[DRAFT]

ISO/IEC TR 30114-1:201x

Extensions of Office Open XML File Formats — Guidelines

2016-02-20

Contents

Foreword ..... **ii**

Introduction ..... **iii**

1 Scope **1**

2 Normative References **2**

3 Adding markup or other data to OOXML documents **3**

3.1 General 3
3.2 MCE: Ignorable elements and attributes (IS 29500-3, §7.2) 3
3.3 MCE: Alternate Content Blocks (IS 29500-3, §7.5) 4
3.4 MCE: Application-defined extension elements (IS 29500-3, §8) 5
3.5 Embedding foreign OPC parts 6

Bibliography **8**

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, Specification, and Technical Reports 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 Technical Reports are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

ISO/IEC TR 30114-1 was prepared by ISO/IEC JTC 1, Information technology, Subcommittee SC 34, Document description and processing languages.

ISO/IEC 30114 consists of the following parts, under the general title *Information technology — Document description and processing languages — Extensions of Office Open XML File Formats*:

- *Part 1: Guidelines* (a Technical Report)
- *Part 2: Character Repertoire Checking* (a Standard)

Introduction

ISO/IEC 29500 was designed to allow the addition of markup and other data to OOXML documents, and to allow OOXML applications unaware of such markup and date to provide reasonable results. ISO/IEC 30114 provides a guideline for such additions, and also specifies a collection of such additions.

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

Part 1:
Guidelines

1 Scope

This Part of ISO/IEC 30114 gives guidelines for the use of extensibility mechanisms in ISO/IEC 29500 (Office Open XML). In particular, it makes clear which of these mechanisms provides lossless round tripping.

2 Normative References

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

ISO/IEC 26300, *Information technology -- Open Document Format for Office Applications (OpenDocument)*

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

ISO/IEC 29500-2, *Information technology — Document description and processing languages — Office Open XML File Formats, Part 2: Open Packaging Conventions*

ISO/IEC 29500-3, *Information technology — Document description and processing languages — Office Open XML File Formats – Part 3: Markup Compatibility and Extensibility*

3 Adding markup or other data to OOXML documents

3.1 General

There are two main ways to add extra markup or other data to OOXML documents:

- Using the extension mechanisms described in 29500-3, “Markup Compatibility and Extensibility (MCE)” (MCE offers three primary mechanisms for extending XML files, each with its own advantages and disadvantages. Each is discussed below.)
- Embedding foreign OPC parts.

3.2 MCE: Ignorable elements and attributes (IS 29500-3, §7.2)

The most commonly used extension mechanism, marking elements or attributes as ignorable, allows lightweight additions to be made to existing markup.

A good use of ignorable markup would be the addition of a custom metadata tag onto a paragraph in a WordprocessingML document. This could be accomplished by declaring a custom namespace, marking it as ignorable, and adding the attribute to the p element in that namespace. The relevant portions of the resulting document.xml part might resemble the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:mc=
  "http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:w=http://schemas.openxmlformats.org/wordprocessingml/2006/main
  xmlns:mymeta="http://mywordprocessorapp.com/metadata" mc:Ignorable="mymeta">
  <w:body>
    <w:p mymeta:tag="marketing_team" w:rsidR="00AD3E96"
      w:rsidRDefault="00120C37">
      <w:r>
        <w:t>hello</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Ignorable markup can be used anywhere in XML parts and requires minimal markup. It allows custom markup to be added to documents while retaining the document's conformance with the standard and allowing it to be opened by a third-party application without errors. However, ignorable elements and attributes will almost definitely be lost if files are round tripped (i.e., opened and saved again) in an application that does not

understand them, as there is no requirement for applications to persist ignorable markup, and typically unknown ignorable markup is stripped during file load.

3.3 MCE: Alternate Content Blocks (IS 29500-3, §7.5)

While ignorable constructs easily allow markup to be added to documents, Alternate Content Blocks (ACBs) allow existing markup to be replaced, with the replacement targeted at particular consumers that understand it.

A good use of ACBs would be in developing an application that preferred to use the ODF format in WordprocessingML paragraph markup. When creating files, the application would continue to write OpenXML markup in order to be compliant to the standard, but would also provide ODF markup in an ACB. When opening files, the application would disregard the OOXML fallback markup and only read the ODF.

The resulting document.xml part for a document with text stored in such a way might appear as shown below (note that, for simplicity, all ODF namespaces are merged into one):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:mc=
  http://schemas.openxmlformats.org/markup-compatibility/2006
  xmlns:w=http://schemas.openxmlformats.org/wordprocessingml/2006/main
  xmlns:myodf="http://mywordprocessorapp.com/odfcontent" mc:Ignorable="myodf">
  <w:body>
    <w:p w:rsidR="00AD3E96" w:rsidRDefault="00120C37">
      <mc:AlternateContent>
        <mc:Choice Requires="myodf">
          <myodf:style myodf:name="T2"
            myodf:parent-style-name="DefaultParagraphFont" myodf:family="text">
            <myodf:text-properties myodf:font-style="italic"
              myodf:font-style-asian="italic"/>
          </myodf:style>
          <myodf:p>This document is stored in<myodf:s/>
          <myodf:span myodf:style-name="T2">two</myodf:span>
          <myodf:s/>formats.</myodf:p>
        </mc:Choice>
        <mc:Fallback>
          <w:r>
            <w:t xml:space="preserve">This document is stored in </w:t>
          </w:r>
          <w:r>
            <w:rPr>
              <w:i/>
            </w:rPr>
            <w:t>two</w:t>
          </w:r>
```



