Parts 1 & 4 Edits Relating to the Part 3 Revision

Francis Cave (francis@franciscave.com)

Rex Jaeschke (rex@RexJaeschke.com)

2013-08-16

During the revision of Part 3 (MCE), it was noticed that various clauses and subclauses of Parts 1 and 4 made reference to features defined in Part 3. The intent of this paper is to identify such references and to point out those which will, or might, need editing based on the changes being proposed for Part 3.

We started by searching Part 1 for occurrences of 'AlternateContent', 'Ignorable', 'PreserveElements' and 'PreserveAttributes'. We also searched for occurrences of 'extLst', in case that threw up any additional references to MCE. (Most of the occurrences of 'ignorable' are not relevant.)

The proposed changes to IS 29500-1:2012 and IS 29500-4:2012, and commentary, follow below in ascending order by Part and subclause number. Direct quotes from those Parts are shaded in grey, like this.

Part 1, §2.1, "Document Conformance", p. 2

A document of conformance class Office Open XML Strict shall be a package of conformance class OPC, as specified in ISO/IEC 29500-2, for which all the following shall hold:

- …
- For each OPC Part of the document of the types listed in §11.3, §12.3, §13.3, §14.2 or §15.2, all the following shall hold:

  i. ~~The part is of conformance class MCE, as specified in ISO/IEC 29500-3~~The Part may contain markup in the Markup Compatibility namespace as specified in ISO/IEC 29500-3
  ii. After the removal of any extensions ~~using the mechanisms~~by an MCE processor as specified in ISO/IEC 29500-3, the part is valid against the strict W3C XML Schema (Appendix A)

Part 1, §10, "Markup Compatibility and Extensibility", p. 28

Office Open XML documents are designed to allow for innovation by extending their capabilities, by using (where allowed) the Application-Defined Extension Elements extLst and ext specified by this Part of ISO/IEC 29500 or by using the Markup Compatibility and Extensibility features specified by ISO/IEC 29500-3:20xx ~~via a scheme defined by Part 3~~. This subclause contains information regarding Office Open XML's use of the Markup Compatibility ~~constructs~~ and Extensibility features, in particular in combination with extension elements.

All the features of ISO/IEC 29500-3:20xx are supported by this Part of ISO/IEC 29500. An MCE processor shall be configured so as not to process the content of any extension element specified by this Part of ISO/IEC 29500,

i.e., specifically not to process any instance of an element or attribute in the MCE namespace for which an extension element is an ancestor element.

The substance of the existing Clause is in §10.1.1 and §10.1.2. We are proposing to remove 'PreserveElements' and 'PreserveAttributes'from Part 3, in which case §10.1.1 can be removed. It seems likely that the revisions to Part 3 will remove anything normative about extension elements, in which case §10.1.2 can also be removed. The way in which 'extLst' interacts with MCE needs to be fully defined in Part 1. This in effect means that §10.1 is redundant, because there are no remaining constraints on Office Open XML's use of the revised (20xx) edition of MCE. It is therefore proposed to modify the introductory paragraph as proposed above and add the second paragraph to specify how MCE processors should be configured to handle extension elements.

~~**Part 1, §10.1, "Constraints on Office Open XML's Use of Markup Compatibility and Extensibility", p. 28**~~

~~While the Markup Compatibility and Extensibility specification is designed for and used by Office Open XML documents, it could also be used to support a much broader range of applications. As a result, the use of some Markup Compatibility and Extensibility features is restricted within Office Open XML documents. These additional requirements are discussed in the following subordinate subclauses. Unless explicitly specified below, all normative requirements of the Markup Compatibility and Extensibility specification apply to Office Open XML documents.~~

~~**Part 1, §10.1.1, "PreserveElements and PreserveAttributes", p. 28**~~

~~The PreserveElements and PreserveAttributes elements, as defined in Part 3, allow a markup language to specify the conditions under which extensions should be round-tripped, even when their contents are edited. Within the context of the markup languages explicitly defined by ISO/IEC 29500, no such conditions are specified, and therefore applications are not obliged to support these hints at any point in an Office Open XML document. Instead, the well-defined extensibility constructs defined below should be used.~~

~~All other constructs defined in Part 3 shall be supported.~~

~~**Part 1, §10.1.2, "Office Open XML Native Extensibility Constructs", p. 28**~~

