

3MF Volumetric & Implicit Extensions

Specification & Reference Guide

Version **1.0.0**

Status Published

Disclaimer

THESE MATERIALS ARE PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to the materials. The entire risk as to implementing or otherwise using the materials is assumed by the implementer and user. IN NO EVENT WILL ANY MEMBER BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS DELIVERABLE OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER MEMBER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

- Change history
- Preface
  - Introduction
  - About this Specification
  - Document Conventions
  - Language Notes
  - Software Conformance
- Part I: Volumetric Extension
  - Chapter 1. Overview of Volumetric Additions
  - Chapter 2. Functions
  - Chapter 3. 3D Image
  - Chapter 4. LevelSet
  - Chapter 5. Volumetric Data
  - Chapter 6. Notes
- Part II. Implicit Extension
  - Chapter 1. Overview of Implicit Additions
  - Chapter 2. DataTypes
  - Chapter 3. Function Implicit
  - Chapter 4. Nodes
  - Chapter 5. Native Nodes
  - Chapter 6. Implicit Evaluation
  - Chapter 7. Notes
- Part III. Appendices
  - Appendix A. Glossary
    - Appendix B. 3MF XSD Schema for the Volumetric and Implicit Extensions
    - Appendix C. Namespaces

- Appendix D: Example file
- References

Change History

Version	Changes Description	Date
1.0	First published version	November 13, 2025

Preface

Introduction

Volumetric/Implicit Modeling is an efficient approach to encode geometrical shapes and spatial properties and is based on a volumetric description. Traditional, explicit modeling methodologies are based on surfaces (e.g. NURBS, triangular meshes) that describe the boundaries of an object. This is illustrated in Figure 1-1. a) a NURBS surface delimitates a region of space. Figure 1-1 b) shows a triangular mesh that describes the same surface. In each case, the top part of the described object is being shown transparently to allow viewing the "inside" of the described object.

The implicit modeling approach relies on a mathematical, field-based description of the whole volume of the object. This is illustrated in Figure 1-1 c). Every point in space has a scalar (grey-scale) value. The iso-surface at value 0 describes the surface of the same object as in Figure 1-1 a). A section of this iso-surface is indicated by the blue line.

The true advantage of volumetric modeling shows when properties of an object vary in space gradually, e.g. color or material-distribution and -composition of an object vary in space. E.g. in Figure 1-2 a) the object has a uniform color except for a red stripe at the front-left surface that gradually turns into turquoise. Note that not only the surface, but also the interior volume has a non-uniform color in this region. Figure 1-2 b) shows a distribution of three different materials, indicated by three different colors, red, blue and green.

This is only a brief illustration of the implicit modeling approach to geometric design and more information can be found in references [1] and [2].

Figure 1-1. Explicit (a) and (b) vs. implicit (c) representation

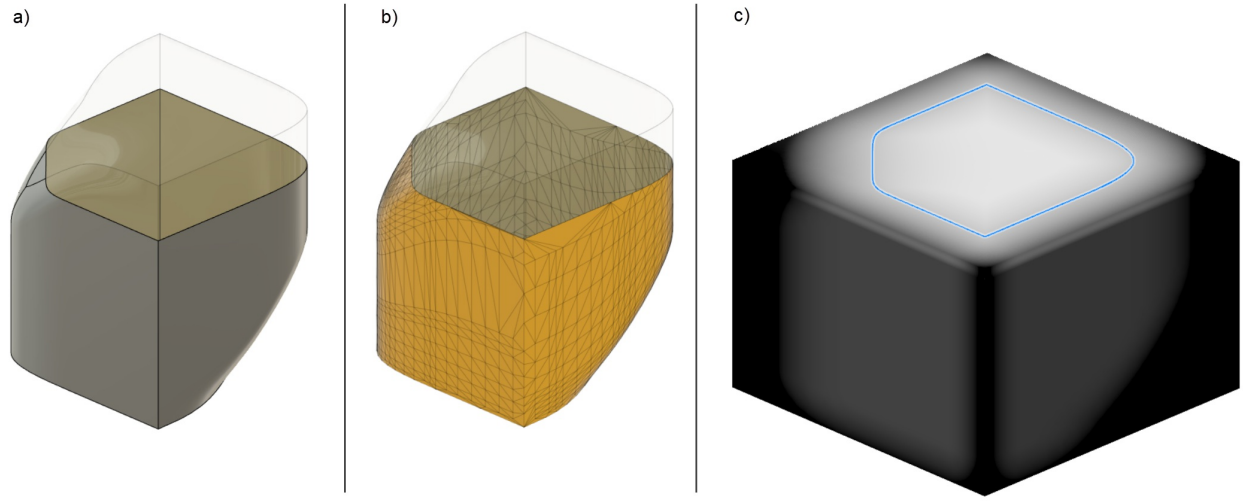
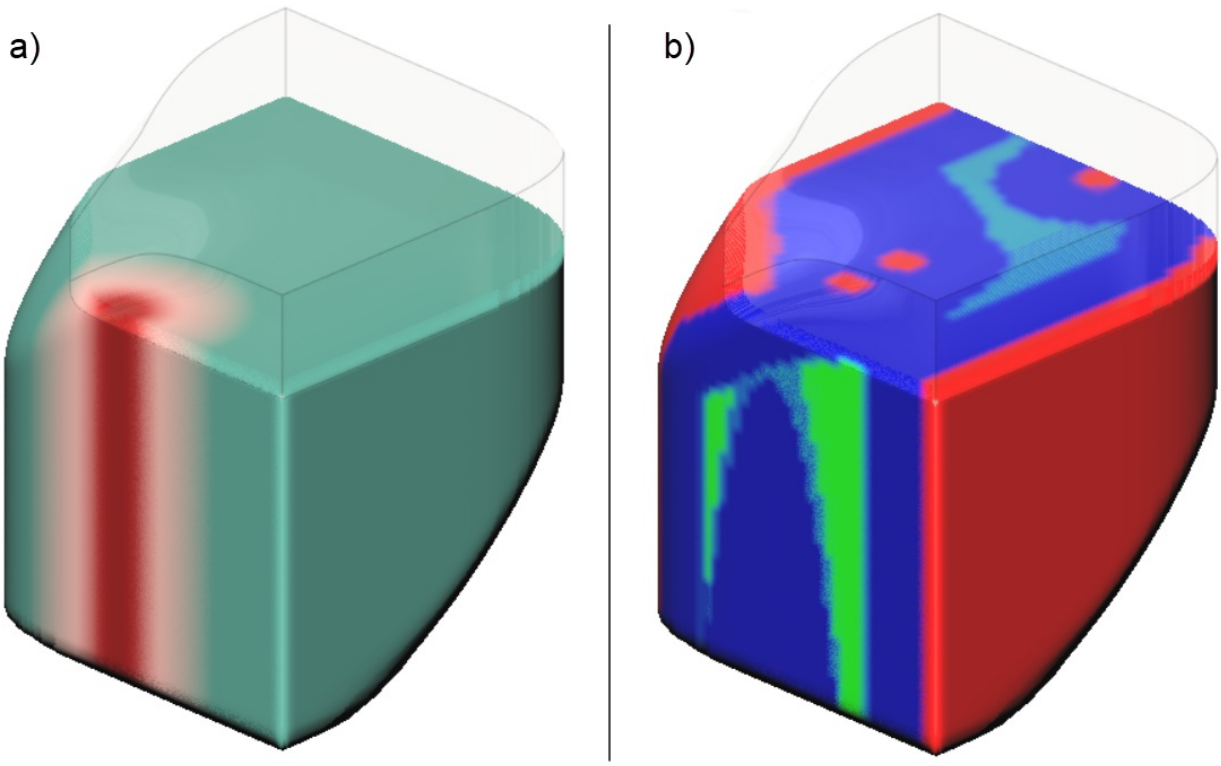


Figure 1-2. Spatially varying properties



About this Specification



This 3MF volumetric 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.

This 3MF implicit specification is an extension to the core 3MF specification and the 3MF volumetric 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, "Volumetric Extension," 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, "Implicit Extension," describes the XML markup for describing implicit functions and their evaluation by defining a graph of nodes and their connections.

Part III, "Appendices," 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 version 1.3. or higher.

Document Conventions

See the standard 3MF Document Conventions documentation:
https://github.com/3MFConsortium/spec_resources/blob/master/document_conventions.md

In this extension specification, the prefix "m" maps to the XML namespace "http://schemas.microsoft.com/3dmanufacturing/material/2015/02", "v" to "http://schemas.3mf.io/3dmanufacturing/volumetric/2022/01" and "i" to "http://schemas.3mf.io/3dmanufacturing/implicit/2023/12". See Appendix C. Namespaces.

Language Notes

See the standard 3MF Language Notes documentation.

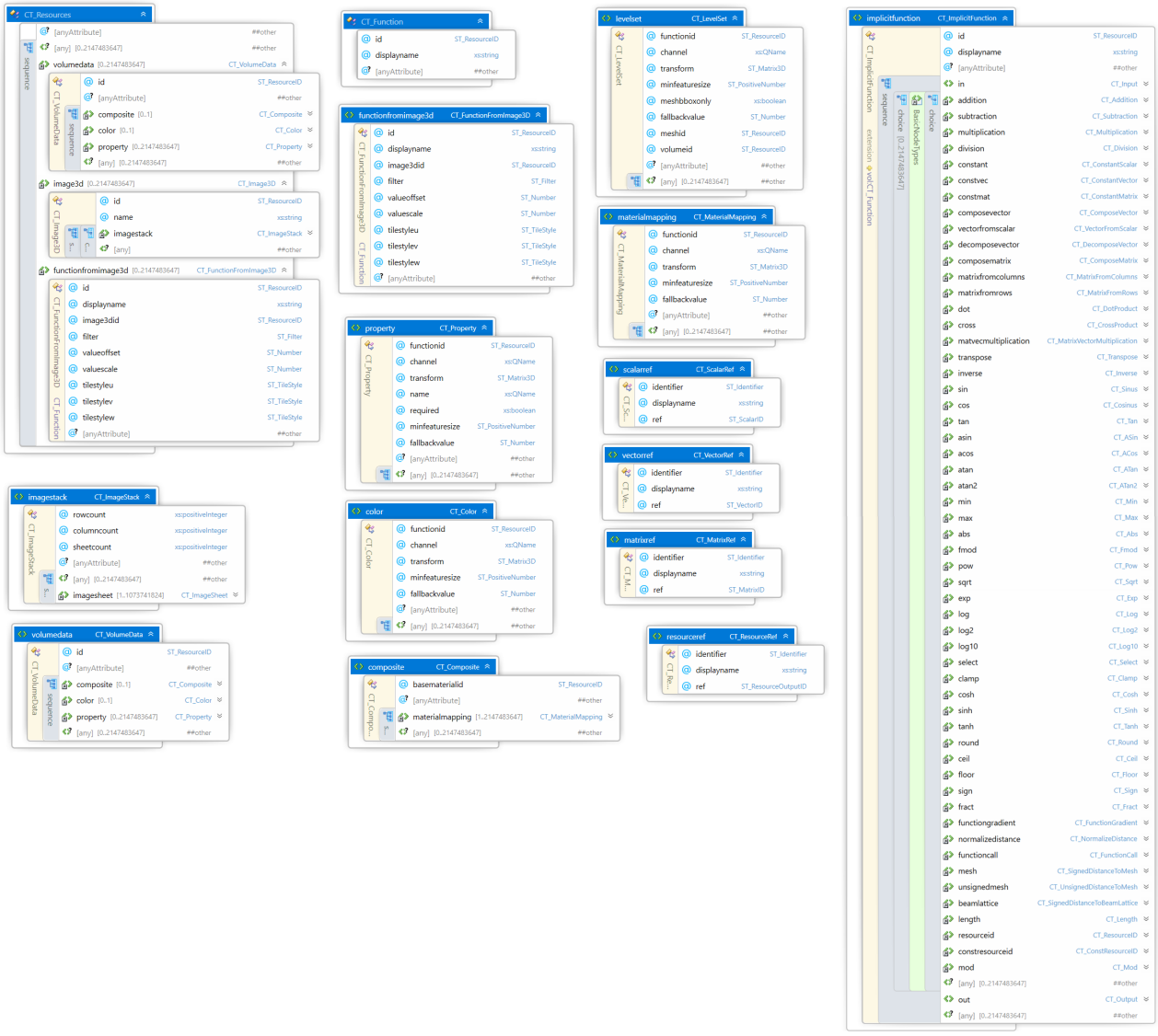
Software Conformance

See the standard 3MF Software Conformance documentation.

Part I. Volumetric Extension

Chapter 1. Overview of Volumetric Additions

Figure 1-1: Overview of model XML structure of 3MF with volumetric additions



This document describes new elements, each of which is OPTIONAL for producers. Consumers MUST be able to parse all new elements but only MUST support levelset shape.

There are two central ideas of this extension. The first is to provide a new geometry representation as an alternative to a mesh using levelsets. The second is to enrich the geometry notion of 3MF with volumetric elements that can represent spatially varying properties which are quite inefficient to handle with a mesh representation, especially in cases where the variation is continuous in space.

This extension is meant to be an exact specification of geometric, appearance-related, material and in fact arbitrary properties, and consumers MUST interpret it as such. However, the intent is also to enable editors of 3MF files to use the designated data structures for efficient interoperability and post-processing of the geometry and properties described in this extension.

A producer using the level set of the volumetric specification MUST mark the extension as required, as described in the core specification. Producers only using the other volume data elements, in particular color-, composite- and property-elements, MAY mark the extension as REQUIRED, and MAY be marked as RECOMMENDED. Producers of 3MF files that do not mark the volumetric extension as required are thus assured that the geometric shape of objects in this 3MF file are not altered by the volumetric specification.

1.1 Breaking Change to the Core Specification

The core specification defines an ordering constraint inside the <resources> element through its XSD: all <object> elements occur only after the repeated sequence that can contain <basematerials> and any extension-defined resources (via the <any> placeholder). Consequently, in a pure core (or non-volumetric) document, producers cannot legally

place an <object> *before* an extension resource, and extension resources cannot legally appear*after* an <object> . This ordering collides with new volumetric requirements where the following extension resources may need to reference mesh objects:

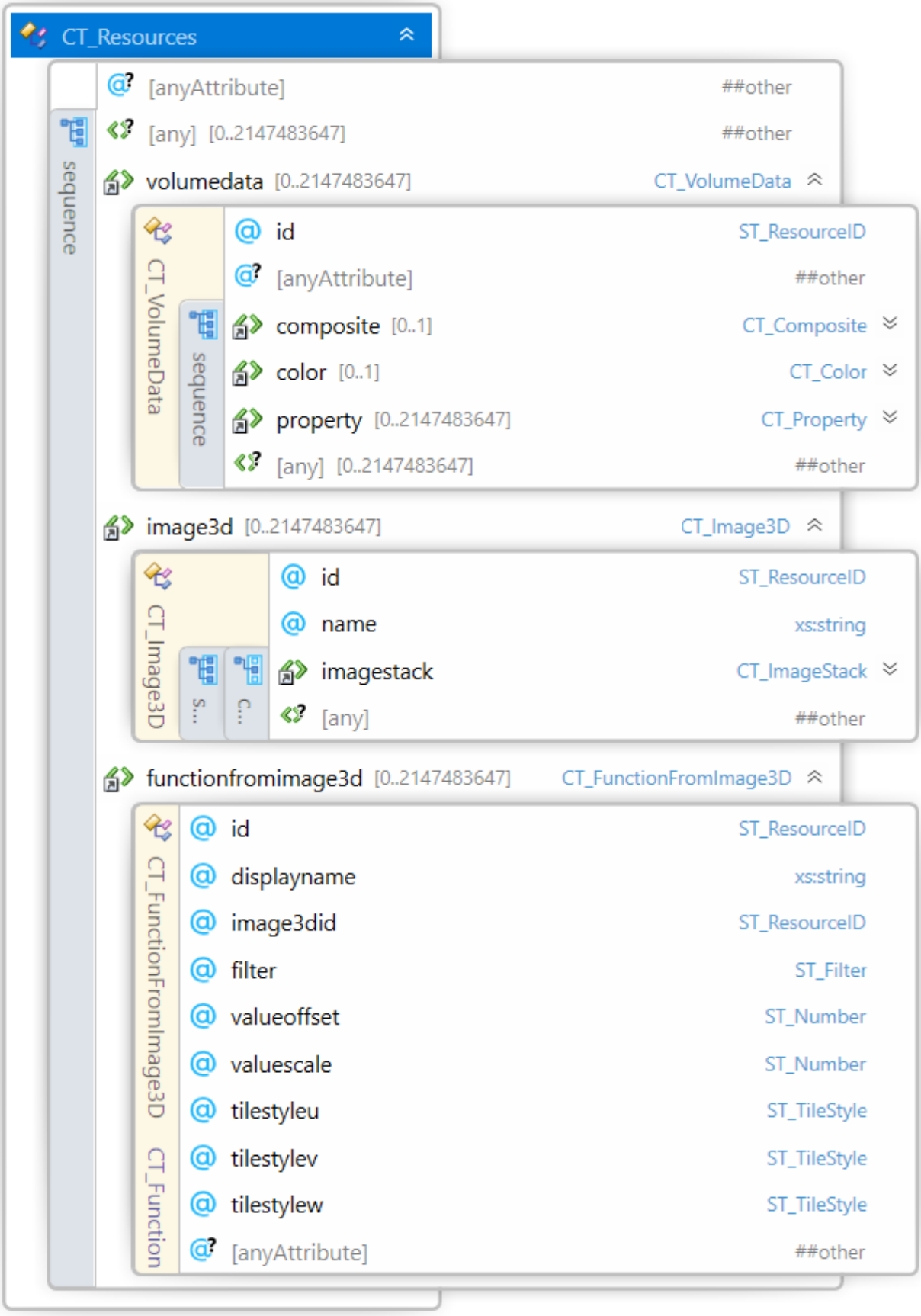
- <v:function> (including <v:functionfromimage3d> , <i:implicitfunction> and PrivateExtensionFunction) – e.g. distance / sampling nodes that take a mesh resource id.
- <v:levelset> – requires meshid to define the evaluation domain.
- (Future) other volumetric/implicit resources that semantically depend on existing geometry.

