Extensibility Strategy for ISO/IEC 29500

Abstract:

This document outlines a proposal for how ISO/IEC 29500 can be extended in a way that provides sufficient stability and backward compatibility for the ISO/IEC 29500 ecosystem and at the same time provides implementers and WG4 the ability to innovate as needed.

Disclaimer: This is a work in progress. Many updates are expected between now and the WG4 Stockholm meeting. Please send all comments to shawnv@microsoft.com.

Introduction

When we consider coming up with a strategy for how the standard should evolve, we need to define a foundation which should be reasonably stable and immutable. I propose that the schemas in ISO/IEC 29500:2008, plus the COR1s and the FPDAMs, make up that foundation.

Our goal would be not to make any breaking changes to this foundation. While we will need to continue reasonable maintenance on this foundation, we should strive to find non-breaking changes to solve our maintenance needs, and only make breaking changes in the most extreme circumstances.

And yet the standard needs to evolve. The standard was designed with extensibility in mind. Here is a sampling of mechanisms built into ISO/IEC 29500 to facilitate the safe extensibility of the standard:

- **XML Namespaces –** Provides for "major" version increments that are not designed to be backward compatible with existing implementations.
- **Extension Lists –** Provides for adding orthogonal data in predefined locations within the XML. See Part 1.
- **Parts –** Provides for adding entirely new payloads into the file container. See Part 2.
- **Alternative Content Blocks –** Provides for adding multiple renditions anywhere in the XML. See Part 3.
- **Ignorable Namespaces –** Provides for adding orthogonal data anywhere in the XML. See Part 3.

Given these technologies, I propose that WG4 adopt a guideline that states that all extensions to the standard be done using one or more of these technologies.

This is the same restriction that we place on implementers of the standard; if they want to respond to customer value, and that response requires adding information to the file format, their file format, for it to be compliant with ISO/IEC 29500, must add the necessary information using the official extension technologies. If they proceed to change the foundation schemas, their files are not compliant.

An Orthogonal Extension

So let's consider a scenario to see how this would play out. Since the Japanese national body has given WG4 notice that it intends to request an amendment to standardize the Microsoft Office 2010 extensions, let's use PowerPoint 2010 as our example for this discussion.

Suppose Microsoft decides to add the ability to group slides into sets of slides, called a section, within PowerPoint 2010. PowerPoint 2010, generating ISO/IEC 29500 transitionally compliant files, needs to find a way to add the new slide transition to its presentation files.

The notion of a section of slides is a grouping or list of slides. Other lists of slides, for example, are stored in the presentation.xml part. Here's the schema for the CT_Presentation:

```
<xsd:complexType name="CT_Presentation">
    <xsd:sequence>
        <xsd:element name="sldMasterIdLst" type="CT_SlideMasterIdList" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="notesMasterIdLst" type="CT_NotesMasterIdList" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="handoutMasterIdLst" type="CT_HandoutMasterIdList" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="sldIdLst" type="CT_SlideIdList" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="sldSz" type="CT_SlideSize" minOccurs="0" maxOccurs="1"/>
```

```
            <xsd:element name="notesSz" type="a:CT_PositiveSize2D" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="smartTags" type="CT_SmartTags" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="embeddedFontLst" type="CT_EmbeddedFontList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="custShowLst" type="CT_CustomShowList" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="photoAlbum" type="CT_PhotoAlbum" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="custDataLst" type="CT_CustomerDataList" minOccurs="0" maxOccurs="1/">
            <xsd:element name="kinsoku" type="CT_Kinsoku" minOccurs="0"/>
            <xsd:element name="defaultTextStyle" type="a:CT_TextListStyle" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="modifyVerifier" type="CT_ModifyVerifier" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="extLst" type="CT_ExtensionList" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="serverZoom" type="a:ST_Percentage" use="optional" default="50%"/>
        <xsd:attribute name="firstSlideNum" type="xsd:int" use="optional" default="1"/>
        <xsd:attribute name="showSpecialPlsOnTitleSld" type="xsd:boolean" use="optional" default="true"/>
        <xsd:attribute name="rtl" type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="removePersonalInfoOnSave" type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="compatMode" type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="strictFirstAndLastChars" type="xsd:boolean" use="optional" default="true"/>
        <xsd:attribute name="embedTrueTypeFonts" type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="saveSubsetFonts" type="xsd:boolean" use="optional" default="false"/>
        <xsd:attribute name="autoCompressPictures" type="xsd:boolean" use="optional" default="true"/>
        <xsd:attribute name="bookmarkIdSeed" type="ST_BookmarkIdSeed" use="optional" default="1"/>
        <xsd:attribute name="conformance" type="s:ST_ConformanceClass"/>
    </xsd:complexType>
```

You'll notice that the bolded line is one of the extensibility technologies mentioned earlier, extension lists.

This provides Microsoft with a location to store additional information that relates to the presentation. The role intent of the extension list is to add orthogonal information to a structure. That seems to work well here: since there is no existing section structure, we're adding new data to the presentation. Hence, Microsoft decide to use this location to store it's sections information.

Microsoft must then look up the CT_ExtensionList to see how the data can be stored. This complex type is defined as follows:

```
    <xsd:complexType name="CT_ExtensionList">
        <xsd:sequence>
            <xsd:group ref="EG_ExtensionList" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
```

To get the full picture, we need a few more things:

```
    <xsd:group name="EG_ExtensionList">
        <xsd:sequence>
            <xsd:element name="ext" type="CT_Extension" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:group>
```

and

```
    <xsd:complexType name="CT_Extension">
        <xsd:sequence>
            <xsd:any processContents="lax"/>
        </xsd:sequence>
        <xsd:attribute name="uri" type="xsd:token"/>
    </xsd:complexType>
```

To sum up these complex types, the extension list allows Microsoft to create a new extension, give it an identifier, and then put whatever XML they need to in that list.

