound to a Namespace

```
<example xmlns=http://www.example.com
    xmlns:mce=http://schemas.openxmlformats.org/markup-compatibility/2006
    xmlns:n1="http://www.example.com/n1"
    xmlns:n1alias=http://www.example.com/n1
    mc:Ignorable="n1alias"
    mc:ProcessContent="n1:foo" >
  <n1alias:foo>
    <bar/>
  </n1alias:foo>
</example>
```

The namespace "http://www.example.com/n1" is declared as ignorable. The pair of this namespace and a local name foo is declared as a process-content name pair by the ProcessContent attribute. The n1alias:foo element matches this pair, because n1 and n1alias share the same namespace name.

A.1.5 ProcessContent Attribute: Non-conformant Use

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <foo1 xmlns:i2="http://www.example.com/i2">
    <foo2 mce:ProcessContent="i2:*">…</foo2>
  </foo1>
</example>
```

The foo2 element has a ProcessContent attribute. The value is a token "i2:*", where i2 is a namespace prefix bound to an in-scope namespace "http://www.example.com/i2". However, this namespace is not declared as ignorable. As such, this example is non-conformant.

A.1.6 MustUnderstand Attribute: Non-conformant Use

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:n1="http://www.example.com/n1">
  <foo mce:MustUnderstand="n1 n2"/>
</example>
```

The MustUnderstand attribute of the element foo contains n1 and n2. Although n1 is bound to an in-scope namespace name "http://www.example.com/n1", n2 is not. As such, this document is non-conformant.

A.1.7 AlternateContent Element: Future Extensibility

```
<example xmlns="http://www.example.com/"
    xmlns:mce="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:mce2="http://www.example.com/markup-compatibility/v2"
    xmlns:n1="http://www.example.com/n1">
  <mce:AlternateContent mce:Ignorable="mce2" mce2:foo="">
    <mce2:NewChoice … />…</mce2:NewChoice>
    <mce:Choice Requires="n1">…</mce:Choice>
    <mce:Fallback>…</mce:Fallback>
  </mce:AlternateContent>
</example>
```

The AlternateContent element has an Ignorable attribute. The AlternateContent element has as children a Choice element and a Fallback element from the same namespace and a NewChoice element from another namespace. Because the namespace of the NewChoice element, which might be part of a hypothetical future version of Markup Compatibility and Extensibility, is declared as ignorable, this document is conformant. If the namespace of the NewChoice element were not declared as Ignorable, the document would be non-conformant.

A.2 Semantic Examples

A.2.1 General

This clause provides examples of processing input documents in a markup specification that supports the use of MCE and the corresponding output documents generated by an MCE processor, given different markup configurations and application configurations.

A.2.2 Ignorable Attribute

This example shows how to use the Ignorable attribute to define ignorable namespaces and how MCE processors with different application configurations process the example input document.

Input document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2"
    xmlns:v3="http://www.example.com/Circles/v3"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="v2 v3">
  <Circle Center="0,0" Radius="20" Color="Blue" v2:Opacity="0.5"
      v3:Luminance="13"/>
<Circles>
```


Three namespaces in this document, “http://www.example.com/Circles/v1”, “http://www.example.com/Circles/v2”, and “http://www.example.com/Circles/v3” capture three versions of a markup specification. Version 1 introduces Circle elements of the namespace for version 1. Version 2 introduces the Opacity attribute of the namespace for version 2. Version 3 introduces the Luminance attribute of the namespace for version 3. In this document, both the namespace for version 2 and that for version 3 are declared as ignorable.

First, suppose that the application configuration contains the namespaces for versions 1, 2, and 3. Then, the output document contains the Opacity and Luminance attributes.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2"
    xmlns:v3="http://www.example.com/Circles/v3">
  <Circle Center="0,0" Radius="20" Color="Blue" v2:Opacity="0.5"
      v3:Luminance="13"/>
</Circles>
```

Second, suppose that the application configuration contains the namespaces for versions 1 and 2 but not the one for version 3. Then, the output document contains the Opacity attribute but does not contain the Luminance attribute.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2">
  <Circle Center="0,0" Radius="20" Color="Blue" v2:Opacity="0.5"/>
</Circles>
```

Third, suppose that the application configuration contains the namespace for version 1 but not those for versions 2 or 3. In this case, the output document contains neither the Opacity attribute nor the Luminance attribute.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1">
  <Circle Center="0,0" Radius="20" Color="Blue"/>
</Circles>
```

A.2.3 Ignorable and ProcessContent Attributes

This example shows how to use the ProcessContent attribute to process child elements within ignorable elements and how MCE processors with different application configurations process the example input document.