In the core specification Section 3.4 the principle is stated: “Producers MUST define each element prior to referencing it elsewhere in the document…”. To avoid forcing producers either to break this principle with forward references or to duplicate geometry, this extension introduces the following BREAKING CHANGE when (and only when) the volumetric extension is declared as REQUIRED via the requiredextensions attribute on <model> :

1. Relaxed resource ordering: <object> elements are no longer constrained to appear only*after* all non-object resources. Within <resources> , <object> , <v:function> , <v:image3d> , <v:volumedata> , and <v:levelset> MAY appear in any order or be interleaved, subject to rule (3) below.
2. Consumer acceptance: Consumers that support this extension MUST accept any ordering of the above resource types and MUST NOT raise an error solely because an <object> precedes or interrupts the sequence of extension resources.
3. Define-before-use: Producers MUST continue to place a resource definition before its first reference.
4. Materials precedence unchanged: Core material/property group resources (e.g. <basematerials> or other extension material groups defined in the core choice) SHOULD still be defined before any <object> or volumetric resource that references them. This change does not relax material group ordering.

1.2. Resources

Element **<resources>**



All new elements defined by the Volumetric Extension live under the core element. The ordering shown is illustrative; the schema does not enforce ordering as these extension elements fall under the core spec's container.

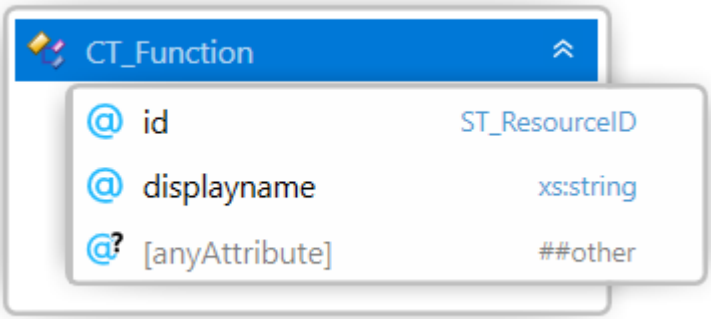
This extension adds the following resource types:

- <function> resources, which can host a <functionfromimage3d> , an <i:implicitfunction> (see Part II), or a PrivateExtensionFunction
- <image3d> resources for embedded volumetric imagery (via <imagestack>)
- <volumedata> resources that describe volumetric properties applied to shapes

Chapter 2. Functions and Function Types

2.1 Functions

Complex Type **CT_Function**



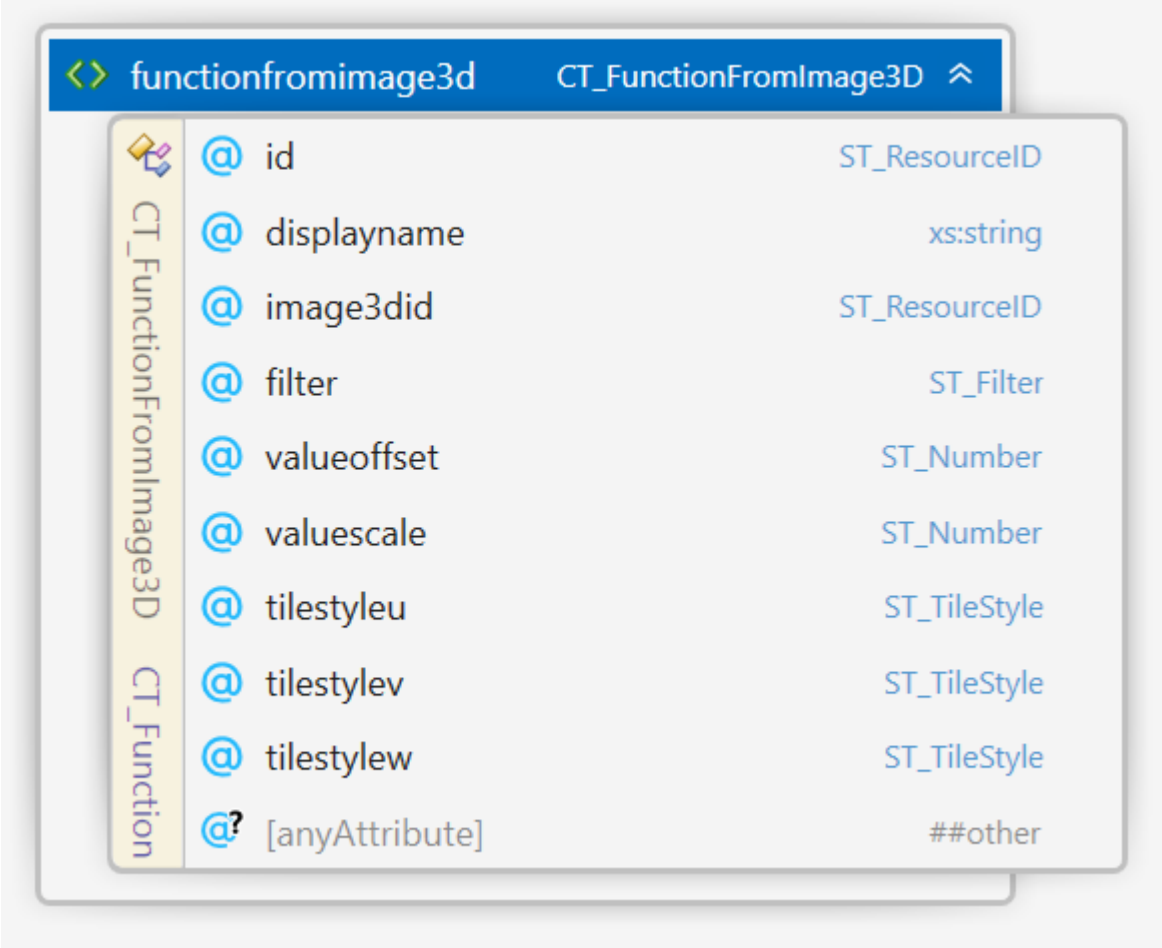
Name	Type	Use	Default	Annotation
id	ST_ResourceID	required		Specifies an identifier for this function resource.
displayname	xs:string			Function resource name used for annotations purposes.

Volumetric data is created with functions that are evaluatable for at given model position. Each function element is assumed to represent a method which can be evaluated within the model being described. Functions have input arguments and output arguments, each argument MUST be one of the supported datatypes enumerated in Chapter 2.

function is a container for one of three distinct function types: <functionfromimage3d> , <implicitfunction> or a PrivateExtensionFunction. The only function type that MUST be supported for the volumetric extension is <functionfromimage3d> which requires an Image3d resource.

2.2 FunctionFromImage3D

Element **<functionfromimage3d>**



Name	Type	Use	Default	Annotation
id	ST_ResourceID	required		Specifies an identifier for this function resource.
displayname	xs:string			Function resource name used for annotations purposes.
image3did	ST_ResourceID	required		Specifies an identifier for the image3d resource that this function uses during evaluation.
valueoffset	xs:double	optional	0.0	Specifies a numerical offset for the samples values
valuescale	xs:double	optional	1.0	Specifies a numerical scaling of the sampled values
filter	ST_Filter		linear	"linear" or "nearest" neighbor interpolation.
tilestyleu	ST_TileStyle		wrap	Determines the behavior of the sampler for texture coordinate u outside the [0,1] range.
tilestylev	ST_TileStyle		wrap	Determines the behavior of the sampler for texture coordinate v outside the [0,1] range.
tilestylew	ST_TileStyle		wrap	Determines the behavior of the sampler for texture coordinate w outside the [0,1] range.

Elements of type <functionfromimage3d> define a function which can be sampled at any point in space from values on a voxel grid defined in the <image3d> element. The function is evaluated by sampling the image3d at the UVW coordinates of the model position. The UVW coordinates are determined by the filter-rule and the tilestyle attributes of the <functionfromimage3d> -element.

Note: The UVW coordinates are normalized to (0,0,0) at the left-front-bottom corner of the voxel grid and (1,1,1) at the right-back-top corner of the voxel grid. So to map the volume data to an object the producer has to provide a transformation matrix that maps the object coordinate system to the normalized coordinates of the <functionfromimage3d> .

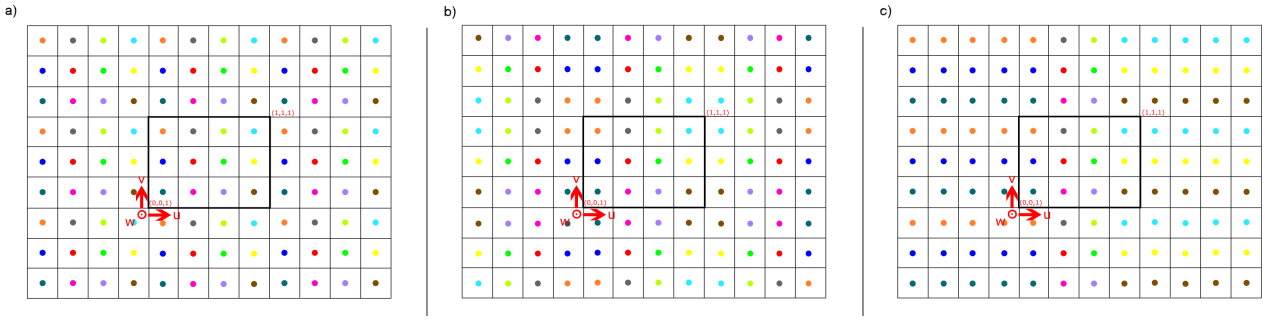
To simplify parsing, producers MUST define <image3d> -elements prior to referencing them via image3did in a <functionfromimage3d> -element.

tilestyle-u, -v or -w:

MUST be one of "wrap", "mirror" or "clamp". This property determines the behavior of the sampler of this <functionfromimage3d> for 3d texture coordinates (u,v,w) outside the [0,1]x[0,1]x[0,1] cell. The different modes have the following interpretation (for s = u, s = v, or s = w):

1. "wrap" assumes periodic texture sampling, see Figure 2-1 a). A texture coordinate s that falls outside the [0,1] interval will be transformed per the following formula: s' = s – floor(s)
2. "mirror" means that each time the texture width or height is exceeded, the next repetition of the texture MUST be reflected across a plane perpendicular to the axis in question, see Figure 2-1 b). This behavior follows this formula: s' = 1 - abs(s - 2 * floor(s/2) - 1)
3. "clamp" will restrict the texture coordinate value to the [0,1] range, see Figure 2-1 c). A texture coordinate s that falls outside the [0,1] interval will be transformed according to the following formula: s' = min(1, max(0,s))

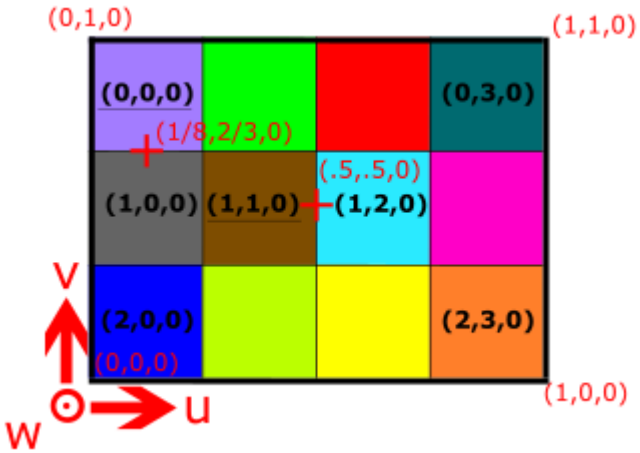
Figure 2-1: Illustration of different tilestyles. a) tilestyle wrap illustrated throughout the second <imagesheet> . b) tilestyle mirror illustrated throughout the second <imagesheet> . c) tilestyle clamp along the u-direction illustrated throughout the second <imagesheet>



filter: The filter attribute defines the interpolation method used when a <functionfromimage3d> is being sampled. This is illustrated in Figure 2-4.

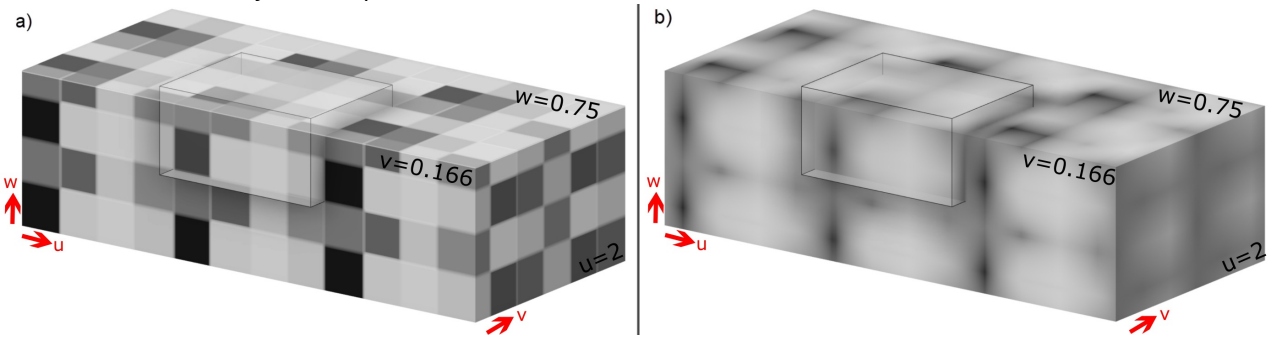
- If the interpolation method of an element of type <functionfromimage3d> is "nearest", sampling it at an arbitrary (u,v,w) returns the floating point value defined by the closest point (u',v',w') to (u,v,w) which transforms back to a voxel center in the 3D image resource. If a coordinate u,v, or w maps exactly at the middle between to voxel centers, sampling (u,v,w) should return the floating point value defined by the voxel center with the lower index value of the two voxel centers in question.

Figure 2-3: voxel lookup using filter method "nearest" neighbor: sampling at uvw=(1/8,2/3,0) evaluates the voxel with index-triple (0,0,0) (not (1,0,0)), and sampling at (u,v,w)=(0.5,0.5,0) evaluates the voxel with index-triple (1,1,0) (not (1,2,0)).