After considering the user experience that they would like to deliver, Microsoft decides it needs a name for the section, a list of slides in that section, and an identifier for that section. And since there is always the possibility that this new functionality will need to be extended, they want to add another extension list. And here is the schema that is chosen:

```
    <xsd:complexType name="CT_Section">
        <xsd:sequence>
            <xsd:element name="extLst" type="p:CT_ExtensionList" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="slideIdLst" type="ST_SlideIdList"/>
        <xsd:attribute name="id" type="s:ST_Guid"/>
    </xsd:complexType>
```

Putting these pieces together, Microsoft has created an implementer extension. An instance example of this looks like this:

```
        <p:presentation>
            …
            <p:extLst>
                <p:ext uri="{521415D9-36F7-43E2-AB2F-B90AF26B5E84}">
                <p14:sectionLst xmlns:p14="http://schemas.microsoft.com/office/powerpoint/2010/main">
                    <p14:section name="Japan’s Section" id="{F2F5261B-7F1F-4862-85DB-557F78D8390F}">
                        <p14:sldIdLst>
                            <p14:sldId id="256" />
                        </p14:sldIdLst>
                    </p14:section>
                </p14:sectionLst>
                </p:ext>
            </p:extLst>
        </p:presentation>
```

This approach works well for Microsoft because it allows Microsoft to deliver additional customer value in its newer products; it also allows its previous version, PowerPoint 2007, to read files containing this information. This is also true for other compliant implementations.

In order to encourage interoperability between its presentation software and other companies’ presentation software, Microsoft documents this implementer extension (see http://msdn.microsoft.com/en-us/library/dd926741.aspx for an example of this).

At some later date, WG4 decides that the section feature meets the criteria to be added to the standard. It has three options at this point:

1. WG4 can accept the design of the implementer extension as-is, in its entirety.

2. WG4 can accept the design of the implementer extension as-is and change the uri attribute of the extension to a different one to differentiate it from the implementer extension;
3. WG4 can modify the design of the implementer extension and assign a new uri value to the extension to differentiate it from the implementer extension;

For the sake of exploring how this works, let us assume that WG4 takes option 3 as this presents the most complex scenario.

WG4 makes two changes to the official extension:

1. the extension uri attribute's value is now "http://www.purl.org/openxml/sections"; and,
2. the name attribute for the section element is renamed "sectionName."

This results in this complex type for the standard (emphasis added to changed line):

```
<xsd:complexType name="CT_Section">
    <xsd:sequence>
        <xsd:element name="extLst" type="p:CT_ExtensionList" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="sectionName" type="xsd:string"/>
    <xsd:attribute name="slideIdLst" type="ST_SlideIdList"/>
    <xsd:attribute name="id" type="s:ST_Guid"/>
</xsd:complexType>
```

And a new instance example would look like this (emphasis added to changed lines):

```
<p:presentation>
    …
    <p:extLst>
        <p:ext uri=" http://www.purl.org/openxml/sections ">
        <p2:sectionLst xmlns:p14="http://www.purl.org/openxml/presentationML">
            <p2:section sectionName="Japan's Section" id="{F2F5261B-7F1F-4862-85DB-557F78D8390F}">
                <p2:sldIdLst>
                    <p2:sldId id="256" />
                </p2:sldIdLst>
            </p14:section>
        </p14:sectionLst>
        </p:ext>
    </p:extLst>
</p:presentation>
```

At this point, Microsoft would have two choices as to how best to handle the newly standardized extension.

1. Microsoft could ignore it and continue just writing out its own extension; or,
2. Microsoft could add the standard extension to the files it generates.

Again, for the sake of exploring the scenario, let's assume that Microsoft takes option 2.

When Microsoft updates PowerPoint to accommodate the new extension, the instance example would look like this:

```
<p:presentation>
    …
    <p:extLst>
        <p:ext uri="{521415D9-36F7-43E2-AB2F-B90AF26B5E84}">
        <p14:sectionLst xmlns:p14="http://schemas.microsoft.com/office/powerpoint/2010/main">
            <p14:section name="Japan’s Section" id="{F2F5261B-7F1F-4862-85DB-557F78D8390F}">
                <p14:sldIdLst>
                    <p14:sldId id="256" />
                </p14:sldIdLst>
            </p14:section>
        </p14:sectionLst>
        </p:ext>
    </p:extLst>
    <p:extLst>
        <p:ext uri=" http://www.purl.org/openxml/sections ">
        <p2:sectionLst xmlns:p14="http://www.purl.org/openxml/presentationML">
            <p2:section sectionName="Japan’s Section" id="{F2F5261B-7F1F-4862-85DB-557F78D8390F}">
                <p2:sldIdLst>
                    <p2:sldId id="256" />
                </p2:sldIdLst>
            </p14:section>
        </p14:sectionLst>
        </p:ext>
    </p:extLst>
</p:presentation>
```

An Alternative Extension

TBD

A New Part Extension

TBD

Looking Into the Future

At some point in the lifetime of ISO/IEC 29500, the ecosystem will have created many extensions. Some will have worked their way into the standard and official extensions and some will remain implementer extensions. It is reasonable to expect a point in time – I would offer a decade down the road, perhaps, if history is any indicator of the future – that WG4 may determine that it is time to update the foundation schemas and fold in and or all of the extensions into the newly updated foundation schemas. But given the nature of document formats, I assert that that point in time is very far away.