Input document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="v2"
    mc:ProcessContent="v2:Blink">
  <v2:Watermark Opacity="v0.1">
    <Circle Center="0,0" Radius="20" Color="Blue"/>
  </v2:Watermark>
  <v2:Blink>
    <Circle Center="13,0" Radius="20" Color="Yellow"/>
  </v2:Blink>
</Circles>
```

Two namespaces in this document, “http://www.example.com/Circles/v1” and “http://www.example.com/Circles/v2”, represent two versions of a markup specification. Version 1 introduces Circles and Circle elements of the namespace for version 1. Version 2 introduces Watermark and Blink elements of the namespace for version 2. The namespace for version 2 is declared as ignorable and the Blink element matches a process-content name pair (“http://www.example.com/Circles/v2”, Blink) declared at the root element.

First, suppose that an application configuration contains the namespaces for Versions 1 and 2. Then, the output document contains all elements in the input document.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2">
  <v2:Watermark Opacity="v0.1">
    <Circle Center="0,0" Radius="20" Color="Blue"/>
  </v2:Watermark>
  <v2:Blink>
    <Circle Center="13,0" Radius="20" Color="Yellow"/>
  </v2:Blink>
</Circles>
```

Second, suppose that an application configuration contains the namespace for version 1 but not the one for version 2. Then, the output document does not contain the Watermark and Blink elements, which are of the namespace for version 2. However, the content of the Blink element is retained, since this element matches a declared process-content name pair.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1">
```



```
  <Circle Center="13,0" Radius="20" Color="Yellow"/>
</Circles>
```

A.2.4 Non-Ignorable and Non-Understood Namespace

This example shows the basic handling of namespaces by MCE processors with different configurations.

Input document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <Circle Center="0,0" Radius="20" Color="Blue" v2:Opacity="0.5" />
</Circles>
```

Two namespaces in this document, “http://www.example.com/Circles/v1” and “http://www.example.com/Circles/v2”, represent two versions of a markup specification. Version 1 introduces Circles and Circle elements of the namespace for version 1. Version 2 introduces an Opacity attribute of the namespace for version 2.

First, suppose that an application configuration contains the namespaces for versions 1 and 2. Then, the output document is identical to the input document, with the possible exception of omitting the declaration of the Markup Compatibility namespace.

Second, suppose that an application configuration contains the namespace for version 1 but not the one for version 2. Then, the MCE processor will report a mismatch when the Opacity attribute is examined in Step 3 (§9.4).

A.2.5 MustUnderstand Attribute

This example shows how to use the MustUnderstand attribute to require handling of namespaces and how MCE processors with different application configurations process the example input document.

Input document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:MustUnderstand="v2">
  <Circle Center="0,0" Radius="20" Color="Blue" v2:Opacity="0.5" />
</Circles>
```

This document is similar to the previous example. The only difference is the addition of the MustUnderstand attribute at the root element.

The MCE processor behaves the same, except that a mismatch is signaled if the application configuration does not contain the namespace "http://www.example.com/Circles/v2".

A.2.6 AlternateContent Element

This example shows how to use AlternateContent, Choice and Fallback elements to specify alternate representations of content and how MCE processors with different application configurations process the example input document.

Input document:

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

Three namespaces in this document, "http://www.example.com/Circles/v1", "http://www.example.com/Circles/v2", and "http://www.example.com/Circles/v3" capture three versions of a markup specification. Version 1 introduces LuminanceFilter and Circle elements of the namespace for version 1. Version 2 introduces the Opacity attribute of the namespace for version 2. Version 3 introduces Circle elements of the namespace for version 3. Both the namespace for version 2 and that for version 3 are declared as ignorable.

First, suppose that the application configuration contains the namespaces for versions 1, 2, and 3. Then, since the Choice element is selected, the output document contains Circle elements of the namespace for version 3 but does not contain the LuminanceFilter element.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2"
    xmlns:v3="http://www.example.com/Circles/v3"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="v2 v3">
  <v3:Circle Center="0,0" Radius="20" Color="Blue" Opacity="0.5"
```



```
      Luminance="13"/>
</Circles>
```

