

[DRAFT]

ISO/IEC 30114-1:201x

**Extensions of Office Open XML
File Formats — Guidelines**

2015-08-08

Contents

Foreword ii

Introduction..... iii

1 Scope 1

2 Normative References 2

3 Terms and Definitions 3

4 Adding data to OOXML documents 4

 4.1 General 4

 4.2 MCE: Ignorable elements and attributes (§xxx)..... 4

 4.3 MCE: Alternate Content Blocks (§xxx) 5

 4.4 MCE: Application-defined extension elements (§xxx) 6

 4.5 Embedding foreign OPC parts 7

Annex A (informative) XXX 8

Bibliography 9

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 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*
- *Part 2: Character Repertoire Checking*

Introduction

xxx

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

Part 1: Guidelines

1 Scope

xxx

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.

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

ISO/IEC 29500-1:201x, *Information technology — Document description and processing languages — Office Open XML File Formats, Part 1: Fundamentals and Markup Language Reference*.

ISO/IEC 29500-3:2015, *Information technology — Document description and processing languages — Office Open XML File Formats – Part 3: Markup Compatibility and Extensibility*

Commented [rcj1]:

3 Terms and Definitions

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

4 Adding data to OOXML documents

This clause is informative

4.1 General

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

- Using the extension mechanisms described in 29500-3, “Markup Compatibility and Extensibility (MCE)”
- Embedding foreign OPC parts.

MCE offers three primary mechanisms for extending XML files, each with its own advantages and disadvantages. Each is discussed below.

4.2 MCE: Ignorable elements and attributes (\$xxx)

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

Commented [CR2]: Should add link (bibliography) to our DII presentations from 2009 (https://www.assembla.com/wiki/show/IS29500/MCE_Best_Practices) and also the MS-. * extension documents

Commented [CR3]: Need new reference in doc

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

4.3 MCE: Alternate Content Blocks (§xxx)

Commented [CR4]: Need new reference in doc

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>
        </mc:Fallback>
      </mc:AlternateContent>
    </w:p>
  </w:body>
</w:document>
```

ISO/IEC 30114-1:201x(E)

```
<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.

4.4 MCE: Application-defined extension elements (§xxx)

Commented [CR5]: Need new reference in doc

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 extLst 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>
```

```

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

```

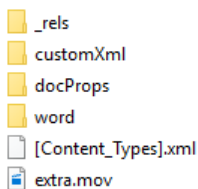
Because consuming spreadsheet applications understand that this data is attached at a cell level, this metadata remains with the cell when it is moved around the sheet via cut/paste or through row/column insertion or deletions above it. Note that some implementations may parse through application-defined-extension 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.

4.5 Embedding foreign OPC parts

Markup consumers are able (but not required) to preserve foreign OPC parts during save operations. Unknown parts are best suited to large amounts of data (either binary or XML) that the creator desires to be preserved during round-trip operations.

A good use of unknown parts would be for an embedded video file attached to a WordprocessingML document. First, we need to add the file to the OPC package:



And then add a relationship to the `_rels\rels` part:

```

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

```

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.

End of informative text

Annex A

(informative)

XXX

Xxx

Bibliography

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

xxx