3MF Production Extension

Specification & Reference Guide

Version	1.1
Status	Published

Table of Contents

Preface

About this Specification

This 3MF Production Extension specification is an extension to the core 3MF specification. This document cannot stand alone and only applies as an addendum to the core 3MF specification. Usage of this and any other 3MF extensions follow an a la carte model, defined in the core 3MF specification.

Part I, "3MF Documents," presents the details of the primarily XML-based 3MF Document format. This section describes the XML markup that defines the composition of 3D documents and the appearance of each model within the document.

Part II, "Appendixes," contains additional technical details and schemas too extensive to include in the main body of the text as well as convenient reference information.

The information contained in this specification is subject to change. Every effort has been made to ensure its accuracy at the time of publication.

This extension MUST be used only with Core specification 1.0.

Document Conventions

Except where otherwise noted, syntax descriptions are expressed in the ABNF format as defined in RFC 4234.

Glossary terms are formatted like *this*.

Syntax descriptions and code are formatted in `monospace` type.

Replaceable items, that is, an item intended to be replaced by a value, are formatted in *`monospace cursive`* type.

Notes are formatted as follows:

> **Note:** This is a note.

Language Notes

In this specification, the words that are used to define the significance of each requirement are written in uppercase. These words are used in accordance with their definitions in RFC 2119, and their respective meanings are reproduced below:

- *MUST.* This word, or the adjective “REQUIRED,” means that the item is an absolute requirement of the specification.
- *SHOULD.* This word, or the adjective “RECOMMENDED,” means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
- *MAY.* This word, or the adjective “OPTIONAL,” means that this item is truly optional. For example, one implementation may choose to include the item because a particular marketplace or scenario requires it or because it enhances the product. Another implementation may omit the same item.

Software Conformance

Most requirements are expressed as format or package requirements rather than implementation requirements.

In order for consumers to be considered conformant, they must observe the following rules:

- They MUST NOT report errors when processing conforming instances of the document format except when forced to do so by resource exhaustion.
- They SHOULD report errors when processing non-conforming instances of the document format when doing so does not pose an undue processing or performance burden.

In order for producers to be considered conformant, they must observe the following rules:

- They MUST NOT generate any new, non-conforming instances of the document format.
- They MUST NOT introduce any non-conformance when modifying an instance of the document format.

Editing applications are subject to all of the above rules.

PART I: 3MF PRODUCTION EXTENSION

Chapter 1. Overview of Additions

This document describes new non-object resources, as well as attributes to the build section for uniquely identifying parts within a particular 3MF package. If not explicitly stated otherwise, each of these resources is OPTIONAL for producers, but MUST be supported by consumers that specify support for the Production Extension of 3MF.

In order to allow for the use of 3MF in high production printing environments, several additions are needed to efficiently support packed build platforms and ensure integrity of the payload:

- Enable the 3MF <build> elements to address objects in separate files within the 3MF package
- Identify each build, each object and each copy of a part with unique identifiers

A consumer supporting the production extension MUST be able to consume non-extended core 3MFs, even if this is not as efficient. As the production extension is just a reorganization of data, a consumer MAY convert a generic core 3MF into a “production extended 3MF” before internally processing the data.

In order to avoid data loss while parsing, a 3MF package which uses referenced objects MUST enlist the production extension as “required extension”, as defined in the core specification.

Chapter 2. Model Relationships

The primary emphasis of this extension is the possibility to store model data in files separate from the root model file and to allow the root model file's build element to reference those resources. This structural approach enables three primary advantages for producers and consumers of 3MF packages with large numbers of individual models:

- The build directive in the root model file can be parsed by consumers without having to parse any actual model data.
- Moving from a single-part 3MF (e.g. from a design application) into a high-part-density 3MF build can be largely a pass-through from the original 3MF to the "production" 3MF.
- Parsing the model objects from individual XML files will often require fewer resources than parsing a monolithic model file that could be more that 500MB in size.

The root model part MAY have relationships to other model parts, whose object resources can be referenced by the parent model stream using the "path" attribute on a build item element or within a component.