Second, suppose that the application configuration contains the namespaces for versions 1 and 2 but not the one for version 3. Then, since the Fallback element is selected, the output document contains the LuminanceFilter and Circle elements of the namespace for version 1 but does not contain Circle elements of that for version 3. The Opacity attribute is not removed, since the application configuration contains the namespace for version 2.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1"
    xmlns:v2="http://www.example.com/Circles/v2">
    <LuminanceFilter Luminance="13">
      <Circle Center="0,0" Radius="20" Color="Blue" v2:Opacity="0.5"/>
    </LuminanceFilter>
</Circles>
```

Third, suppose that the application configuration contains the namespace for version 1 but not those for versions 2 or 3. Then, since the Fallback element is selected, the output document contains the LuminanceFilter element and Circle elements of the namespace for version 1 but does not contain Circle elements of that for version 3. Furthermore, since the application configuration does not contain the namespace for version 2, the Opacity attribute is removed.

Output document:

```
<Circles xmlns="http://www.example.com/Circles/v1">
    <LuminanceFilter Luminance="13">
      <Circle Center="0,0" Radius="20" Color="Blue"/>
    </LuminanceFilter>
</Circles>
```

A.2.7 Ignorable Content Inside Application-Defined Extension Elements

This example shows how to use ignorable content in application-defined extension elements in order to further extend an existing extension.

ChrisOffice v1 is a hypothetical implementation of ISO/IEC 29500 that allows sound effects to be applied to existing SpreadsheetML conditional formatting. This data is stored inside application-defined extension elements in order that other ISO/IEC 29500-conformant applications can round-trip that data, and in SpreadsheetML, an application-defined extension element (extLst) is already defined under the conditionalFormattingElements element.

ChrisOffice v2 adds the ability to use video as well as audio. To allow ChrisOffice v1 to understand everything except the video, a different namespace must be used to avoid ChrisOffice discovering unknown content in

understood namespaces. In this example, ChrisOffice v1 doesn't have its own extension elements within extension elements, so the extra content is ignorable in order that ChrisOffice v1 discards it upon load.

Input document:

```
…
<conditionalFormattingElements>
  <extLst>
    <ext uri="myurl" xmlns:co1="http://chrisoffice/v1">
      <co1:soundeffect mc:Ignorable="co2" xmlns:co2="http://chrisoffice/v2">
        <co1:sourceFile>moo.mp3</co1:sourceFile>
        <co2:sourceVideo>cow.mpg</co2:sourceFile>
      </co1:soundeffect>
    </ext>
  </extLst>
</conditionalFormattingElements>
…
```

ChrisOffice v1 will discard the video when it reads the file. Because processing of MCE constructs is not permitted inside application-defined extension elements, applications that do not understand the original sound effect construct will not needlessly throw away the entire extension element including the new video content. Because the extension element is understood by ChrisOffice v1, and the format of that extension element is known to contain further MCE constructs, ChrisOffice will subsequently invoke an MCE processor to process the content of that extension element and discard the video when it reads the file since the co2 namespace is declared as ignorable.

Output document (ChrisOffice v1):

```
…
<conditionalFormattingElements>
  <extLst>
    <ext uri="myurl" xmlns:co1="http://chrisoffice/v1">
      <co1:soundeffect mc:Ignorable="co2" xmlns:co2="http://chrisoffice/v2">
        <co1:sourceFile>moo.mp3</co1:sourceFile>
      </co1:soundeffect>
    </ext>
  </extLst>
</conditionalFormattingElements>
…
```

End of informative text.

Annex B
(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.

A markup document can satisfy requirements of this Part 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 ISO/IEC 29500.

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: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.

End of informative text.

Bibliography

In addition to the Normative References, the following are useful references for implementers and users of this International Standard:

ISO/IEC 19757-2:2008, *Information technology — Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*

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

ISO/IEC 29500-1:2012, *Information technology — Document description and processing languages — Office Open XML File Formats, Part 1: Fundamentals and Markup Language Reference.*

ISO/IEC 29500-4:2012, *Information technology — Document description and processing languages — Office Open XML File Formats, Part 4: Transitional Migration Features.*

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/

ICS 35.060; 35.240.30

Price based on 33 pages