- If the interpolation method of an element of type <functionfromimage3d> is "linear", sampling it at an arbitrary (u,v,w) returns the floating point defined by trilinearly interpolating between the eight point coordinates defining a box that contains the arbitrary (u,v,w), which transforms back to voxel centers in the 3D image resource.

Figure 2-4: filter attributes "nearest" (a) and "linear" (b). The greyscale channel ("Y") of the image3d of Figure 3-1 is reused in this example. The region shown is clipped at w=0.75, v=1/6 and u=2. The grey wireframe box indicates the UVW unit box. The tilestyle is "wrap" in all directions.



valueoffset and valuescale :

The values V' sampled from the <image3d> are linearly scaled via valueoffset and valuescale giving a sampled value V'' = V'*valuescale + valueoffset .

A <functionfromimage3d> is a container for an image3d which is evaluatable. In contrast to implicit functions, the inputs and outputs of a <functionfromimage3d> are fixed and are not defined in the markup.

The outputs can be referenced by <volumedata> elements (e.g. <color>) or <levelset> using the channel attribute. In the implicit namespace a <functionfromimage3d> can be referenced by a functionCall node in the same way as an implicit function with the listed inputs and outputs.

A <functionfromimage3d> has the following input and outputs:

Inputs:

Identifier	Type	Description
pos	vector	UVW coordinates of the point to be evaluated. Points outside the range from (0, 0, 0) to (1 , 1, 1) will be mapped according to the tile style

The output values are in the range from 0 to 1. Please seeChapter 3.2 for more information on the input pixel layouts.

Outputs:

Identifier	Type	Description
color	vector	Vector containing the RGB values (x=red, y=green, z=blue), alpha is ignored
red	scalar	Scalar containing the red value
green	scalar	Scalar containing the green value
blue	scalar	Scalar containing the blue value
alpha	scalar	Scalar containing the alpha value

The appearance of color and red, green, blue might seem redundant, but allows to also use the output directly as a vectorial field.

Example Usage:

```
<v:image3d id="2">
  <v:imagestack rowcount="821" columncount="819" sheetcount="11">
   <v:imagesheet path="/volume/layer_01.png"/>
   ...
  </v:imagestack>
 </v:image3d>
<v:functionfromimage3d id="3" displayname="function from image3d" image3did="2" valueoffset="0" valuescale="1400" tilestyleu="wrap"
tilestylev="clamp" tilestylew="mirror" filter="linear" />
...
<v:volumedata id="3">
   <v:property name="Temp" transform="0.01 0 0 0 0.01 0 0 0 0.01 0.5 0.5 0.5" functionid="3" channel="red"/>
</v:volumedata>
<object id="4">
   <mesh volumeid="3">
      <vertices>
         ...
      </vertices>
      <triangles>
         ...
      </triangles>
   </mesh>
</object>
```

2.3 PrivateExtensionFunction

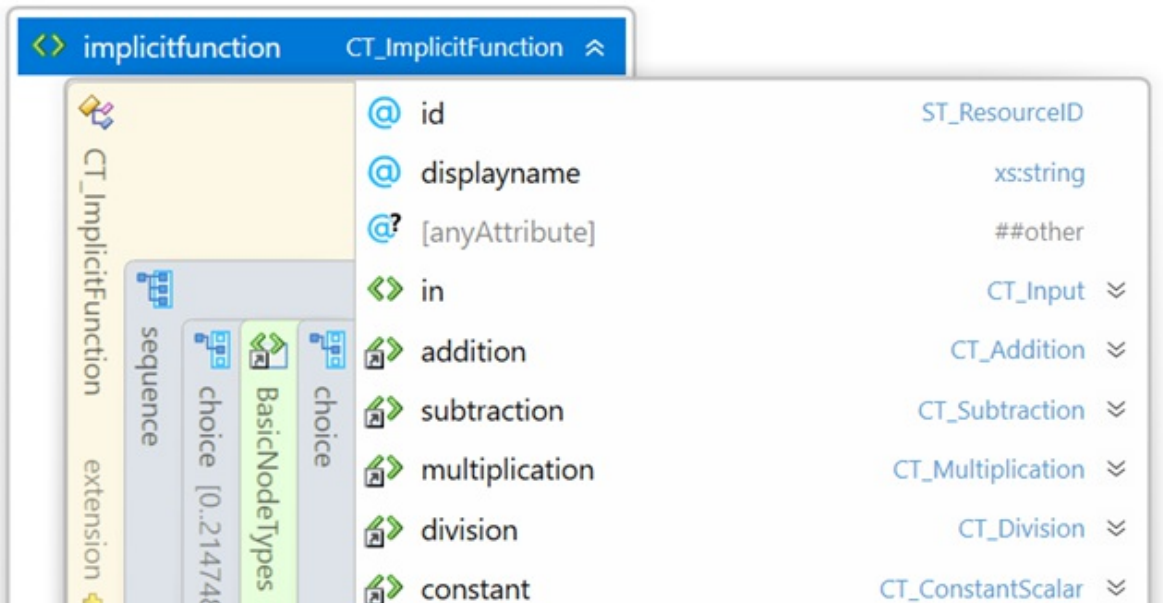
Element **<PrivateExtensionFunction>**

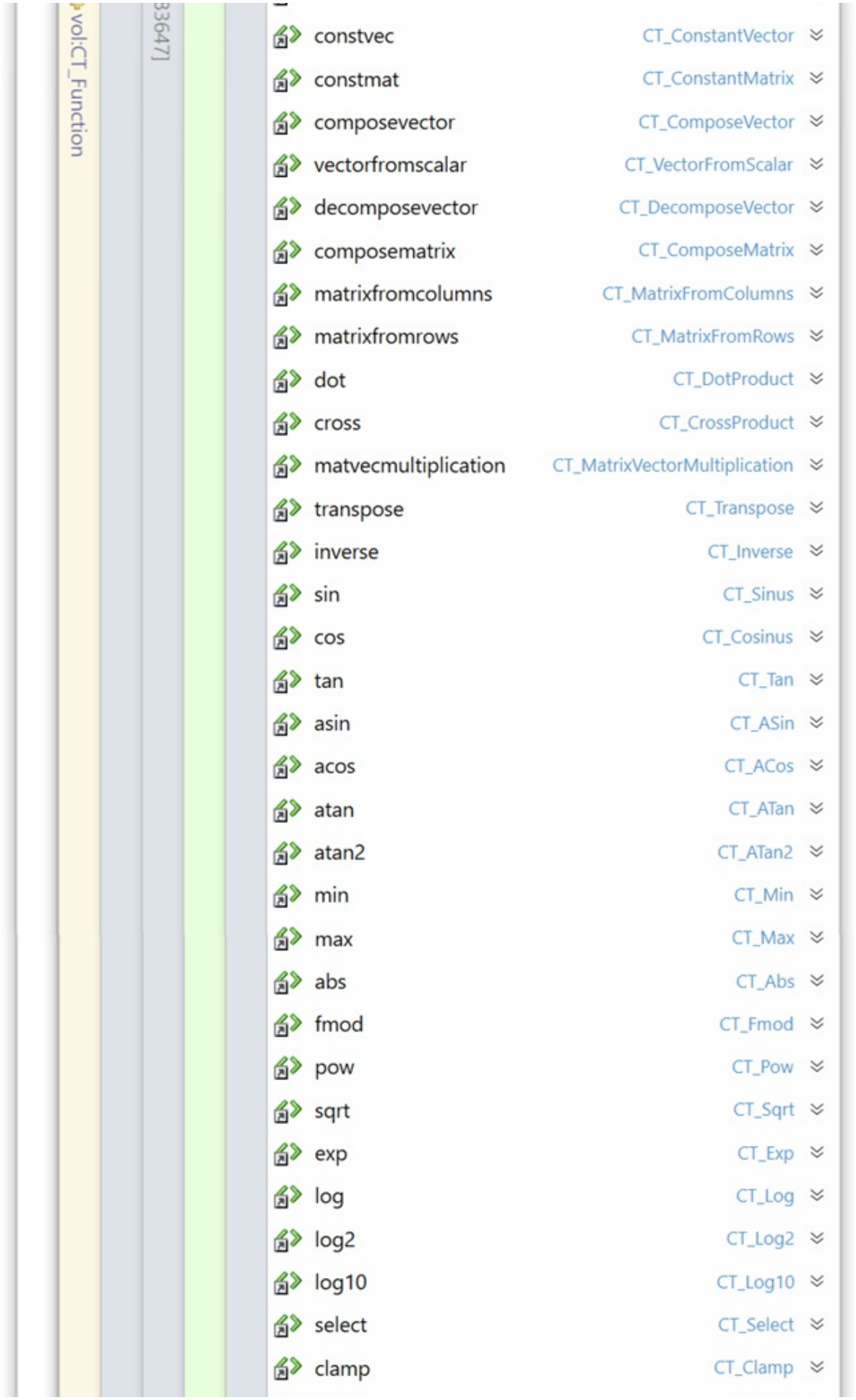
Name	Type	Use	Default	Annotation
id	ST_ResourceID	required		Specifies an identifier for this function resource.
displayname	xs:string			Function resource name used for annotations purposes.
xmlns	ST_namespace	required		Specifies the namespace of the function.

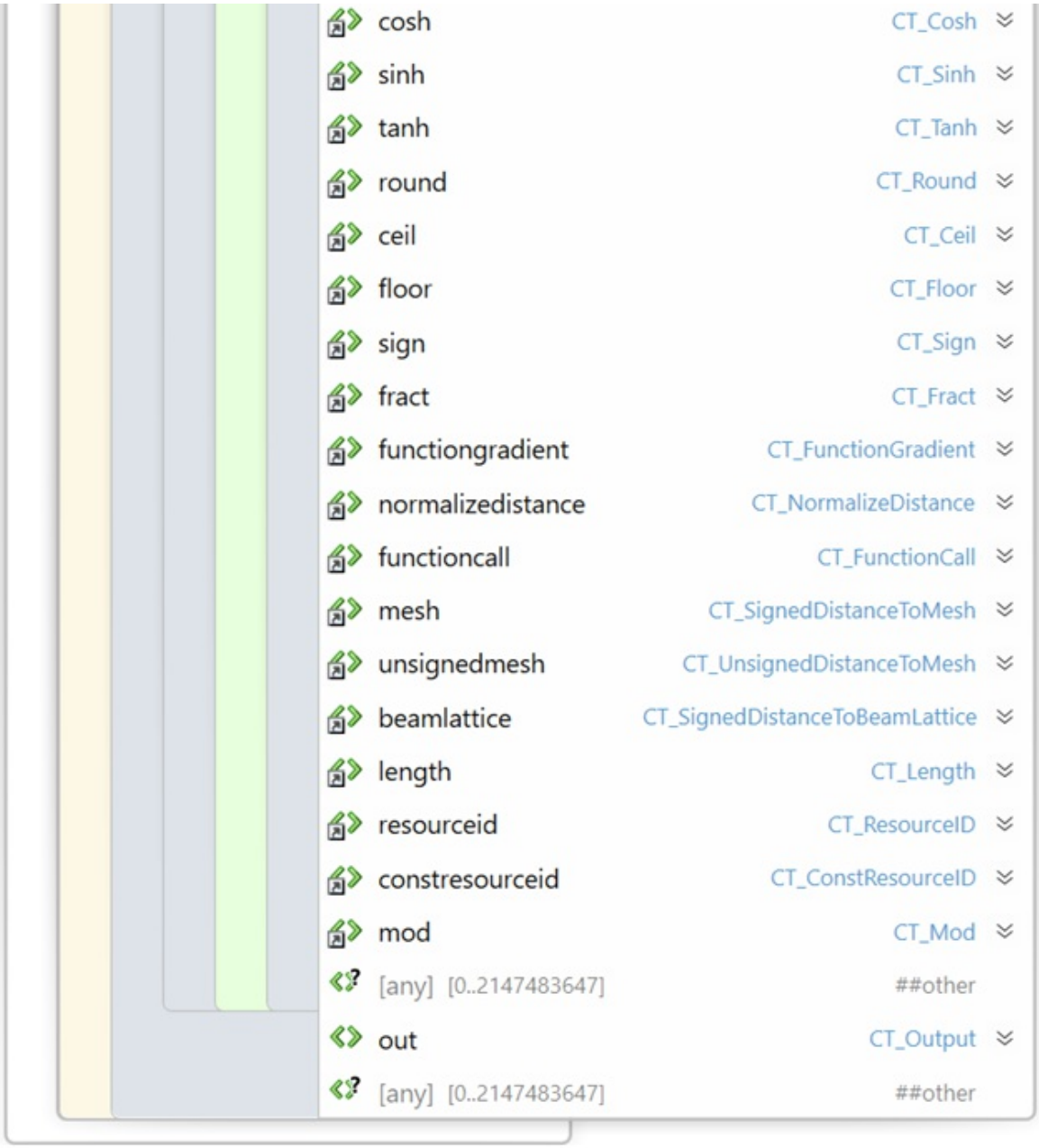
PrivateExtensionFunction is an OPTIONAL function type to support. This function can take either a <scalar> or <vector> input and returns either a <scalar> or <vector> . The intent of this function type is to allow users to extend the volumetric specification for custom functionality that is not possible with the existing functions.

2.4 ImplicitFunction

Element **<i:implicitfunction>**




vol:CT_Function
33647]
constvec CT_ConstantVector
constmat CT_ConstantMatrix
composevector CT_ComposeVector
vectorfromscalar CT_VectorFromScalar
decomposevector CT_DecomposeVector
composematrix CT_ComposeMatrix
matrixfromcolumns CT_MatrixFromColumns
matrixfromrows CT_MatrixFromRows
dot CT_DotProduct
cross CT_CrossProduct
matvecmultiplication CT_MatrixVectorMultiplication
transpose CT_Transpose
inverse CT_Inverse
sin CT_Sinus
cos CT_Cosinus
tan CT_Tan
asin CT_ASin
acos CT_ACos
atan CT_ATan
atan2 CT_ATan2
min CT_Min
max CT_Max
abs CT_Abs
fmod CT_Fmod
pow CT_Pow
sqrt CT_Sqrt
exp CT_Exp
log CT_Log
log2 CT_Log2
log10 CT_Log10
select CT_Select
clamp CT_Clamp



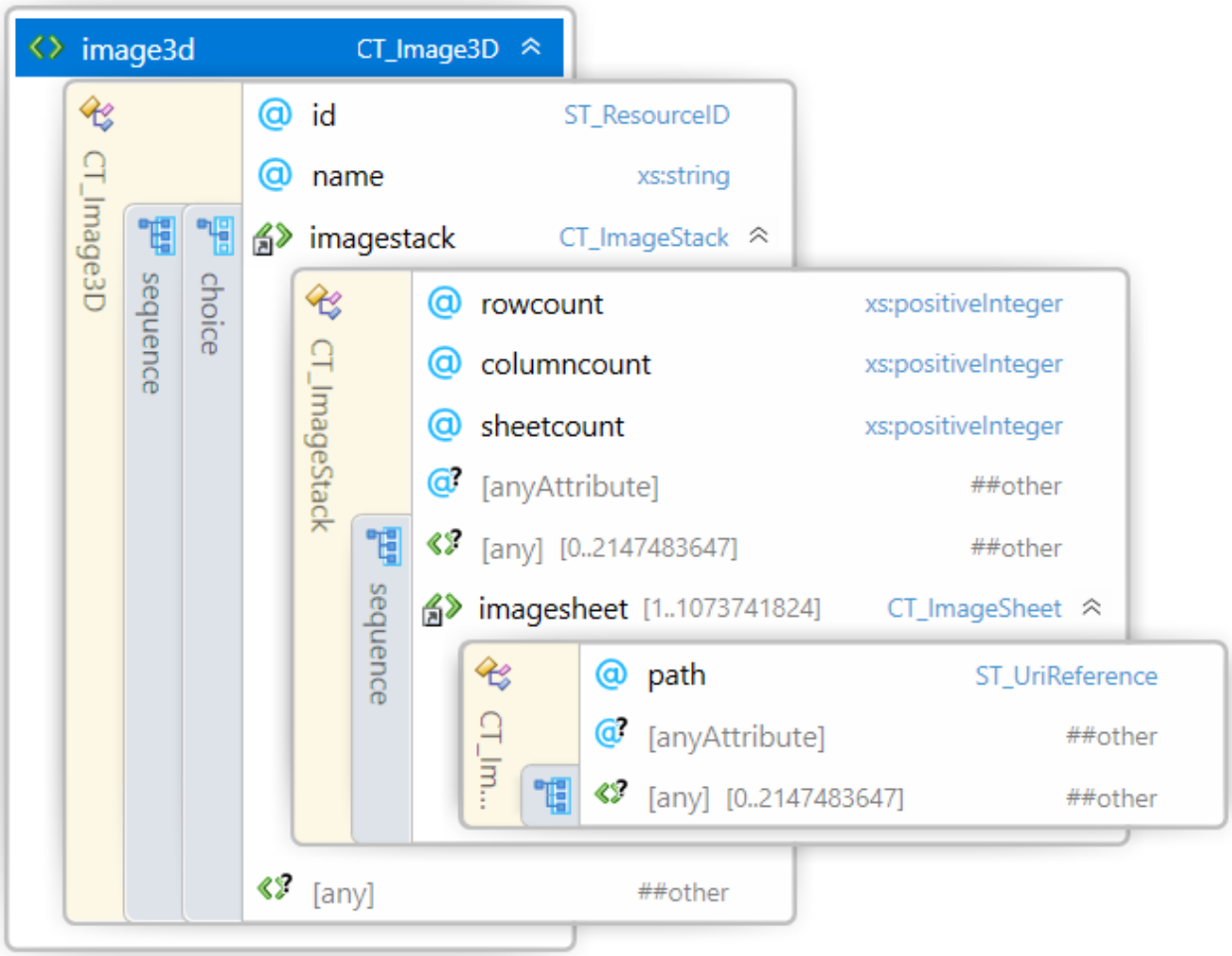
Name	Type	Use	Default	Annotation
id	ST_ResourceID	required		Specifies an identifier for this function resource.
displayname	xs:string			Function resource name used for annotations purposes.
xmlns	ST_namespace	required	implicit	Specifies the namespace of the function.