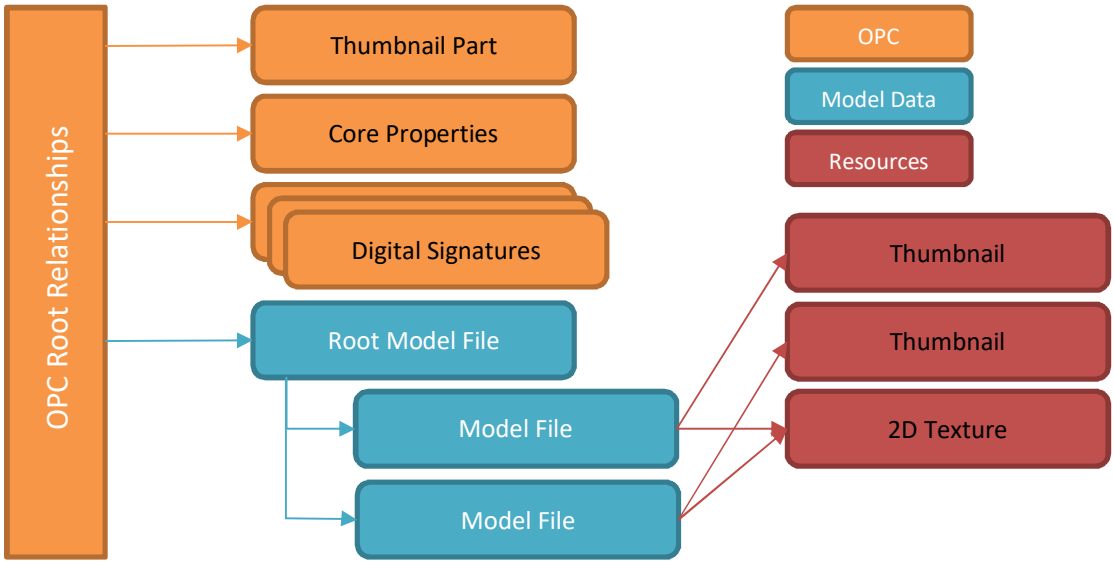
As defined in the core 3MF spec, only the build section of the root model file contains valid build information. Other model streams SHOULD contain empty build sections. Every consumer MUST ignore the build section entries of all referenced child model files.

Further, only a component element in the root model file may contain a path attribute. Non-root model file components MUST only reference objects in the same model file.

These two limitations ensure there is only a single level of "depth" to multi-file model relationships within a package and explicitly prevents complex or circular object references.

Chapter 3. Production Extension Data Details

Figure 3–1. A typical production 3MF Document with multiple model streams

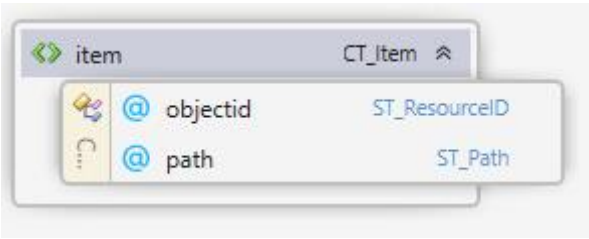


3.1. The Path Attribute

3.1.1. Item

Within the <item> element of the build section in the root model, there is a new, optional attribute called “path”. Path is an absolute path to the target model file inside the 3MF container that contains the target object. When the path attribute is used, objectid becomes a reference to the object within the referenced model.

diagram



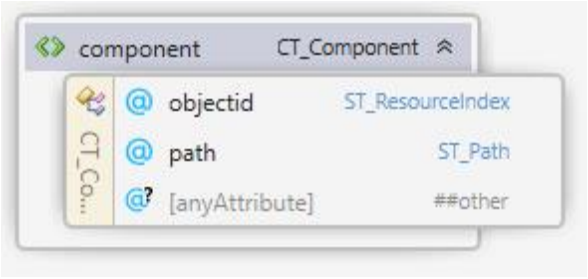
attributes

Name	Type	Use	Default	Fixed	Annotation
path	**ST_Path**	required			A file path to the model file being referenced. The path is an absolute path from the root of the 3MF container.
objectid	**ST_ResourceID**	required			Objectid is part of the core 3MF specification, and its use in the production extension the same: objectid indexes into the model file to the object with the corresponding id. The only difference is that the path attribute identifies the target file from which to load the specified object.

3.1.2. Component

Within the <component> elements of component-based objects, the “path” attribute references objects in non-root model files. Path is an absolute path to the target model file inside the 3MF container that contains the target object. The use of the path attribute in a component element is ONLY valid in the root model file.