~~Clause 12 of Part 3 specifies the ability for a markup language to define additional constructs for extensibility of a specific markup language. Within the context of Office Open XML documents, the extLst element(s) defined in individual markup languages shall allow the round-tripping of all unknown content regardless of the state of the PreserveElements and PreserveAttributes elements. See the reference material in §17–23 for additional information on the XML elements that allow the use of the extLst construct.~~

Part 1, §17.17.3, "Roundtripping Alternate Content", p. 1297–1298

~~Office Open XML defines a mechanism for the storage of content which is not defined by ISO/IEC 29500, for example extensions developed by future software applications which leverage the Office Open XML formats. This mechanism allows for the storage of a series of alternative representations of content, of which the consuming application should use the first alternative whose requirements are met.~~

[*Example*: Consider an application which creates a new paragraph property intended to make the colors of its text change colors randomly when it is displayed. This functionality is not defined in ISO/IEC 29500, and so the application might choose to create an alternative representation setting a different manual color on each character for clients which do not understand this extension using an AlternateContent block as follows:

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

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

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

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

~~possible~~. This behavior is therefore application-defined. For further discussion of alternate content blocks see §L.1.18.4.

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

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

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

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

~~The state of the alternate choices (preserved or not) is dependent on the application hosting the file. Preserving the content involves storing each non-taken choice while the file is being edited, and writing out the file with an AlternateContent block when it is resaved. *end example*~~]

The normative definition for AlternateContent is in Part 3, so there is no need to re-specify any part of that here. As such, almost all of this subclause content is tutorial in nature. As it happens, all but one sentence of this subclause exists, verbatim, in the informative subclause Part 1, §L.1.18.4, “Roundtripping Alternate Content”. As such, that text can be removed from §17.17.3.

The only text from §17.17.3 that is not in §L.1.18.4 is the underlined sentence in the following: “However, WordprocessingML does not explicitly define a set of locations where applications shall attempt to store and roundtrip all non-taken choices whenever possible. This behavior is therefore application-defined.” Right now,

that text is normative, and by declaring this behavior to be application-defined, each conforming implementation is required to document that behavior. If this statement were removed or moved to informative text (which would require the removal of the behavior mention), the behavior would become unspecified, which does not require anything of a conforming implementation. So, if the intent is to require such an implementation to document such behavior, this statement needs to stay normative. However, this whole clause could be reduced to contain a single paragraph containing (essentially) this normative text, rather than repeat material that can more properly be found in §L.1.18.4.

Part 1, §18.2.7, "ext (Extension)", p. 1555

Each extension within an extension list shall be contained within an ext element. Extensions shall be versioned by namespace, using the uri attribute, and shall be allowed to appear in any order within the extension list. Any number of extensions shall be allowed within an extension list.

When extension lists are processed, a consumer might understand some extensions, and might not understand other extensions. The preservation model for extensions is that unprocessed extensions shall always be preserved (when consuming) and written out (when producing) in whole, ~~as long as the underlying schema extended by the extension list remains~~as long as there is not some ancestor element of the extension list that is discarded as a result of either MCE processing (when consuming). [*Example*: If a spreadsheetML sheet contains several extensions within an extension list, and through runtime processing that sheet is removed from the workbook, then the extensions associated with that sheet must not be written out when producing the resulting markup document. *end example*]

Markup namespaces within extensions shall not be required to be listed in the Ignorable Compatibility-Rule attribute~~, nor shall these namespaces be required to be listed in the PreserveElements and PreserveAttributes Compatibility-Rule attributes~~. [*Note*: See ~~Part 3~~§10 for additional discussion on Application-Defined Extension Elements and processing rules. *end note*]

…

Upon encountering extensions, a processing consumer shall determine whether it knows how to process extensions using the value of the uri. If the consumer knows how to process such an extension, the markup contained within that extension is processed. Otherwise, the extension content shall be preserved so long as the ~~underlying~~ structure ~~being extended by~~that contains the extLst has not been removed.

…

References to MCE features that have been removed from Part 3 should be removed here. The phrases "underlying schema" and "underlying structure" are unclear. The highlighted example is poor, because the normative text concerns processing when consuming (i.e. reading/opening) or producing (i.e. writing/saving) a document, and not what might or might not happen between the two. This standard cannot usefully comment upon what happens between consumption and production, as this is implementation-dependent. An implementation might (reasonably or unreasonably) give the user the option of removing extension elements before saving the document. The example therefore needs to be improved.