ImplicitFunction is an OPTIONAL function type to support for the Volumetric specification in the implicit namespace. The function requires an input DataType and an output DataType.

Chapter 3. 3D Image

3.1 3D Image

Element **<image3d>**



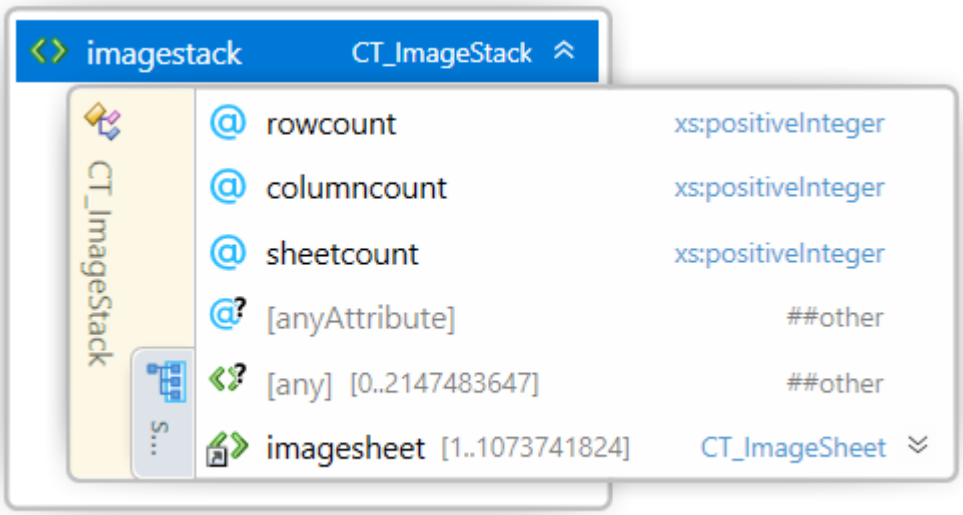
Name	Type	Use	Default	Annotation
id	ST_ResourceID	required		Specifies an identifier for this image3d resource.
name	xs:string			3d image resource name used for annotations purposes.

Volumetric data can be encoded as 3d images that consist of voxels. Each <image3d> element is assumed to represent a finite voxel grid from which data can be sampled.

<image3d> is a container for different representations of voxeldata. This specification defines only the <imagestack> -elements. Later versions of this specification might provide alternative child elements for the <image3d> element.

3.2 ImageStack

Element **<imagestack>**



Name	Type	Use	Default	Annotation
rowcount	xs:positiveinteger	required		Number of pixel rows in all child <imagesheet> -elements.
columncount	xs:positiveinteger	required		Number of pixel columns in all child <imagesheet> -elements.
sheetcount	xs:positiveinteger	required		Number of <imagesheet> -elements within this <imagestack> element.

Volumetric images can be embedded inside a 3MF file using groups of PNG images that represent a stack of images.

All <imagesheet> -elements within an imagestack MUST have the same number of rows and columns that is specified in the rowcount and columncount-attributes, respectively. rowcount, columncount and sheetcount MUST not exceed 1024^3, each. The total number of voxels MUST be limited by 1024^5. There MUST be exactly sheetcount <imagesheet> -elements under <imagestack> that are implicitly ordered starting with index 0.

Imagestack objects, and thus all underlying <imagesheet> elements, MUST follow one of the input pixel layouts shown in the table below. All imagesheets within an imagestack MUST have the same input pixel layouts, and each channel MUST have the same bit-depth across all imagesheets. Pixel values sampled from a PNG file with a bitdepth of N bits will be normalized to pixelvalue / (2^N-1) , i.e. a fully separated channel is normalized to 1, the minimum sampled value is normalized to 0.

The following table shows the logical interpretation of sampling the "R", "G", "B" or "A"-channel depending on the input pixel layouts. The meaning of symbols is as follows: R – red, G – green, B – blue, A – alpha, Y – greyscale.

Input pixel layout				
RGBA	R	G	B	A
RGB	R	G	B	1
YA	Y	Y	Y	A
Y	Y	Y	Y	1

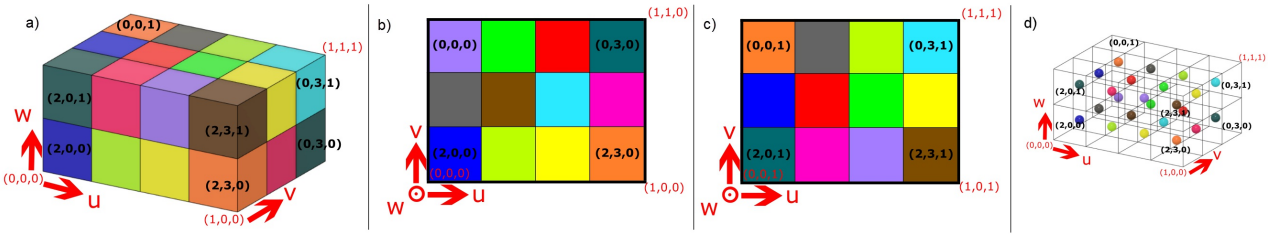
For example, if a function output from <functionfromimage3d> in a 3MF-file that maps to the R(ed) channel, but the referenced image is only monochromatic, then the greyscale channel is interpreted as the R color channel. Similarly, color values sampled from a monochromatic image are interpreted as if all "R", "G", "B" color channels share the same greyscale value. If there is no alpha channel present in the image, the highest possible value 1 MUST be used.

The <imagestack> -element defines a voxel grid of values (e.g. RGB, grey-Alpha, grey) values distributed in a cuboid ({0,1,...,rowcount-1} x {0,1,...,columncount-1} x {0,1,...,sheetcount-1}). The left-front-bottom corner of this grid corresponds to the (0,0,0)-UVW coordinate when this 3D Image is being sampled, whereas the right-back-top corner corresponds to the (1,1,1) UVW-coordinate. Each <imagesheet> corresponds to one PNG-file in the package. Figure 2-1 a) illustrates a voxel grid with rowcount=3 , columncount=4 and sheetcount=2 voxels. Voxel indices are shown as bold black triple, the UVW-coordinate values as red triples. Figure 3-1 b) illustrates the voxel indices and the UVW-values throughout the first <imagesheet> , Figure 3-1 c) illustrates these quantities throughout the second <imagesheet> . A voxel index triple (i,j,k) corresponds to a voxel with rowindex i , columnindex j and sheetindex k .

Note: The columnindex (j) relates to the UVW-coordinate U , whereas the rowindex i relates to the UVW-coordinate V . This definition is in line with the Materials and Properties specification https://github.com/3MFConsortium/spec_materials/blob/1.2.1/3MF%20Materials%20Extension.md#chapter-6-texture-2d.

The sampling rules for UVW values are determined by the filter-rule, and the behavior for UVW-values outside the unit-cube are determined by the tilestyle attributes of the <functionfromimage3d> .

Figure 3-1: Voxel indices and UVW-texture space of a sample voxel grid: a) shows a voxel grid of 3x4x2 voxels. b) shows a section view of the bottom voxels, c) shows a section view of the top voxels. The orange voxel at the right, front and bottom of a) has rowindex=2, columnindex=3 and sheetindex=0. d) shows the voxel centers of this configuration.



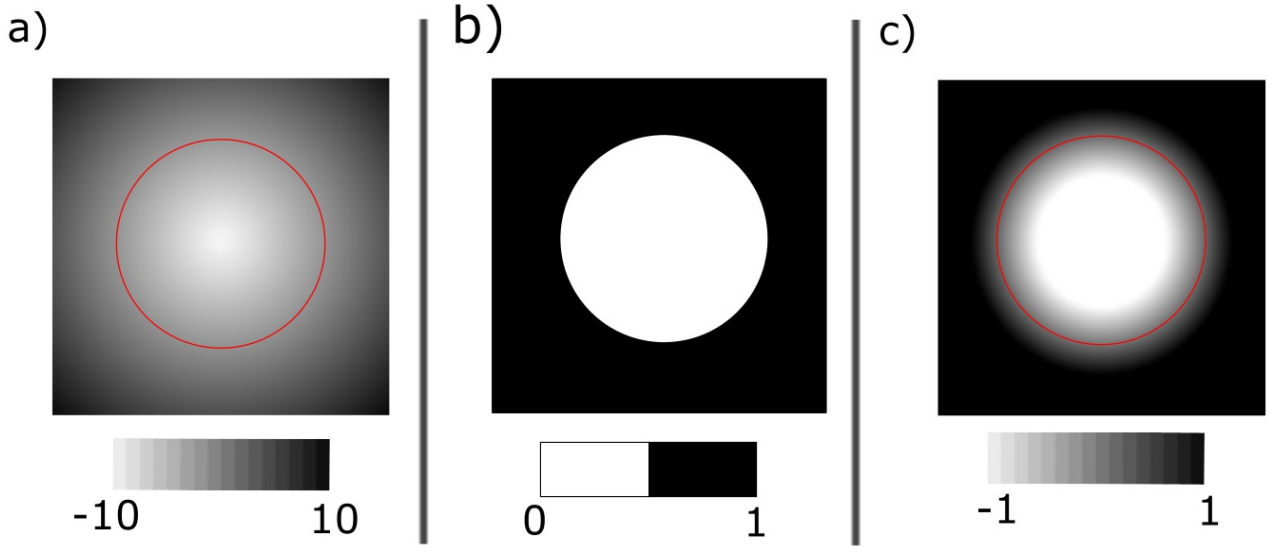
3.2.1 File Formats

PNG images can provide acceptable compression and bit-depth for the boundary-function, color information, material mixing ratios or arbitrary property information.

The following describes recommendations for the channel bit depth of PNG images used in this specification and is based on the nomenclature in the specification of the Portable Network Graphics (PNG, https://www.w3.org/TR/PNG) format.

- Color information, material mixing ratios and arbitrary properties can be deduced from PNG images with arbitrary channel depth. It is RECOMMENDED to store color into RGB-channels within a PNG.
- It is RECOMMENDED to store image information that will be used as levelset-function to represent a boundary in PNGs with one channel only. A typical approach to store this levelset information is to encode the signed distance field of the boundary of the object in this channel, or to limit the encoding to a narrow region around the boundary of the object. A different option is to deduce the levelset-function from a channel with binary values, i.e. from images of image type "greyscale" with bit-depth of 1 or an indexed-color with bit depths of 1, but with a very high spatial resolution.

Figure 3-2: 2D-slice through the levelset representation of a sphere of radius 5 illustrated as red circle. a) the greyscale values of the image represent the signed distance function in a box around the sphere. Values range from -10 to 10. b) High resolution image of the slice. White indicates a point that is inside the sphere, black outside the sphere. c) Image encoding the signed distance in a narrow band of thickness 2 around the boundary of the sphere. Values outside this band only indicate whether the point is inside or outside the sphere, as in b).



- Producers SHOULD store information for which they require high resolution in image channels with bit depth of 16. Most professional image editing tools and standard implementations of the PNG format support channels with 16 bit.

3.2.2 OPC package layout

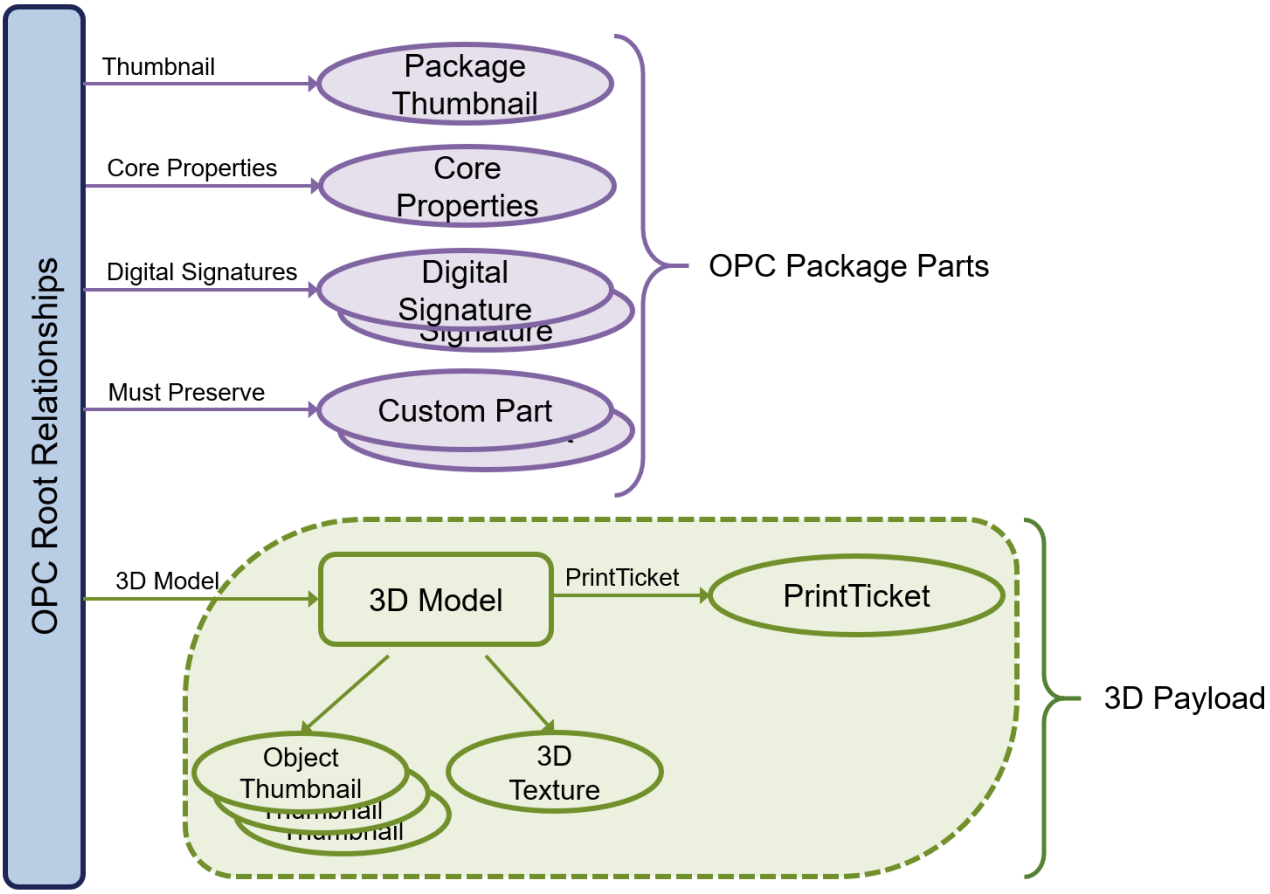
Note: Introductory information about the Open Packaging Conventions (OPC) can be found in the 3MF Core Specification (see https://github.com/3MFConsortium/spec_core/blob/1.3.0/3MF%20Core%20Specification.md#11-package).

