3MF Consolidated Schema

User Guide

Version	1.0.0
Status	Draft

Contents

Revisions:

Version	Changes	Date/Editor
1.0.0	Initial Release	JZ – 4/12/22

Introduction

The normative requirements for 3MF consist of a Core specification and several Extension specifications. The Core specification appendices include an XML schema. Each 3MF Extension specification has appendices that document the schema objects relevant to the extension and how they fit into the Core schema. However, a single consolidated schema covering the Core specification and all the related Extensions is not part of the formal 3MF standards documentation.

As part of the 3MF test suite development, QualityLogic, Inc developed a consolidated 3MF schema so that test cases could be validated prior to delivering them the 3MF consortium. This required changes to the organization of various schema objects and several compromises were made when conflicts in optionality or restrictions existed across extension specifications.

This consolidated schema is being made available to the user community to simplify the task of validating 3MF package parts against the schema requirements. And to enable 3MF users to integrate their private namespace extensions into the consolidated schema to provide complete validation of the 3MF packages they generate. Please note that the Core and Extension specifications appendices remain the authoritative resource for the 3MF schema.

The consolidated schema utilized the requirements from the following 3MF specifications:

- *3MF Core Specification – Version 1.3.0*
- *3MF Materials and Properties Extension – Version 1.2.1*
- *3MF Production Specification – Version 1.1.2*
- *3MF Slice Specification – Version 1.0.2*
- *3MF Beam Lattice Extension – Version 1.2.0*
- *3MF Secure Content Extension – 1.0.2*

Consolidated Schema Structure

The following files are part of the consolidated schema:

- qli_3MF.xsd – This is the root schema file. Files below are imports
  - qli_balls_import.xsd
  - qli_beamlattice_import.xsd
  - qli_material.xsd
  - qli_mirrormesh.xsd
  - qli_printticket.xsd
  - qli_production_import.xsd
  - qli_SecureContent.xsd
  - qli_slice_import.xsd
  - qli_triangleset.xsd
  - qli_xenc_core_import.xsd
  - qli_xmldsig_import.xsd
  - xml.xsd
- The following standalone schema files are also included in the distribution
  - opc-contentTypes.xsd
  - opc-coreProperties.xsd
  - opc-relationships.xsd

In some cases, element and attribute definitions were moved from their extension of origin into other schema files to implement the consolidation of the various schema fragments.

XML Schema Basics

For a variety of reasons, 3MF content creators may add private namespace data to the model part of a 3MF package. This private namespace data is typically ignored by most 3MF content consumers but may have special meaning for other 3MF editors and consumers that can interpret the private namespace data. An XML schema allows you to define the following:

- The elements and attributes that can appear in a document
- The number of (and order of) child elements
- Data types for elements and attributes
- Default and fixed values for elements and attributes

An example of an XML Schema is shown below:

```
<xs:schema xmlns="http://private.namespace.com" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xml="http://www.w3.org/XML/1998/namespace" targetNamespace="http://private.namespace.com"
elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
   <!-- Complex Types -->
   <xs:complexType name="CT_Master_Assembly">
      <xs:sequence>
         <xs:element name="part1" type="xs:string"/>
         <xs:element name="part2" type="CT_Sub_Assembly" />
         <xs:element ref="part3" />
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="CT_Sub_Assembly">
      <xs:attribute name="pin" type="xs:integer"  default="6" />
      <xs:attribute name="plate" type="xs:integer" default="10"/>
      <xs:attribute name="cap" type="ST_Enums" default="big"/>
   </xs:complexType>
   <!-- Simple Types -->
   <xs:simpleType name="ST_Enums">
      <xs:restriction base="xs:string">
         <xs:enumeration value="none"/>
         <xs:enumeration value="big"/>
         <xs:enumeration value="small"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- Elements -->
   <xs:element name="master" type="CT_Master_Assembly"/>
   <xs:element name="part3" type="xs:integer"/>
   <!-- Attributes -->
   <xs:attribute name="cap2" type="xs:string"/>
   <xs:attribute name="new_list" type="ST_Enums"/>
</xs:schema>
```