Diagram



Attributes

Name	Type	Use	Default	Fixed	Annotation

	path	**ST_Path**	required	A file path to the model file being referenced. The path is an absolute path from the root of the 3MF container.
	objectid	**ST_ResourceID**	required	Objectid is part of the core 3MF specification, and its use in the production extension the same: objectid indexes into the model file to the object with the corresponding id. The only difference is that the path attribute identifies the target file from which to load the specified object.

3.2. Path Usage

The path attribute is optional, even for 3MF containers that claim support for the production extension. It is possible to construct a 3MF package with objects both in the root model file and in other model files in the container.

Some considerations for using multiple file 3MF constructions with the path attribute:

Objects referenced by the path and objected attribute of the build item inherit all of their properties from their own object definition and resources. All of the resources associated with the referenced object (textures, materials, thumbnails, name, part number, etc.) must come from the referenced object file.

The model level Metadata element is only valid in the root model file of a 3MF package. All Metadata elements in other model files in a 3MF package will be ignored.

A path attribute can reference an object in a target file that is made-up of components. In this case, the same processing rules apply as with a local component object: the object transforms are relative to the item transform and consumers MUST not alter the relative transformations within the component objects.

A root model based component can be partially, or fully, composed of objects from different model files. This allows a single component reference to place objects from multiple files (including the root model file) into a virtual assembly that it bound to the origin of the build item transformation. Any consumer of a 3MF package that contains path attributes in components in a non-root model file MUST generate an error for that package.

Because there can be only a single build section (in the root model), there is at most a single level of referenced objects in any 3MF container. This eliminates the possibility for complex or circular object references between model files in 3MF containers.

3.3. OPC Relation Files

All model files in the 3MF package must be referenced in .rels files in order to conform with OPC standards. The root model file MUST always be referenced in the root .rels file with Id=”0”. Example root .rels file:

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Type="http://schemas.microsoft.com/3dmanufacturing/2013/01/3dmodel" Target="/3D/build.model"
    Id="rel0" />
    <Relationship Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/thumbnail"
    Target="/Metadata/thumbnail.png" Id="rel4" />
</Relationships>
```

Non-root model files must not be referenced from the root .rels file. Referenced model files must be included the .rels file from the referencing model file according to the part relationship defined in OPC. For example, assuming that the root model file in the /3D folder is named model.model, the non-root model file references must be in the /3D/_rels/model.model.rels file:

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Type="http://schemas.microsoft.com/3dmanufacturing/2013/01/3dmodel"
    Target="/3D/object1.model" Id="rel1" />
    <Relationship Type="http://schemas.microsoft.com/3dmanufacturing/2013/01/3dmodel"
    Target="/3D/object2.model" Id="rel2" />
    <Relationship Type="http://schemas.microsoft.com/3dmanufacturing/2013/01/3dmodel"
    Target="/3D/object3.model" Id="rel3" />
</Relationships>
```

Chapter 4. Identifying Build Components

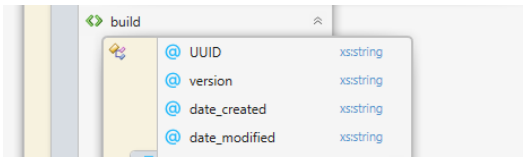
Components of 3MF containers need to be uniquely identifiable in order to ensure tracking of builds and parts through build processes. Within a given 3MF container, build items can be uniquely identified by providing a UUID for each <item>. Individual objects (models) in a build can be uniquely identified with a UUID on each <object> element. Individual component-based parts can be identified using a UUID attribute on <component> elements.

The Production Extension REQUIRES that both <item> and <component> elements include the UUID attribute as a mechanism to identify model instances being used in the 3MF package.

In some environments, it is crucial that the builds themselves can be uniquely identified within, and even across, different physical printers. In order to support cross-printer and cross-print-job build identification, <build> elements in the root model part REQUIRE a UUID attribute.