It is RECOMMENDED that producers of 3MF Documents with the Volumetric Extension specification use the following part naming convention:

Paths of <imagesheet> SHOULD consist of four segments. "/3D/volumetric/" as the first two segments, the name of a <image3d> -element that references this <imagesheet> as third segment (for example "/3D/volumetric/mixingratios/", and the name of the imagesheet as last segment (for example "sheet0001.png"). Each part in the 3MF package that is referred to by the path of an <imagesheet> MUST be associated with the 3D Model part via the 3D Texture relationship.

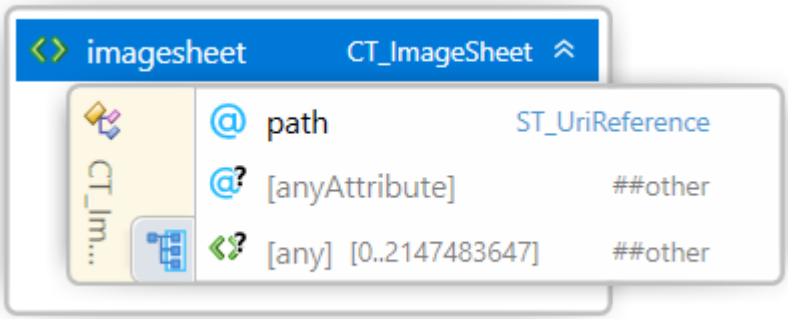
This implies that all parts for <imagesheet> in an imagestack SHOULD be located in the same OPC folder.

Figure 3-3: OPC package layout



3.2.3 3D Image Sheet

Element **<imagesheet>**



Name	Type	Use	Default	Annotation
path	ST_UriReference	required		Specifies the OPC part name (i.e. path) of the image data file

Each <imagesheet> element has one required attribute. The path property determines the part name (i.e. path) of the 2D image data (see chapter 6 of the Materials & Properties Extension specification for more information).

Note: Other file formats like OpenVDB, OpenEXR, or VTK offer similar functionality as a stack of PNGs and are more efficient at doing so. However, their use in a manufacturing environment is hard as these formats are conceptually more complex and harder to implement. Therefore, this specification relies on the human readable and conceptually simpler stack of PNGs. Later versions of this extension, or private extension of the 3MF format MAY use different 3D image formats to encode a volumetric data as different child elements of <image3d> and benefit from their advanced features. The remainder of this specification deals with the mapping of volumetric data onto mesh-objects in a 3MF file and giving this volumetric data a meaning for additive manufacturing processes. Adding a different data format for a voxel grid as child under <image3d> would not affect the remaining parts of this specification.

Chapter 4 LevelSet

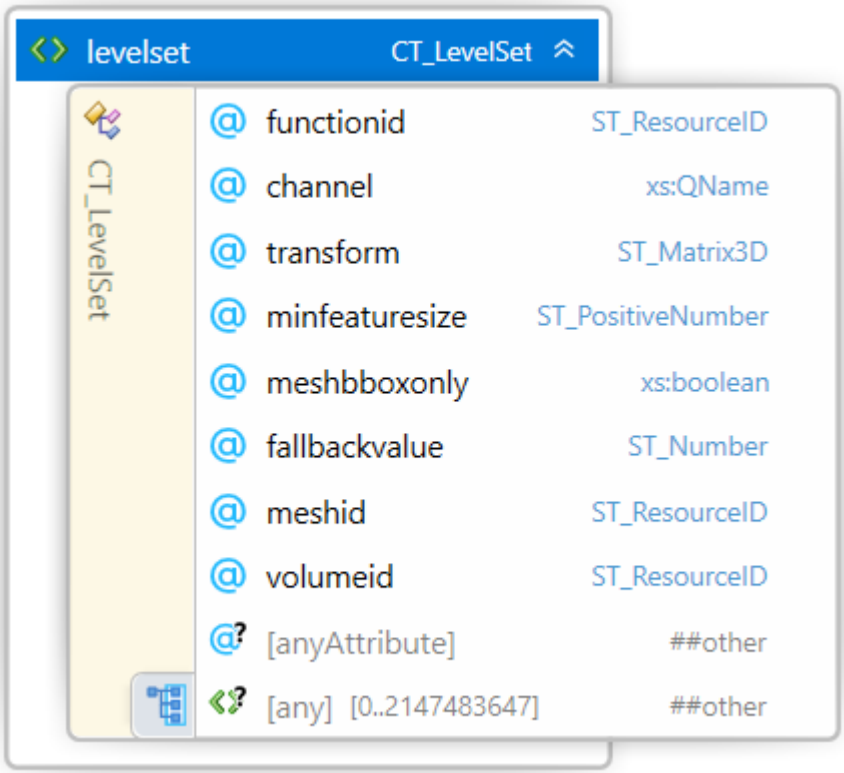
4.1.1 LevelSet element

A powerful application of Volumetric and Implicit modeling is the ability to define the shape of an object from volumetric information. Therefore we are introducing the concept of a **<levelset>** element which can be used to define the boundary of a shape using a levelset function. This is analogous to how a mesh defines the boundary between the inside and outside of the shape. In this case the mesh surface represents the surface of the levelset value equal to zero.

The child-element of the <levelset> -element references a functionID that must have a scalar output. This 'shape' represents the levelset function that MUST be evaluated to determine the actual shape of the object.

Since fields can be evaluated in an unbounded way, a closed mesh is required to enclose any levelset element to make the evaluation space bounded. For example a simple box that represents the bounding box of the geometry encoded in the meshid -attribute. There are cases where a producer would want to specify a bounding box for evaluation. In that case one can set the meshbboxonly -attribute to true. The <levelset> -element must be evaluated within the extents of the mesh referenced by the meshid -attribute.

Element **<levelset>**



Name	Type	Use	Default	Annotation
functionid	ST_ResourceID	required		ResourceID of the <function> that provides the boundary as a level set.
channel	xs:QName	required		Name of the output of the function to be used for the levelset. The output must be a scalar.
transform	ST_Matrix3D		Identity	Transformation of the object coordinate system into the <function> coordinate system.
minfeaturesize	ST_PositiveNumber		0	Specifies the minimum size of features to be considered in the boundary.
meshid	ST_ResourceID	required		ResourceID of the <mesh> that is used to define the evaluation domain of the level set.
meshbboxonly	xs:boolean		false	Indicates whether to consider only the bounding box of the mesh for the level set.
fallbackvalue	ST_Number		0	Specifies the value to be used for this data element if the output of the referenced function is undefined.
volumeid	ST_ResourceID			ResourceID of a <volumedata> resource to apply on the object.

The <levelset> -element is used to describe the interior and exterior of an object via a levelset function.

If meshbboxonly is set to true, the boundary is only intersected with the bounding box of the mesh. This allows the consumer to evaluate the boundary without computing the intersection with the mesh, otherwise the boundary is intersected with the mesh.

To simplify parsing, producers MUST define a <function> element prior to referencing it via the functionid attribute in a <levelset> element.

functionid:

ResourceID of the <function> that provides the boundary as a levelset. The function MUST have an input of type vector with the name "pos" and an output of type scalar that matches the name given as the channel attribute of the <levelset> element.

channel: Defines which function output channel is used as the levelset function. The channel MUST be of type scalar.

transform:

The transformation of the object coordinate system into the scalar field coordinate system. If the boundary-property of the enclosing mesh is being sampled at position (x,y,z) in the mesh's local object coordinate system, the referenced scalar field must be sampled at position (x',y',z') = T*(x,y,z) . See Figure 6-1 for an illustration of this transform in the sampling process.

minfeaturesize:

The minimum size of features to be considered in the boundary. This is used as a hint for the consumer to determine the resolution of the boundary. If the consumer is not able to resolve features of this size, it SHOULD raise a warning.

meshbboxonly:

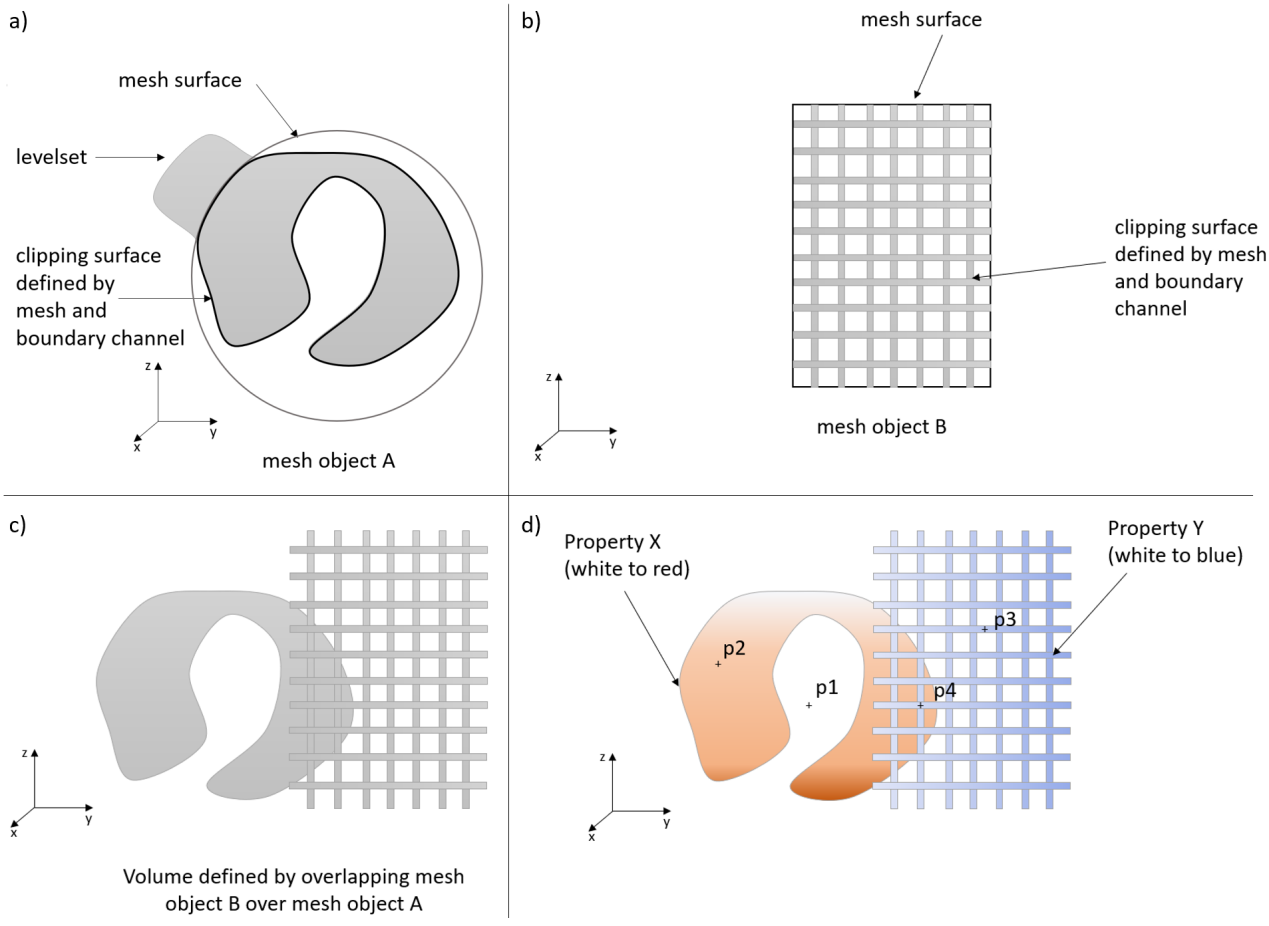
If this attribute is set to "true", the boundary is only intersected with the bounding box of the mesh. This allows the consumer to evaluate the boundary without computing the intersection with the mesh.

fallbackvalue:

Any undefined result MUST be evaluated as the value provided.

VolumeDetermination

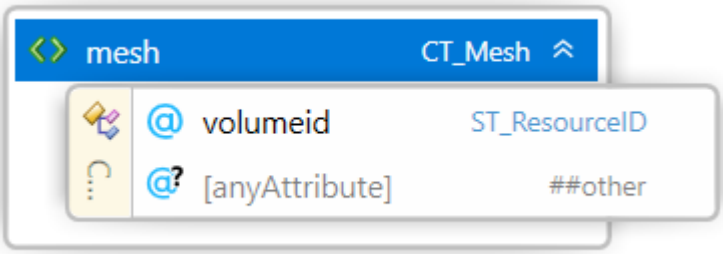
Figure 4-1: a) LevelSet A with a Mesh clipping surface. b) Mesh object B (rectangle) with <volumedata> child element Y. The mesh objects are defined in the order A-B. c) shows the volume defined by the overlapped mesh objects. d) shows <volumedata> child element X in object A, and <volumedata> child element Y in object B. The table lays out how these <volumedata> child elements are sampled at positions p1 to p4.



Chapter 5. Volumetric Data

5.1. Volumetric Data extension to Resources

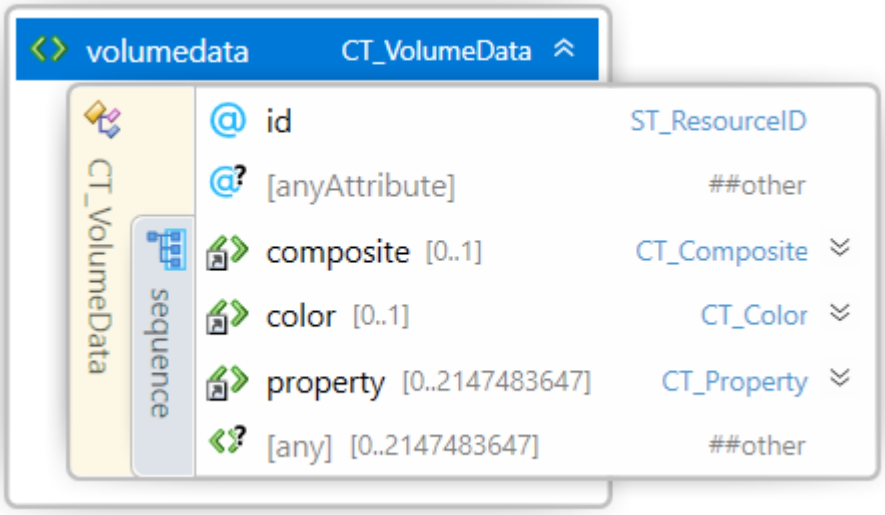
Element **<resources>**



All volumetric elements are defined under the core <resources> element. The volumetric <volumedata> element is a new OPTIONAL resource that can be referenced by a Shape (i.e. a <mesh> or <levelset> element).

5.2. Volumetric Data

Element **<volumedata>**



Name	Type	Use	Default	Annotation
id	ST_ResourceID	required		ResourceID of volume data by which it can be referenced.

The <volumedata> defines the volumetric properties in the interior of a Shape.

The child elements of the <volumedata> element reference a function that must match the signature requirements of the respective child element. Volumedata MUST only be referenced by an object type "mesh" or "levelset" unless explicitly allowed by shapes defined in other extensions. This ensures that the <volumedata> applies to a volume. Moreover, the volumedata-element MUST not be used in a mesh that is referenced as "originalmesh" by any other mesh. This excludes the possibility to implicitly mirror volumedata, which makes it easier to consume files with this extension.

The <volumedata> element can contain up to one <composite> child element, up to one <color> element, and up to 2^31-1 <property> elements.

The child elements modify the enclosing Shape by specifying color, material composition and other arbitrary properties of the Shape object.

To rationalize how this specification modifies the definition of geometry within a 3MF model, the concept of a "clipping surface" of a mesh with a <volumedata> element is introduced. The clipping surface is defined by the surface of the enclosing Shape. This implicitly takes into account any geometry defined by e.g. the beamlattices specification.