Part 1, §18.2.10, "extLst (Future Feature Data Storage Area)", p. 1557–1558

…

[*Note*: Allowing markup specification extensions and private markup extensions within an extension list does not violate interoperability because the rules articulated within §10, §18.2.7 and Part 3, §12 describe how markup producers and consumers must generate and consume markup documents containing ~~application defined~~ extension elements, including how to avoid and when to generate error conditions. *end note*]

…

Is the reference to Part 3, §12, "Application-Defined Extension Elements", still valid? If so, we need to change this to the new number for that clause. [Several, early clauses were deleted from Part 3 during the revision.]. The term "error conditions" is probably misleading. Would "conditions likely to be interpreted as errors" be better?

Part 1, §L.7.3, "Future Extensibility", p. 5001

This clause provides a high-level overview of the extensibility model for Office Open XML documents, and a description of packaging conventions in the context of DrawingML and PresentationML. Two main constructs are described: extensibility lists (extLst/ext) and alternate content blocks (AlternateContent).

To illustrate certain points, a number of examples refer to versions of a (fictitious) PresentationML consumer/producer called PML. The 2003 version is called PML 2003; the 2007 version is called PML 2007; and so on.

…

There is inconsistent use of terminology: "Application-Defined Extension Elements", "extension elements", "extensions", "extensibility lists",... We need to decide which is the right term in each context and provide definitions that relate the various terms used. Preferably, we should reduce the number of terms used.

Part 1, §L.7.3.4.2, "AlternateContent Blocks", p. 5005

…

This doesn't seem to offer anything over and above what Part 3 has. In any event. ISO editing rules do not permit a parent [sub]clause to contain text if it has subclauses. As such, we should consider removing this text.

Part 1, §L.7.3.4.2.1, "AlternateContent Syntax", p. 5006

…

It seems that if this is specified in Part 3, it need not be repeated here.

Part 1, §L.7.3.4.2.2, "Example", pp. 5006–5009

…

Part 1, §L.7.3.4.2.3, "AlternateContent Round-Trip Behavior", p. 5009

AlternateContent maintains multiple representations for the same content. Consider an extreme case. Using the example above, let's suppose one edited the label using PML 2007. As PML 2007 wouldn't understand future representations, there is no possibility that it could keep PML 2009's markup consistent with the edit performed. Considering a simple case, let's suppose one deleted the label in its entirety. PML 2007 would only know how to delete the corresponding textbox, and would not know how to update the corresponding cntrLblPr.

If this textbox contained sensitive information, one might consider this a security leak. The user's belief is that the information in the textbox was deleted, yet it persists in an alternate representation.

To solve this problem, all AlternateContent choices are discarded when an edit is performed at the location the AlternateContent is placed. It is the consuming client's responsibility to replace the discarded AlternateContent with a new representation.

If an edit to the label occurred in PML 2007, the PML 2009 version is discarded.

If an edit occurs in PML 2009, since PML 2009 understands both PML 2007 and PML 2009 schemas, it is possible for PML 2009 to write an updated AlternateContent Block encompassing an update to both choices.

This seems to be yet further duplication of the material on round-tripping alternate content.

Part 4, §2.1, "Document Conformance", p. 2

A document of conformance class Office Open XML Transitional shall be a package of conformance class OPC, as specified in ISO/IEC 29500-2, for which all the following shall hold:

- The document obeys all constraints specified in this Part of ISO/IEC 29500
- The document is of category Wordprocessing, Spreadsheet, or Presentation. These categories are defined in ISO/IEC 29500-1:2011 §4
- VML Drawing Parts (§8.1) contain markup in the Markup Compatibility namespace as specified in ISO/IEC 29500-3 ~~are of conformance class MCE, as specified in ISO/IEC 29500-3~~. Any child elements of the root element of VMLDrawing Parts are valid against the VML schema shown in A.6, "VML", after the removal of any extensions specified using the mechanisms in ISO/IEC 29500-3. VML Drawing Parts obey all constraints specified in this Part of ISO/IEC 29500
- For each OPC Part of the document of the types listed in §9.1 or ISO/IEC 29500-1:2011 §11.3, §12.3, §13.3, §14.2, and §15.2, all the following shall hold:

i. The part is of conformance class MCE, as specified in ISO/IEC 29500-3The Part may contain markup in the Markup Compatibility namespace as specified in ISO/IEC 29500-3

ii. After the removal of any extensions ~~using the mechanisms~~by an MCE processor as specified in ISO/IEC 29500-3, the part is valid against the Transitional W3C XML Schema (Annex A)

The revision of Part 3 proposes deleting the Conformance clause (and hence conformance class MCE). The tracked changes proposed above correspond to those proposed in DR 13-0009 for Part 1.