4.1. Build

Element **<build>**

Diagram	

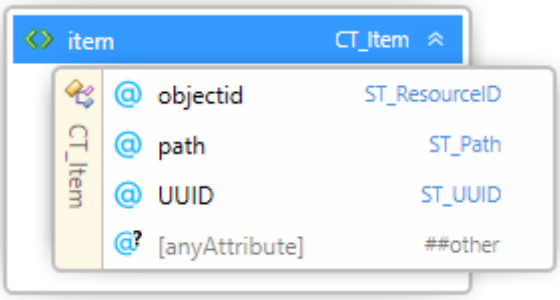
attributes

Name	Type	Use	Default	Fixed	Annotation
uuid	**ST_UUID**	required			A universally unique ID that allows the build to be identified over time and across physical clients and printers.

Producers MUST provide a UUID in the root model file build element to ensure that a 3MF package can be tracked across uses by various consumers.

4.1.1. Item

Element **<item>**

Diagram	

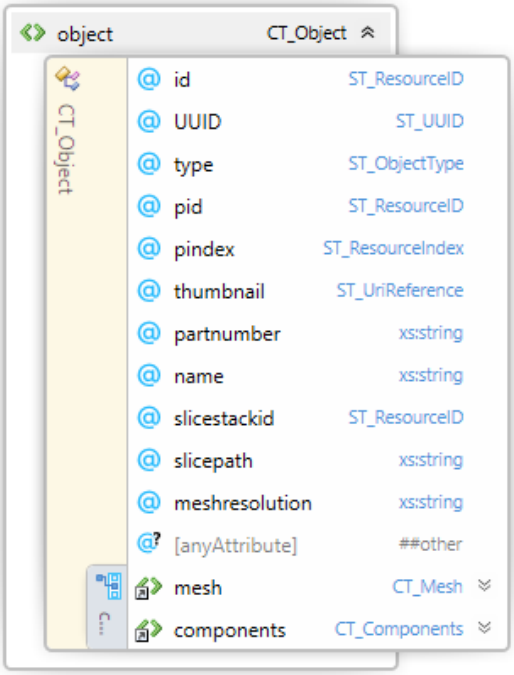
attributes

Name	Type	Use	Default	Fixed	Annotation
UUID	**ST_UUID**	required			A globally unique identifier for each item in the 3MF package which allows producers and consumers to track part instances across 3MF packages.

Producers MUST include UUID's for all build items for traceability across 3MF packages.

4.2. Object

Element <**object**>

Diagram	

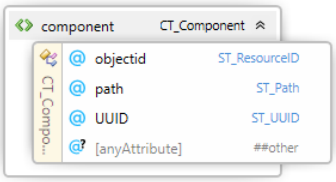
attributes	Name	Type	Use	Default	Fixed	Annotation
	UUID	**ST_UUID**	required			A globally unique identifier for each <object> in the 3MF package which allows producers and consumers to track object instances across 3MF packages. In the case that an <object> is made up of <components>, the UUID represents a unique ID for that collection of object references.

Producers MUST include UUID's in all <object> references to ensure that each object can be reliably tracked.

> **Note:** As for the use case of production data, the uniqueness properties are sufficient, this specification tries to avoid the technical details about conforming UUIDs according to ITU-T X.667 ISO/IEC 9834-8:2004. "Unique identifier" always may mean any of the four UUID variants described in IETF RFC 4122, which includes Microsoft GUIDs as well as time-based UUIDs.

4.2.1. Component

Element <**component**>

Diagram	
	

attributes	Name	Type	Use	Default	Fixed	Annotation
	UUID	**ST_UUID**	required			A globally unique identifier for each object component in the 3MF package which allows producers and consumers to track part instances across 3MF packages.

Producers MUST include UUID's in all component-based object references to ensure that each instance of an object can be reliably tracked.

PART II. APPENDIXES

Appendix A. Glossary

3D model. The markup that defines a model for output.

3D Model part. The OPC part that contains a 3D model.

3MF. The 3D Manufacturing Format described by this specification, defining one or more 3D objects intended for output to a physical form.

3MF Document. The digital manifestation of an OPC package that contains a 3D payload that conforms with the 3MF specification.

Consumer. A software, service, or device that reads in a 3MF Document.

Producer. A software, service, or device that writes out a 3MF Document.

XML namespace. A namespace declared on the <model> element, in accordance with the XML Namespaces specification.