```
            <w:r>
              <w:t xml:space="preserve"> formats.</w:t>
            </w:r>
          </mc:Fallback>
        </mc:AlternateContent>
      </w:p>
    </w:body>
  </w:document>
```

ACBs allow for the replacement of existing markup for consumers that understand it. Much like ignorables, there is no requirement for applications to preserve ACBs on round-trip operations, so data may be lost if third-party applications are used to open and save files.

3.4 MCE: Application-defined extension elements (IS 29500-3, §8)

Application-defined extension elements essentially allow markup designers to put "this space left for future expansion" elements into their formats. Syntactically, these are similar to ignorable elements but, because they only appear at predefined locations, markup consumers can easily keep track of unknown extension elements, which makes round tripping a simpler proposition.

In 29500-1, SpreadsheetML is the only markup that utilises extension elements (see extLst). extLst elements occur at several predefined points in SpreadsheetML and allow the markup to be extended in a manner that permits round-trip.

A good use of SpreadsheetML's extension elements would be a spreadsheet application whose developers wished to add the ability for cells to be denoted as model inputs or outputs. Such an application could use these tags at runtime and, if users were to round-trip the files in other applications, the markup would be preserved.

The CT_Cell type in SpreadsheetML contains an exLst element, so this will be an acceptable extension point. It contains an unbounded collection of ext elements, and the developer may add an ext with the developer's extension's markup. The resulting sheetdata for a given spreadsheet might look something like the following:

```
  <sheetData>
    <row r="1" spans="1:1" x14ac:dyDescent="0.25">
      <c r="A1">
        <v>1234</v>
        <extLst>
          <ext xmlns:mymodel=http://myspreadsheetapp.com/modelInputsAndOutputs
            uri="http://myspreadsheetapp.com/modelInputsAndOutputs">
            <mymodel:modelSpecifier cellType="input" name="interestRate">
              <mymodel:description>Specifies the interest rate to be passed into
  amortisation model.</mymodel:description>
            </mymodel:modelSpecifier>
          </ext>
```

```
        </extLst>
      </c>
    </row>
  </sheetData>
```

Because consuming spreadsheet applications understand that this data is attached at a cell level, this metadata remains with the cell when it is moved around the sheet via cut/paste or through row/column insertion or deletions above it. Note that some implementations may parse through application-defined-extension elements and modify constructs within them – Microsoft Excel, for example, will look for any sqref elements in the namespace http://schemas.microsoft.com/office/excel/2006/main. It assumes that they will contain spreadsheet row/column references and adjust them appropriately if that referenced cell area is moved around at runtime.

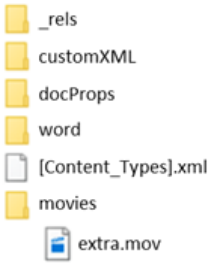
Application-defined extension elements are only usable in locations pre-defined by a markup language, but allow for data preservation in round-trip scenarios.

3.5 Embedding foreign OPC parts

Markup consumers are able (but not required) to preserve OPC parts with unrecognized relationship types during save operations. Such foreign parts are best suited to data (either binary or XML) that the creator desires to be preserved during round-trip operations.

One good use of a foreign part would be for an embedded video file attached to a WordprocessingML document. This involves three steps:

Firstly, the file is added to the OPC package:



Secondly, a relationship item is added to the _rels\.rels part. Relationships in this part are known as “package relationships,” since the source is the package as a whole.

```
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId5" Type="http://example.org/myexample"
  Target="movies/extra.mov"/>
</Relationships>
```

Thirdly, an element is added to the [Content_Types].xml stream, unless an appropriate element was already present.


```
<Default Extension="mov" ContentType="video/quicktime"/>
```

The content will likely be preserved on round-trip through non-understanding applications, and because there is no requirement to serialise it into XML this extension mechanism is well-suited to binary data such as video or images.

A second good use of a foreign part would be to embed richer metadata than supported in the Core Properties part of an OPC package (IS 29500-2, §10) or the Custom Properties part of an OOXML document (IS 29500-1, §15.2.12.2). For example, an ONIX for Books record could be embedded in the package for a WordProcessingML document using the same approach as outlined above. Although the ONIX record would be in XML, applications would not be required to parse or understand the record.

For an ONIX record in a part with name example_ONIX.xml, an appropriate location would be in a folder called "meta". An appropriate package relationship would be:

```
<Relationship Id="rId56" Type="http://ns.editeur.org/onix/3.0/reference"
Target="meta/example_ONIX.xml"/>
```

If not already present in the [Content_Types].xml stream, the following addition would be appropriate:

```
<Default Extension="xml" ContentType="application/xml"/>
```

Bibliography

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

Document Interoperability Initiative (DII) Workshop on MCE - Redmond, WA. September 18, 2009 https://msdn.microsoft.com/ja-jp/openspecifications/dn767917#tile=event091809

Excel (.xlsx) Extensions to the Office Open XML SpreadsheetML File Format https://msdn.microsoft.com/en-us/library/dd922181.aspx

Word Extensions to the Office Open XML (.docx) File Format https://msdn.microsoft.com/en-us/library/dd773189.aspx

Office Drawing Extensions to Office Open XML Structure https://msdn.microsoft.com/en-us/library/dd905216.aspx

PowerPoint (.pptx) Extensions to the Office Open XML File Format https://msdn.microsoft.com/en-us/library/dd926741.aspx