An XML fragment that conforms to this schema would look like…

```
<master xmlns="http://private.namespace.com" ">
   <part1>String</part1>
   <part2 cap="big" plate="10" pin="6"/>
   <part3>0</part3>
</master>
```

An excellent source of information on the syntax for working with XML schemas is the XML Schema Tutorial at the W3 Schools web site (link below)

- https://www.w3schools.com/xml/schema_intro.asp

When working with XML schemas it is helpful to use a tool like Altova's XMLSpy which supports the following key features (http://www.altova.com/xmlspy):

- Validation of XML Schema syntax
- Validation of XML documents against a specified schema
- Generation of a sample XML document from a schema
- Ability to view an XML schema in a graphical format
- Great error information when schemas or XML documents do not validate

Integrating a Private Namespace Schema

If you would like to integrate a private namespace schema into the 3MF Consolidated schema, here are some helpful hints using the example schema shown earlier in this document.

Importing your private schema

Your private namespace schema should reside in the same subdirectory as the other consolidated schema files and should validate without errors using a program like XMLSpy. You will want to import your private schema into each of the consolidated schema files that you plan to integrate with. Imports should immediately follow the "xs:schema" element with the following syntax:

```
<xs:import namespace="http://private.namespace.com" schemaLocation="private.xsd"/>
```

You will also need to declare the namespace with a prefix using the "xs:schema" element "xmlns" attribute. The syntax is as follows:

```
<xs:schema xmlns:pn="http://private.namespace.com" ….>
```

Note that the "pn" prefix will be used in all subsequent integration steps to indicate that the elements and attributes you will be adding are part of your private namespace.

Add attribute to existing set of attributes

In the 3MF schema, attributes are typically defined in a complex object along with the related sub elements. Here is an example of an attribute from the private namespace example being added to the list of attributes for the build item:

```
<xs:complexType name="CT_Item">
  <xs:sequence>
    <xs:element ref="metadatagroup" minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="objectid" type="ST_ResourceID" use="required"/>
  <xs:attribute name="transform" type="ST_Matrix3D"/>
  <xs:attribute name="partnumber" type="xs:string"/>
  <xs:attribute ref="p:path"/>
  <xs:attribute ref="p:UUID" use="required"/>
  <xs:attribute ref="pn:cap2" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

The added attribute would appear in an XML document as pn:cap2="string"

Add element to existing set of elements

In the 3MF schema, elements are typically clustered in groups of sub element in a complex object. Here is an example of an element from the private namespace example being added to the list of allowable mesh sub elements:

```
<xs:complexType name="CT_Mesh">
  <xs:sequence>
    <xs:element ref="vertices"/>
    <!--  left triangle as optional for now-->
    <xs:element ref="triangles" minOccurs="0"/>
    <xs:element ref="ts:trianglesets" minOccurs="0"/>
    <xs:element ref="mm:mirrormesh" minOccurs="0"/>
    <xs:element ref="b:beamlattice" minOccurs="0"/>
    <xs:element ref="pn:part3"/>
  </xs:sequence
</xs:complexType>
```

The added element would appear in an XML document as `<pn:part3>12</pn:part3>`

Add new enumeration to existing set of enumerations

Enumeration values are not namespace specific, so you cannot just add an additional value to a list of enumerations and expect consumers to ignore the new enumeration if they are unaware of it. One solution would be to create a set of enumerations in your private namespace and map them to a private name space attribute. This could override the default set of enumerations for consumers that understood the private namespace extension while maintaining interoperability with other 3MF consumers.

Here is an example of an attribute from the private namespace example that is mapped to a set of enumerations and added to the list of attributes for the build item:

```
<xs:complexType name="CT_Item">
  <xs:sequence>
    <xs:element ref="metadatagroup" minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="objectid" type="ST_ResourceID" use="required"/>
  <xs:attribute name="transform" type="ST_Matrix3D"/>
  <xs:attribute name="partnumber" type="xs:string"/>
  <xs:attribute ref="p:path"/>
  <xs:attribute ref="p:UUID" use="required"/>
  <xs:attribute ref="pn:new_list" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

The added attribute would appear in an XML document as `pn:new_list="big"` with the enumeration “big” or possibly one of the other enumerations.