Appendix B. 3MF Production Extension Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://schemas.microsoft.com/3dmanufacturing/production/2015/06"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xml="http://www.w3.org/XML/1998/namespace"
targetNamespace="http://schemas.microsoft.com/3dmanufacturing/production/2015/06"
elementFormDefault="unqualified" attributeFormDefault="unqualified" blockDefault="#all">
   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>

   <!-- Complex Types -->

   <xs:complexType name="CT_Item">
      <xs:attribute name="objectid" type="ST_ResourceID" use="required"/>
      <xs:attribute name="path" type="ST_Path"/>
      <xs:attribute name="UUID" type="ST_UUID" use="required"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

    <xs:complexType name="CT_Component">
      <xs:attribute name="objectid" type="ST_ResourceID" use="required"/>
      <xs:attribute name="path" type="ST_Path"/>
      <xs:attribute name="UUID" type="ST_UUID" use="required"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>

   <xs:complexType name="CT_Object">
      <xs:choice>
         <xs:element ref="mesh"/>
         <xs:element ref="components"/>
      </xs:choice>
      <xs:attribute name="id" type="ST_ResourceID" use="required"/>
      <xs:attribute name="UUID" type="ST_UUID" use="required"/>
      <xs:attribute name="type" type="ST_ObjectType" default="model"/>
      <xs:attribute name="pid" type="ST_ResourceID"/>
      <xs:attribute name="pindex" type="ST_ResourceIndex"/>
      <xs:attribute name="thumbnail" type="ST_UriReference"/>
      <xs:attribute name="partnumber" type="xs:string"/>
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="slicestackid" type="ST_ResourceID"/>
      <xs:attribute name="slicepath" type="xs:string"/>
      <xs:attribute name="meshresolution" type="xs:string">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:enumeration value="build" />
               <xs:enumeration value="preview" />
               <xs:enumeration value="boundingbox" />
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <!-- Simple Types -->

   <xs:simpleType name="ST_Path">
      <xs:restriction base="xs:string">
      </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="ST_UUID">
      <xs:restriction base="xs:string">
         <xs:pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
      </xs:restriction>
   </xs:simpleType>

</xs:schema>
```

Appendix C. Standard Namespaces and Content Types

C.1 Namespaces

Production http://schemas.microsoft.com/3dmanufacturing/production/2015/06

References

BNF of Generic URI Syntax

"BNF of Generic URI Syntax." World Wide Web Consortium.
http://www.w3.org/Addressing/URL/5_URI_BNF.html

JPEG

Hamilton, Eric. "JPEG File Interchange Format, Version 1.02." World Wide Web Consortium. 1992.
http://www.w3.org/Graphics/JPEG/jfif3.pdf

Open Packaging Conventions

Ecma International. "Office Open XML Part 2: Open Packaging Conventions." 2006. http://www.ecma-international.org

PNG

Duce, David (editor). "Portable Network Graphics (PNG) Specification," Second Edition. World Wide Web Consortium. 2003. http://www.w3.org/TR/2003/REC-PNG-20031110

sRGB

Anderson, Matthew, Srinivasan Chandrasekar, Ricardo Motta, and Michael Stokes. "A Standard Default Color Space for the Internet-sRGB, Version 1.10." World Wide Web Consortium. 1996.
http://www.w3.org/Graphics/Color/sRGB

Unicode

The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: *The Unicode Standard, Version 4.0*. Boston, MA: Addison-Wesley, 2003.

XML

Bray, Tim, Eve Maler, Jean Paoli, C. M. Sperlberg-McQueen, and François Yergeau (editors). "Extensible Markup Language (XML) 1.0 (Fourth Edition)." World Wide Web Consortium. 2006.
http://www.w3.org/TR/2006/REC-xml-20060816/

XML C14N

Boyer, John. "Canonical XML Version 1.0." World Wide Web Consortium. 2001.
http://www.w3.org/TR/xml-c14n.

XML Namespaces

Bray, Tim, Dave Hollander, Andrew Layman, and Richard Tobin (editors). “Namespaces in XML 1.0 (Second Edition).” World Wide Web Consortium. 2006. http://www.w3.org/TR/2006/REC-xml-names-20060816/

XML Schema

Beech, David, Murray Maloney, Noah Mendelsohn, and Henry S. Thompson (editors). “XML Schema Part 1: Structures,” Second Edition. World Wide Web Consortium. 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

Biron, Paul V. and Ashok Malhotra (editors). “XML Schema Part 2: Datatypes,” Second Edition. World Wide Web Consortium. 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/