This clipping surface trims any volumetric data defined therein. Any data outside the clipping surface MUST be ignored. The geometry that should be printed is defined by the interior of the clipping surface.

Note Volumetric content is always clipped to the clipping surface of the shape that embeds it. If a property (color, composite or properties) defined at the surface of an object conflicts with the property within the object defined by this

extension, the surface property has precedence in the following sense: The consumer MUST define a surface layer with a thickness as small as possible to achieve the surface property on the outside of the object. Outside of this thin surface region, the volumetric property MUST be applied everywhere within the object.

Note Defining conflicting surface- and volumetric properties can be very powerful, e.g. if one wants to add a high resolution surface texture over a lower resolution volumetric color. However, producers MUST be aware that the thickness of this surface layer can vary between consumers depending on the manufacturing technology they employ.

The properties at surface regions that are not explicitly specified are instead given by the volumetric properties.

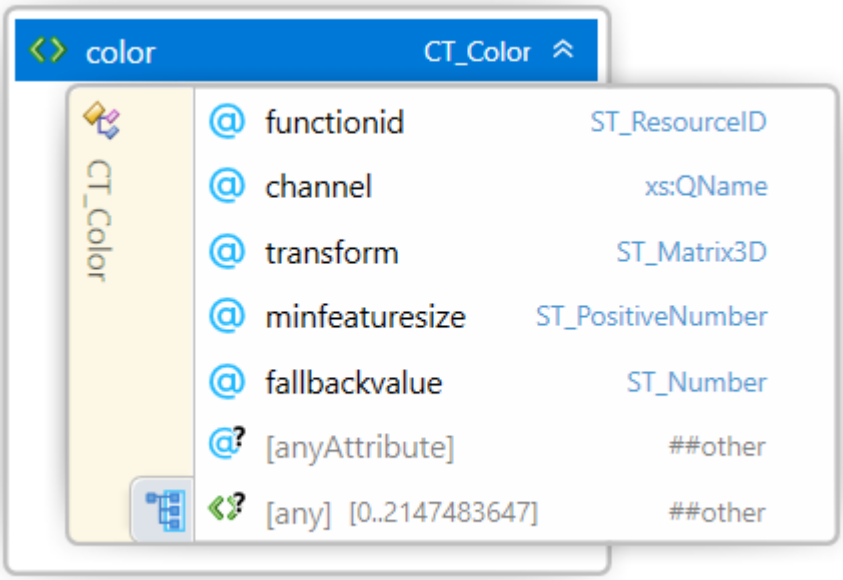
Conflicting properties must be handled as follows:

1. Producers MUST not define colors, materials or properties via child elements of the <volumedata> element that are impossible on physical grounds (e.g. non-conducting copper).
2. Consumers that read files with properties that cannot be realized due to limitations specific to them (e.g. a specific manufacturing device that does not support a material in a specific color), SHOULD raise a warning, but MAY handle this in any appropriate way for them. If there is an established process between Producer and Consumer, resolution of such conflicts SHOULD be performed e.g. via negotiation through printer capabilities and a print ticket.

Note: In the case where objects with different <volumedata> child elements overlap, only the <volumedata> child elements from the last object can be used. This makes sure that <volumedata> child elements of an overlapped object do not determine the value of any <volumedata> child elements of an overlapping object. Figure 4-1 illustrates this behavior.

5.2.1 Color element

Element **<color>**



Name	Type	Use	Default	Annotation
functionid	ST_ResourceID	required		Model Resource Id of the function providing the color
transform	ST_Matrix3D			Transformation of the object coordinate system into the coordinate system of the referenced function.
channel	xs:QName	required		Name of the function output to be used as color. The output must be of type vector.
minfeaturesize	ST_PositiveNumber		0	Hint for the minimum size of features.
fallbackvalue	ST_Number		0	Specifies the value to be used for this data element if the output of the referenced function is undefined.

To simplify parsing, producers MUST define the function referenced by functionid prior to the <color> -element.

The <color> element is used to define the color of the object. The color MUST be interpreted in linearized sRGB color space as defined in the Materials and Properties specification https://github.com/3MFConsortium/spec_materials/blob/1.2.1/3MF%20Materials%20Extension.md#12-srgb-and-linear-color-values.

The vector components x , y and z of the 3D vector field are interpreted as the "R", "G" and "B" channels of the color of the enclosing meshobject, respectively. If either channel evaluates to a value <0 or >1 it has to be truncated at 0 or 1, respectively.

This specification does not capture well the properties for semi-transparent, diffusive materials. This specification is useful for defining parts with non transparent, opaque materials, e.g. for indicating wear and tear, sectioning the models and printing with non transparent materials.

transform:

The transformation of the object coordinate system into the coordinate system of the function (e.g., normalized coordinates for <functionfromimage3d>). If this <color> >-element is being sampled at position (x,y,z) in the mesh's local object coordinate system, the 3D vector field must be sampled at position (x',y',z') = T*(x,y,z) .

channel

Name of the function output to be used as color. The output must be of type vector.

minfeaturesize:

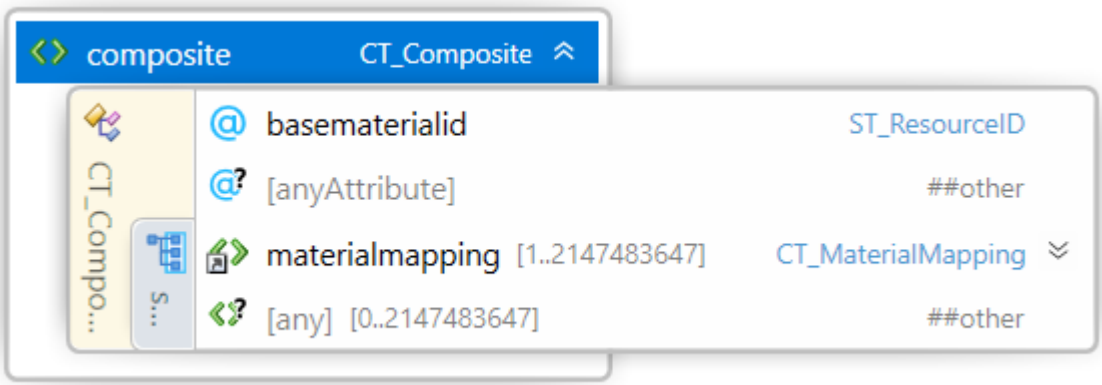
The minimum size of features to be considered in the color. This is used as a hint for the consumer to determine the resolution of the color estimation. It might also be used to determine the level of supersampling required, if the printer cannot reproduce the resolution. If the consumer is not able to resolve features of this size, it SHOULD raise a warning.

fallbackvalue:

Any undefined result MUST be evaluated as the provided value. The fallback value is specified as a scalar and MUST be applied across the result vector element-wise.

5.2.2 Composite element

Element **<composite>**



Name	Type	Use	Default	Annotation
basematerialid	ST_ResourceID	required		ResourceID of the <basematerials> element that holds the <base> -elements referenced in the child <materialmapping> -elements.

The <composite> element describes a mixing ratio of printer materials at each position in space. The CONSUMER can determine the halftoning, mixing or dithering strategy that can be used to achieve these mixtures.

This element MUST contain at least one <materialmapping> element, which will encode the relative contribution of a specific basematerial to the material mix.

The number of <base> -elements in the <basematerials> element referenced by a <composite> element MUST equal the

number of <materialmapping> -elements in the <composite> element. To simplify parsing, producers MUST define the referenced <basematerials> -element prior to referencing it via the basematerialid in a <composite> -element.

Producers MUST NOT create files where the sum of all values in its child <materialmapping> -elements is smaller than 10^-5 . If the total is smaller than this threshold, the mixing ratio is up to the consumer.

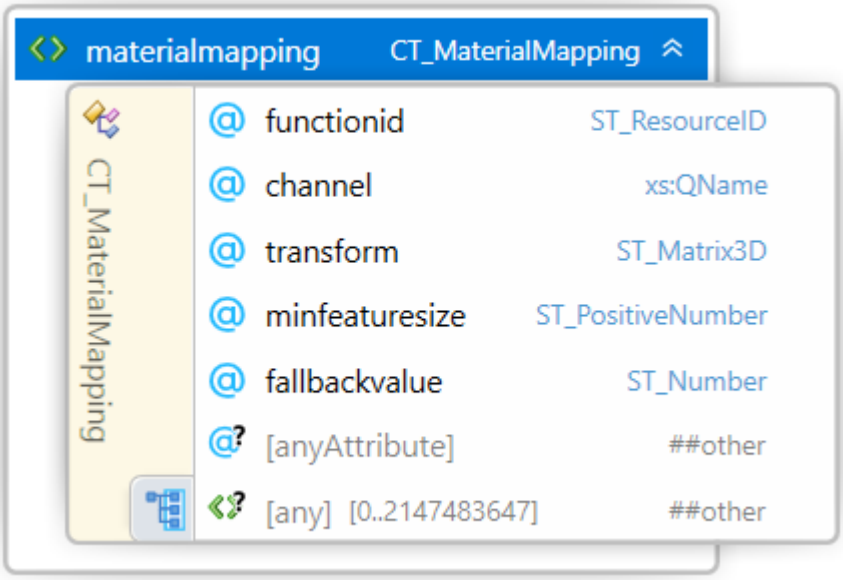
- If there are N materials, then the mixing ration of material i at point X is given by:

  value of channel i / sum(value of all N mixing contributions at point X)

The order of the -elements defines an implicit 0-based index. This index corresponds to the index defined by the <base> - elements in the <basematerials> -element of the core specification.

5.2.3 Material mapping element

Element **<materialmapping>**



Name	Type	Use	Default	Annotation
functionid	ST_ResourceID	required		ResourceID of the <function> providing the mixing contribution value for a material in the <basematerial> -element.
transform	ST_Matrix3D			Transformation of the object coordinate system into the <function> coordinate system
channel	xs:QName	required		Name of the function output to be used for the mixing contribution. The output must be a scalar.
minfeaturesize	ST_PositiveNumber		0	Hint for the minimum size of features.
fallbackvalue	ST_Number		0	Specifies the value to be used for this data element if the output of the referenced function is undefined.

The <materialmapping> element defines the relative contribution of a specific material to the mixing of materials in it's parent <composite> -element.

To simplify parsing, producers MUST define the referenced <function> -element prior to referencing it via the functionid in a <materialmapping> -element.

transform:

The transformation of the object coordinate system into the scalar field coordinate system. If any channel of a <materialmapping> is being sampled at position (x,y,z) in the mesh's local object coordinate system, the referenced scalar field must be sampled at position (x',y',z') = T*(x,y,z) .

channel:

Name of the function output to be used for the mixing contribution. The output must be a scalar. The value is clamped to the range [0,1].

minfeaturesize:

The minimum size of features to be considered in the mixing contribution. This is used as a hint for the consumer to determine the resolution of the mixing contribution. If the consumer is not able to resolve features of this size, it SHOULD raise a warning.

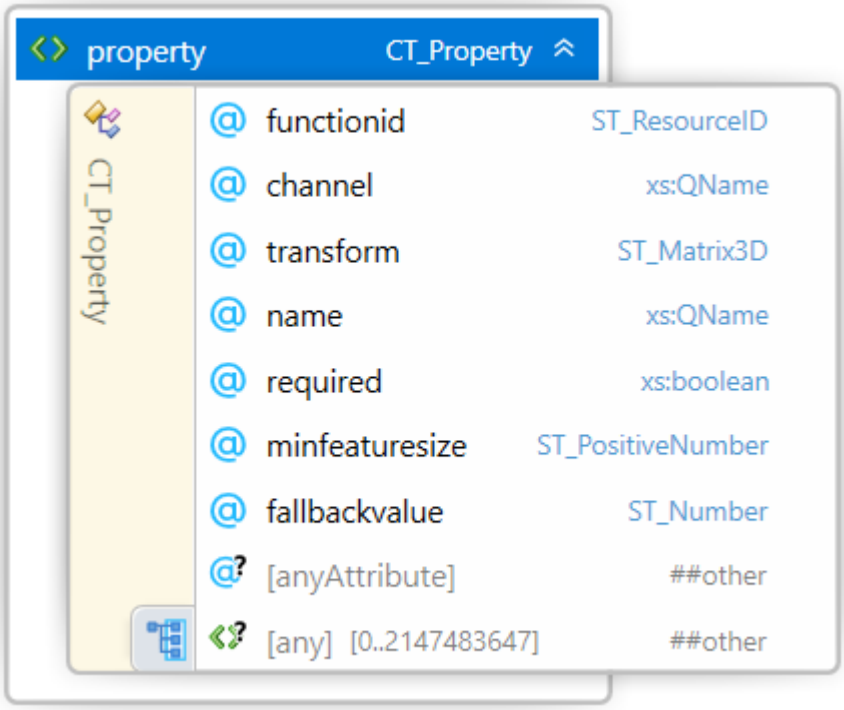
fallbackvalue:

If this attribute is set, any undefined result MUST be evaluated as the provided value.

If the sampled value of a <function> is <0 it must be evaluated as "0".

5.2.4 Property element

Element **<property>**



Name	Type	Use	Default	Annotation
functionid	ST_ResourceID	required		ResourceID of the <function> that provides the value of this property
transform	ST_Matrix3D			Transformation of the object coordinate system into the <function> coordinate system
channel	xs:QName	required		Name of the function output to be used for the property.
name	xs:QName	required		Contains either the name of the property, defined in a 3MF extension specification, or the name of a vendor-defined property. Either MUST be prefixed with a valid XML namespace name declared on the <model> element.
required	xs:boolean		false	Indicator whether this property is required to process this 3MF document instance.
fallbackvalue	ST_Number		0	Specifies the value to be used for this data element if the output of the referenced function is undefined.

The <property> element allows to assign any point in space a scalar or vectorial value of a freely definable property. This can be used to assign, e.g. opacity, conductivity, or translucency.

To simplify parsing, producers MUST define the referenced <function> -element prior to referencing it via the functionid in a <property> -element.

transform:

The transformation of the object coordinate system into the <function> coordinate system. If a <property> -element is being sampled at position (x,y,z) in the mesh's local object coordinate system, the referenced <function> must be sampled at position (x',y',z') = T*(x,y,z) .

channel:

Name of the function output to be used for the property. Note that the type of the output determines the type of the property.

This specification does not provide qualified names for such properties as part of the standard volumetric namespace. A later extension of the 3MF format might define such qualified names as part of a different extension specification or a later version of the volumetric extension specification. Producers that want to specify such properties now, SHOULD define a qualified name that can e.g. be called "http://www.vendorwwebsite.com/3mf/vendor13mfextension/2021/05". The specifications of private namespaces (that are not ratified by the 3MF Consortium) MUST be negotiated between producer and consumer of a 3MF file.

The names of <property> -elements MUST be unique within a <volumedata> . This name MUST be prefixed with a valid XML namespace name declared on the <model> element. The interpretation of the value MUST be defined by the owner of the namespace.

Note: The producer of a 3MF file is responsible for assembling the values in the <property> (and the referenced <function> such that sampling it in the print-bed coordinate system as a e.g. physical property, is sensible. This requirement is particularly important if the accumulated transformation T0 between print-bed coordinates and local object coordinates is not a rigid body transformation. (The transformation T0 of the print-bed coordinate system into the object coordinate system is given by the transform -attributes on the item and component -elements in the path that leads to this object in the build -hierarchy of the 3MF Core Specification (see https://github.com/3MFConsortium/spec_core/blob/1.3.0/3MF%20Core%20Specification.md#3431-item-element and https://github.com/3MFConsortium/spec_core/blob/1.3.0/3MF%20Core%20Specification.md#421-component, and Figure 6.1 in this document).

If the interpretation of a property might result in a conflict with the standard volumedata-elements (boundary, color, composite) the namespace-owner MUST specify a resolution to the conflict. A producer MUST NOT create files with properties that conflict with each other.

If a physical unit is necessary, the namespace owner MUST define a unique and unambiguous physical unit system for the namespace. The unit system SHOULD be metric.