Reference a simple type

A simple type can be a list of enumerations, as in the prior example, or perhaps a restriction on the allowable values for a standard XML data type using regular expressions. Our example schema defines an attribute as follows:

```
<xs:attribute name="cap2" type="xs:string"/>
```

If we would like to require the "cap2" attribute to have 3 upper case alpha digits, you would define it as follows in the schema:

```
<xs:simpleType name="ST_cap2">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z][A-Z][A-Z]"/>
  </xs:restriction>
</xs:simpleType>

<xs:attribute name="cap2" type="ST_cap2"/>
```

The attribute with the new restrictions would be referenced as before in the complex type for the build item:

```
<xs:complexType name="CT_Item">
  <xs:sequence>
    <xs:element ref="metadatagroup" minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="objectid" type="ST_ResourceID" use="required"/>
  <xs:attribute name="transform" type="ST_Matrix3D"/>
  <xs:attribute name="partnumber" type="xs:string"/>
  <xs:attribute ref="p:path"/>
  <xs:attribute ref="p:UUID" use="required"/>
  <xs:attribute ref="pn:cap2" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

The added attribute with restrictions would appear in an XML document as pn:cap2="ADV"

Reference a complex type

A complex type may be used to define objects such as a list of sub elements or a set of attributes. Our example private namespace schema defines the following complex type:

```
<xs:complexType name="CT_Master_Assembly">
  <xs:sequence>
    <xs:element name="part1" type="xs:string"/>
    <xs:element name="part2" type="CT_Sub_Assembly" />
    <xs:element ref="part3" />
  </xs:sequence>
</xs:complexType>
```

This complex type can be used as the type for another element in the private namespace schema, such as…

```
<xs:element name="master" type="CT_Master_Assembly"/>
```

And this element can then be referenced in another 3MF schema as follows:

```
<xs:complexType name="CT_MetadataGroup">
  <xs:sequence>
    <xs:element ref="metadata" maxOccurs="unbounded"/>
    <xs:element ref="pn:master"/>
  </xs:sequence>
</xs:complexType>
```

With the resulting output in an XML document:

```
<metadatagroup>
  <metadata name="xml:lang" type="String" preserve="true">text</cr:metadata>
  <pn:master>
    <pn:part1>String</pn:part1>
    <pn:part2 cap="big" plate="10" pin="6"/>
    <pn:part3 cap="big" plate="10" pin="6"/>
  </pn:master>
<metadatagroup>
```

Consolidated Schema Exceptions

The following are instances where the consolidated schema may produce validation results that differ from the schema definitions in the 3MF Core and Extension specifications:

qli_3MF.xsd

- "xs:any" was commented out in CT_Resources as it was causing errors with the integration of the slice extension. Implication: you will get a schema error if a sub element of "resources" is present that is not part of the consolidated schema definition.

- minOccurs was set to zero for the "item" element in CT_Build as it is only required in a root model part. Implication: You will not get a schema validation error if you omit the "item" sub element under the "build" element in a root model file

- minOccurs was set to zero for triangles in CT_Mesh as the Beam Lattice extension and Triangles Set schema allow the "triangles" element to be empty. Implications: You will not get a schema validation error if there are no "triangle" sub elements under the "triangles" element

- "xs:any" was commented out in CT_Mesh as it was causing errors with the integration of the Beam Lattice extension. Implication: you will get a schema error if a sub element of "mesh" is present that is not part of the consolidated schema definition.

- minOccurs was set to zero for "vertices" in CT_Vertices as the Mirror Mesh schema allows it to be zero and the Beam Lattice extension allows it to be 2 (normal is 3). Implications: You will not get a schema validation error if there are no "vertex" sub elements under the "vertices" element

qli_slice_import.xsd

- minOccurs of zero was deleted for "slice" and "sliceref" in CT_SliceStack to catch empty slice stacks in test cases. Implications: You will not get a schema validation error if there is an empty slice stack

- "required" was removed from the "p1", "p2", and "pid" attributes in CT_Segment as this was considered an error in the schema based on the UML diagrams and slicer behaviors. Implications: A schema validation error will not be generated if these attributes are omitted in the "segment" element.