If a <property> is marked as required , and a consumer does not support it, it MUST warn the user or the appropriate upstream processes that it cannot process all contents in this 3MF document instance. Producers of 3MF files MUST mark all volumetric <property> -elements required to represent the design intent of a model as required .

Chapter 6. Notes

6.1. Evaluation Graph

The elements in this specification form an acyclic directed graph when evaluating the value of any volumedata-subelement.

It is RECOMMENDED that calculations during evaluation of the graph are performed in at least single-precision floating-point arithmethic, according to IEEE 754.

6.2. Evaluation Process

Equipped with the language elements of this specification, one can recapitulate the core concepts with an overview of the sampling process.

Figure 6-1 illustrates the 3MF elements, the different coordinate systems and transforms between them when a <volumedata> element (in this case <color>) is sampled in the object coordinate space.

Figure 6-1 a) The object's color is sampled at position (+) in the print-bed coordinate system. The clipping surface is hinted at with a wireframe. The transformation T0 of the print-bed coordinate system into the object coordinate system is given by the transform -attributes on the item and component -elements in the path that leads to this object in the build -hierarchy of the 3MF Core Specification (see https://github.com/3MFConsortium/spec_core/blob/1.3.0/3MF%20Core%20Specification.md#3431-item-element and https://github.com/3MFConsortium/spec_core/blob/1.3.0/3MF%20Core%20Specification.md#421-component).

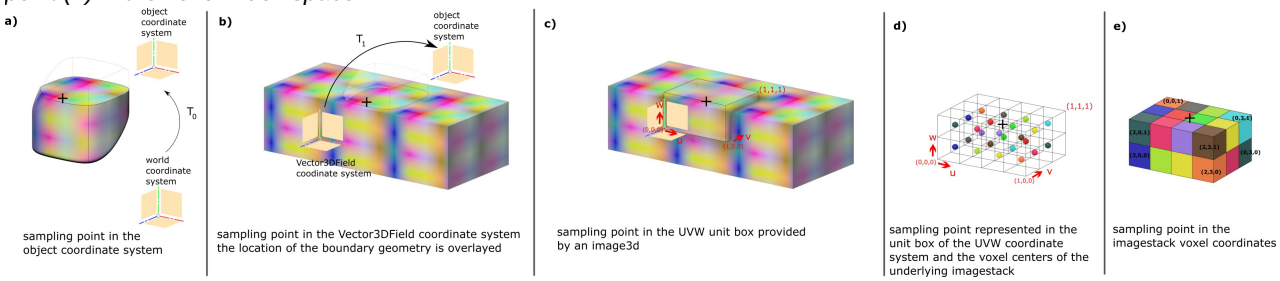
Figure 6-1 b) shows the <functionfromimage3d> underlying the color of the object. The sampling point is represented in the coordinate system of the <functionfromimage3d> . The transformation T1 from object coordinate space to <functionfromimage3d> coordinate system is given by the transform -element in the <color> -element. The original clipping surface from a) is only shown for illustration porpuses. It does not exist in the <functionfromimage3d> context. The color value sampled in this illustration directly originates from a <functionfromimage3d> element.

Figure 6-1 c) Shows the <functionfromimage3d> again. The unit box of the UVW coordinate system is shown as a wireframe. The transformation T2 between <functionfromimage3d> coordinate system and UVW space is given according to the transform -attribute of the <functionfromimage3d> element.

Figure 6-1 d) Shows the UVW coordinate space of the <image3d> -element and where the sampling point (+) is evaluated, and the UVW-locations to which the voxel centers of the underlying <imagestack> -element map.

Figure 6-1 e) illustrates where the sampling point (+) ends up in the voxel index space of the <imagestack> . The mapping of UVW to voxel indices in the <imagestack> -element is described in Chapter 2. 3D Image.

Figure 6-1: Illustration of the different coordinate systems and 3MF elements in the sampling process. a) the object to be sampled at position (+). b) A view into the <functionfromimage3d> . The original clipping surface from a) is only shown for illustration porpuses. c) Shows the <functionfromimage3d> again. The unit box of the UVW coordinate system is shown as a wireframe. d) The UVW coordinate space and the UVW-locations to which the voxel-centers map. e) The sampling point (+) in the voxel index space.



6.3. Limitations

This specification is limited in scope. Three noteworthy limitations are:

1. One cannot overlay a function over a meshless (composite) object, one has to duplicate the volume data elements at the object leaves.
2. The fields in this definition are limited to be scalar- (or 1D-vector-) valued and 3D-vector valued. Vectors of other dimensionality must be assembled by the producer / consumer using multiple scalar / 3D-vector fields.
3. It is not possible to assemble a field object that represents the RGB and alpha-channels from images. However one may assamble and blend data from RGBA-images explicitely by extracting the alpha channel into a scalar field and by using the alpha scalar field as a composition mask of two RGB 3d vector fields.

Part II. Implicit Extension

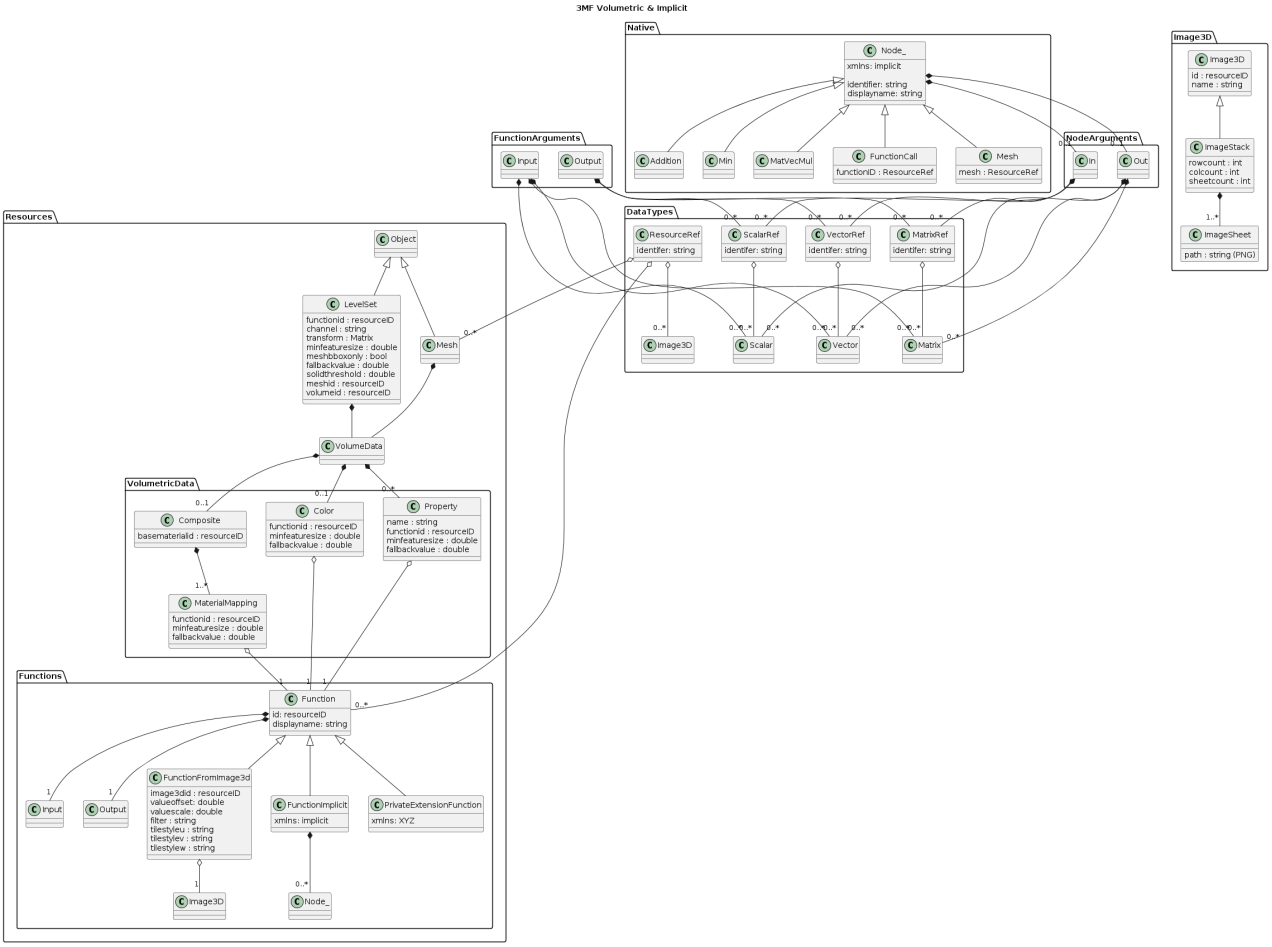
The implicit namespace extension enriches the volumetric extension by facilitating the use of closed form functions. These provide an alternative to <functionfromimage3d> for generating volumetric data.

The functions are members of volumetric data that define a field with arbitrary precision. These functions can be integrated with the existing children of volumedata (materialmapping, property, color), where they are defined at every point within the mesh or its bounding box. These functions are created via a connected node set. They link inputs and

outputs, and allow interaction with other resources.

Chapter 1. Overview of Implicit Additions

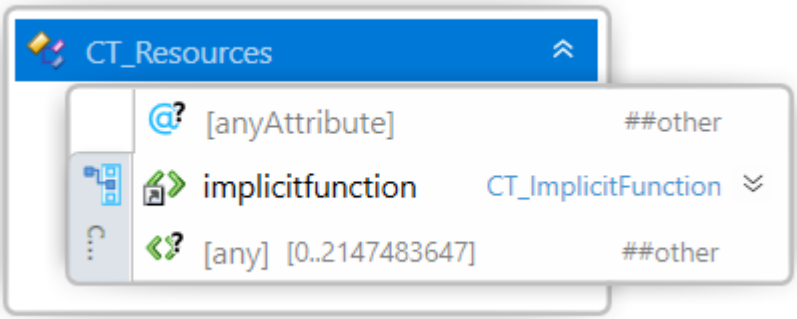
Figure 1-1: Overview of model XML structure of 3MF with implicit additions



Implicit adds <scalarref> , <vectorref> , <matrixref> , <resourceref> , <i:implicitfunction> and Native nodes. Optionally PrivateExtensionFunction can be defined.

1.1. Resources

Element **<resources>**



All new elements defined by the Implicit Extension also live under the core <resources> element.

This extension adds <i:implicitfunction> resources in the implicit namespace, whose contents define a node graph of native operations and references.

Chapter 2. DataTypes

The volumetric extension of 3MF, defines 4 new datatypes that are used for definition of the outputs of functions for volumetric evaluation. They allow to reference the output of nodes in a graph or define the mapping of the output channels for the sampling of an image3d with functionFromImage3D. The References are of the type ST_NodeOutputIdentifier.

2.1 ST_NodeOutputIdentifier

The ST_NodeOutputIdentifier is a simple type used to represent an identifier for a node output in the format of "nodename.outputname".

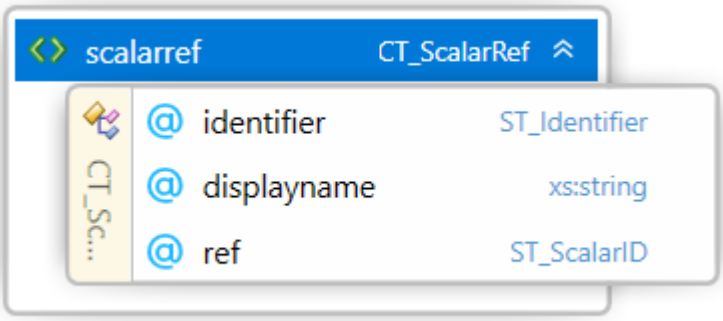
ST_ScalarID, ST_VectorID, ST_MatrixID and ST_ResourceOutputID are derived from ST_NodeOutputIdentifier.

Format

The format of ST_NodeOutputIdentifier is "nodename.outputname". The identifier must consist of alphanumeric characters and underscores. The dot (.) separates the node name and the output name.

2.2 ScalarReference

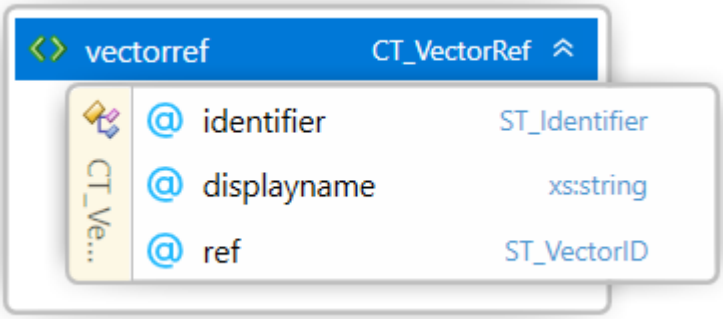
Element <scalarref>



Name	Type	Use	Default	Annotation
identifier	ST_Identifier	required		Specifies an identifier for this scalar resource.
displayname	xs:string	optional		The name to be displayed e.g. for annotation
ref	ST_ScalarID	required		Reference to the scalar in the form "NodeIdentifier.ScalarIdentifier".

2.3 VectorReference

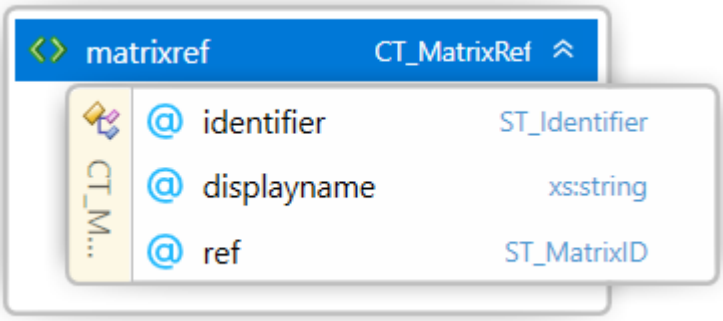
Element <vectorref>



Name	Type	Use	Default	Annotation
identifier	ST_Identifier	required		Specifies an identifier for this vector resource.
displayname	xs:string	optional		The name to be displayed e.g. for annotation.
ref	ST_VectorID	required		Reference to the scalar in the form "NodeIdentifier.VectorIdentifier".

2.4 MatrixReference

Element <matrixref>

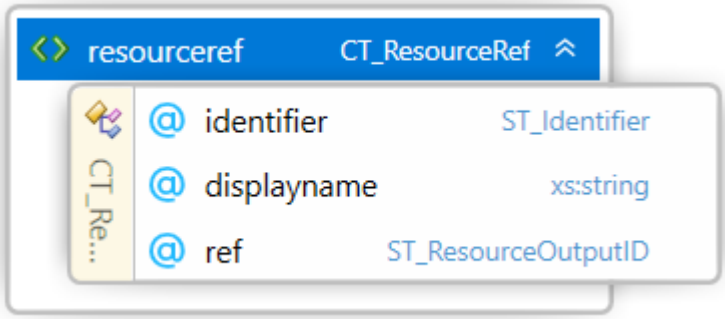


Name	Type	Use	Default	Annotation
identifier	ST_Identifier	required		Specifies an identifier for this matrix resource.
displayname	xs:string	optional		The name to be displayed e.g. for annotation
ref	ST_MatrixID	required		Reference to the scalar in the form "NodeIdentifier.VectorIdentifier".

2.5 ResourceReference

Element <resourceref>

References to resources are used for the volumeData element and to create a function reference.



Name	Type	Use	Default	Annotation
identifier	ST_Identifier	required		Specifies an identifier for this function resource.
displayname	xs:string	optional		The name to be displayed e.g. for annotation
ref	ST_ResourceOutputID	required		Reference to the resource output in the form "NodeIdentifier.OutputIdentifier".

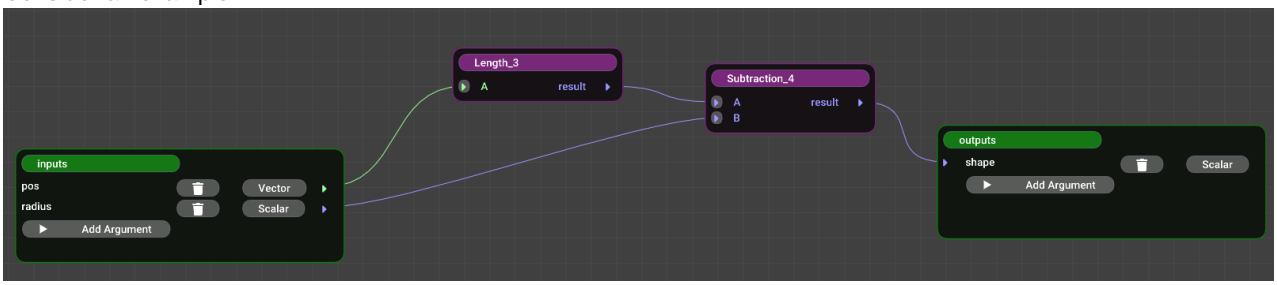
Chapter 3. Function Implicit

The *implicit* namespace enhances the *volumetric extension* by providing a way for the definition of closed form functions using the element <implicitfunction> in the <resources> section. It can be utilized for generating volumetric data as an

alternative to <functionfromimage3d> or referenced by <levelset> . These functions can be nested and can have an arbitrary number of inputs and outputs.

When used as input for <levelset> , the functions are evaluated at each point within the mesh or its bounding box. These functions are constructed through a graph-connected node set that is connected to both the function's inputs and outputs. Some of node types allow the usage of other resources, like computing the signed distance to mesh. Also a functionFromImage3D can be called from inside of a function.

Consider an example:




```
<i:implicitfunction id="5" displayname="sphere">
  <i:in>
    <i:vector identifier="pos" displayname="pos"></i:vector>
    <i:scalar identifier="radius" displayname="radius"></i:scalar>
  </i:in>
  <i:length identifier="Length_3" displayname="Length_3" tag="">
    <i:in>
      <i:vectorref identifier="A" displayname="A" ref="inputs.pos"></i:vectorref>
    </i:in>
    <i:out>
      <i:scalar identifier="result" displayname="result"></i:scalar>
    </i:out>
  </i:length>
  <i:subtraction identifier="Subtraction_4" displayname="Subtraction_4" tag="">
    <i:in>
      <i:scalarref identifier="A" displayname="A" ref="Length_3.result"></i:scalarref>
      <i:scalarref identifier="B" displayname="B" ref="inputs.radius"></i:scalarref>
    </i:in>
    <i:out>
      <i:scalar identifier="result" displayname="result"></i:scalar>
    </i:out>
  </i:subtraction>
  <i:out>
    <i:scalarref identifier="shape" displayname="shape" ref="Subtraction_4.result"></i:scalarref>
  </i:out>
</i:implicitfunction>
```

In this example, the *function* representing a sphere takes two inputs, a vector 'pos' and a scalar value 'radius'.

Links are defined by back-referencing the output of one node to the input of another node.

The <i:length> node computes the length of the input vector 'pos'. The connection to the input vector 'pos' is established through the <i:vectorref> element. References have the format nodename.outputname. In the case of function arguments the nodename is the reserved name 'inputs'.

The 'subtraction' node computes the difference between the length of the input vector 'pos' and the scalar value 'radius'. The connection to the length output is established through the <i:scalarref> element.

The <i:out> element defines the output of the function. The connection to the subtraction output is established through the <i:scalarref> element.

It showcases the flexibility of defining various mathematical operations like length computation and subtraction through nested nodes within the function. The result of these computations can be accessed through the <i:out> member.

Furthermore, a function can include basic mathematical operations like additions, subtractions, and multiplications to

cosines, logarithms or clamping. The operations can use different types like scalars, vectors, and matrices.

This flexible architecture allows the user to define almost any mathematical function with an arbitrary number of inputs and outputs, allowing extensive customization for generating volumetric data.

Chapter 4. Nodes

A node has a unique identifier and an arbitrary display name. A node must not have the identifier "inputs" or "outputs". Identifiers are restricted to alphanumeric characters.

Chapter 5. Native Nodes

Overview of native nodes

Node Type	Description
addition	addition of two values
subtraction	subtraction operation
multiplication	multiplication operation
division	division operation
constant	constant scalar value
constvec	constant vector
constmat	constant matrix
composevector	vector composition operation
vectorfromscalar	vector from scalar operation
decomposevector	vector decomposition operation
composematrix	matrix composition operation
matrixfromcolumns	matrix composition from column vectors
matrixfromrows	matrix composition from row vectors
dot	dot product operation
cross	cross product operation
matvecmultiplication	matrix-vector multiplication operation
transpose	matrix transpose operation
inverse	matrix inverse operation
sin	sine function operation
cos	cosine function operation
tan	tangent function operation
asin	arcsine (asin) function operation
acos	arccosine (acos) function operation
atan	arctangent (atan) function operation
atan2	two-argument arctangent function operation
min	minimum value operation
max	maximum value operation
abs	absolute value operation
fmod	Remainder of floating point division
mod	Modulo operation
pow	power function operation
sqrt	square root function operation

Node Type	Description
exp	exponential function operation
log	natural logarithm function operation
log2	base 2 logarithm function operation
log10	base 10 logarithm function operation
select	selection operation
clamp	clamping operation
cosh	hyperbolic cosine function operation
sinh	hyperbolic sine function operation
tanh	hyperbolic tangent function operation
round	rounding operation
ceil	ceiling operation
floor	floor operation
sign	signum operation
fract	fractional part extraction operation
functiongradient	spatial gradient of a function
normalizedistance	normalized distance from a function
functioncall	function call operation
beamlattice	signed distance to beam lattice operation
mesh	signed distance to mesh operation
unsignedmesh	unsigned distance to mesh operation
length	length operation
constresourceid	constant resource ID

addition

Description: Performs an addtion of the inputs "A" and "B" and writes the result to the output "result".

Inputs:

Identifier	Description
A	First summand
B	Second summand

Outputs:

Identifier	Description
result	Sum of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	vector is added componentwise
matrix	matrix	matrix	matrix elements are added componentwise

Example Usage:

```
<i:addition identifier="addition1" displayname="Addition 1">
  <i:in>
    <i:scalarref identifier="A" ref="constant1.c1"/>
    <i:scalarref identifier="B" ref="inputs.radius"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:addition>
```

constant

Description: Node for a constant scalar value. The output must have the identifier "value".

Example:

```
<i:constant identifier="constant1" displayname="Constant 1" value="1.0">
  <i:out>
    <i:scalar identifier="value"/>
  </i:out>
</i:constant>
```

Inputs:

Identifier	Description
None	

Outputs:

Identifier	Description
value	Constant scalar value

constvec

Description: Node for a constant vector value. The output must have the identifier "vector".

Inputs:

None

Outputs:

Identifier	Description
vector	Constant vector value

Attributes:

Attribute	Type	Required	Description
x	double	yes	X-component of the vector
y	double	yes	Y-component of the vector
z	double	yes	Z-component of the vector

Example Usage:

```
<i:constvec identifier="constant1" displayname="Constant 1" x="1.0" y="2.0" z="3.0">
  <i:out>
    <i:vector identifier="vector"/>
  </i:out>
</i:constvec>
```

constmat

Description: Node for a constant 4x4 matrix value. The output must have the identifier "matrix".

Inputs:

None

Outputs:

Identifier	Description
matrix	Constant 4x4 matrix

Attributes:

Attribute	Type	Required	Description
matrix	ST_Matrix4x4	yes	The constant 4x4 matrix representation

Example Usage:

```
<i:constmat identifier="constant1" displayname="identity"
   matrix="1.0 0.0 0.0 0.0
         0.0 1.0 0.0 0.0
         0.0 0.0 1.0 0.0
         0.0 0.0 0.0 1.0">
   <i:out>
      <i:matrix identifier="matrix"/>
   </i:out>
</i:constmat>
```

constresourceid

Description: Defines a model resource id as a constant value.

Inputs:

None

Outputs:

Identifier	Description
value	Resource ID

Attributes:

Attribute	Type	Required	Description
value	ST_ResourceID	yes	The model resource id

Example Usage:

```
<i:constresourceid identifier="resourceid1" displayname="Resource Id 1" value="1">
   <i:out>
      <i:resourceid identifier="value"/>
   </i:out>
</i:constresourceid>
```

composevector

Description: Node for composing a vector from 3 scalar values. The inputs must have the identifiers x, y and z.

Inputs:

Identifier	Description	Type
x	X-coordinate	scalar
y	Y-coordinate	scalar
z	Z-coordinate	scalar

Outputs:

Identifier	Description	Type
result	Composed vector (x,y,z)	vector

Example Usage:

```
<i:composevector>
  <i:in>
    <i:scalarref identifier="x" ref="inputs.x"/>
    <i:scalarref identifier="y" ref="inputs.y"/>
    <i:scalarref identifier="z" ref="inputs.z"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:composevector>
```

vectorfromscalar

Description: Node for creating a vector from a single scalar value. The input must have the identifier "A".

Inputs:

Identifier	Description	Type
A	A -> X, Y, Z	scalar

Outputs:

Identifier	Description	Type
result	vector (A, A, A)	vector

Example Usage:

```
<i:vectorfromscalar>
  <i:in>
    <i:scalarref identifier="A" ref="inputs.x"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:vectorfromscalar>
```

decomposevector

Description: Node for decomposing a vector into 3 scalar values. The input must have the identifier "A", and the outputs must have the identifiers x, y, and z.

Inputs:

Identifier	Description	Type
A	Vector to decompose	vector

Outputs:

Identifier	Description	Type
x	X-coordinate	scalar
y	Y-coordinate	scalar
z	Z-coordinate	scalar

Example Usage:

```
<i:decomposevector>
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector"/>
  </i:in>
  <i:out>
    <i:scalar identifier="x"/>
    <i:scalar identifier="y"/>
    <i:scalar identifier="z"/>
  </i:out>
</i:decomposevector>
```

composematrix

Description: Composes a matrix from 16 scalar values.

Inputs:

Identifier	Description
m00	Scalar value for element (0,0) of the matrix
mRC	Scalar value for element (R,C) of the matrix
...	...
m33	Scalar value for element (3,3) of the matrix

Outputs:

Identifier	Description
result	Composed matrix

Example Usage:

```
<i:composematrix identifier="composeMatrix_0" displayname="composed matrix">
 <i:in>
  <i:scalarref identifier="m00" ref="constantM00.value"/>
  <i:scalarref identifier="m01" ref="constantM01.value"/>
  <i:scalarref identifier="m02" ref="constantM02.value"/>
  <i:scalarref identifier="m03" ref="constantM03.value"/>
  <i:scalarref identifier="m10" ref="constantM10.value"/>
  <i:scalarref identifier="m11" ref="constantM11.value"/>
  <i:scalarref identifier="m12" ref="constantM12.value"/>
  <i:scalarref identifier="m13" ref="constantM13.value"/>
  <i:scalarref identifier="m20" ref="constantM20.value"/>
  <i:scalarref identifier="m21" ref="constantM21.value"/>
  <i:scalarref identifier="m22" ref="constantM22.value"/>
  <i:scalarref identifier="m23" ref="constantM23.value"/>
  <i:scalarref identifier="m30" ref="constantM30.value"/>
  <i:scalarref identifier="m31" ref="constantM31.value"/>
  <i:scalarref identifier="m32" ref="constantM32.value"/>
  <i:scalarref identifier="m33" ref="constantM33.value"/>
 </i:in>
 <i:out>
  <i:matrix identifier="result"/>
 </i:out>
</i:composematrix>
```

matrixfromcolumns

Description: Node for composing a matrix from column vectors. The 4th row is set [0,0,0,1].

Inputs:

Identifier	Description
A	First column vector
B	Second column vector
C	Third column vector
D	Fourth column vector

Outputs:

Identifier	Description
matrix	Composed matrix

The operation can be used for the following types of inputs and outputs:

A	B	C	D	result
vector	vector	vector	vector	matrix

Example Usage:

```
<i:matrixfromcolumns identifier="matrixfromcolumns" displayname="composed matrix">
   <i:in>
      <i:vectorref identifier="A" ref="vector0.vector"/>
      <i:vectorref identifier="B" ref="vector1.vector"/>
      <i:vectorref identifier="C" ref="vector2.vector"/>
      <i:vectorref identifier="D" ref="vector3.vector"/>
   </i:in>
   <i:out>
      <i:matrix identifier="result"/>
   </i:out>
</i:matrixfromcolumns>
```

matrixfromrows

Description: Node for composing a Matrix from row vectors. The 4th column is set to (0,0,0,1).

Inputs:

Identifier	Description
A	First row vector
B	Second row vector
C	Third row vector
D	Fourth row vector

Outputs:

Identifier	Description
result	Composed matrix

The operation can be used for the following types of inputs and outputs:

A	B	C	D	result	comment
vector	vector	vector	vector	matrix	Each row vector represents each row in the matrix

Example Usage:

```
<i:matrixfromrows identifier="matrixfromrows" displayname="composed vector">
  <i:in>
    <i:vectorref identifier="A" ref="vector0.vector"/>
    <i:vectorref identifier="B" ref="vector1.vector"/>
    <i:vectorref identifier="C" ref="vector2.vector"/>
    <i:vectorref identifier="D" ref="vector3.vector"/>
  </i:in>
  <i:out>
    <i:matrix identifier="result"/>
  </i:out>
</i:matrixfromrows>
```

multiplication

Description: Performs the multiplication A x B = result. The inputs must have the identifier "A" and "B", and the output must have the identifier "result". All inputs must be of the same type (scalar, vector, matrix).

Inputs:

Identifier	Description
A	First value to be multiplied
B	Second value to be multiplied

Outputs:

Identifier	Description
result	Result of the multiplication

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	vector components are multiplied component-wise
matrix	matrix	matrix	matrix elements are multiplied component-wise

Example Usage:

```
<i:multiplication identifier="multiplication1" displayname="Multiplication 1">
 <i:in>
  <i:scalarref identifier="A" ref="constant1.c1"/>
  <i:scalarref identifier="B" ref="inputs.radius"/>
 </i:in>
 <i:out>
  <i:scalar identifier="result"/>
 </i:out>
</i:multiplication>
```

subtraction

Description: Subtracts the inputs "A" and "B" and writes the difference to the output "result".

Inputs:

Identifier	Description
A	First value to subtract
B	Second value to subtract

Outputs:

Identifier	Description
result	Difference of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	Vector components are subtracted component-wise
matrix	matrix	matrix	Matrix elements are subtracted component-wise

Example Usage:

```
<i:subtraction identifier="subtraction1" displayname="Subtraction 1">
  <i:in>
    <i:scalarref identifier="A" ref="constant1.c1"/>
    <i:scalarref identifier="B" ref="inputs.radius"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:subtraction>
```

division

Description: Performs a division of the inputs "A" and "B" and writes the quotient to the output "result".

Inputs:

Identifier	Description
A	Dividend
B	Divisor

Outputs:

Identifier	Description
result	Quotient

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	Vector components are divided component-wise
matrix	matrix	matrix	Matrix elements are divided component-wise

Example Usage:

```
<i:division identifier="division1" displayname="Division 1">
  <i:in>
    <i:scalarref identifier="A" ref="constant1.c1"/>
    <i:scalarref identifier="B" ref="inputs.radius"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:division>
```

dot

Description: Performs a dot product of two vectors "A" and "B" and writes the result to the output "result".

Inputs:

Identifier	Description
A	First vector
B	Second vector

Outputs:

Identifier	Description
result	Dot product of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
vector	vector	scalar	dot product of the two vectors

Example Usage:

```
<i:dot identifier="dotproduct1" displayname="Dot Product 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
    <i:vectorref identifier="B" ref="inputs.vector2"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:dot>
```

cross

Description: Performs a cross product of the inputs "A" and "B" and writes the result to the output "result".

Inputs:

Identifier	Description
A	First vector
B	Second vector

Outputs:

Identifier	Description
result	Cross product of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
vector	vector	vector	cross product of the two vectors

Example Usage:

```
<i:cross identifier="crossproduct1" displayname="Cross Product 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
    <i:vectorref identifier="B" ref="inputs.vector2"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:cross>
```

matrixvectormultiplication

Description: Performs the matrix vector multiplication `A x B = result`. The output must be a vector with the name "result".

Inputs:

Identifier	Description
A	Matrix
B	Vector

Outputs:

Identifier	Description
result	Product of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
matrix	vector	vector	

Example Usage:

```
<matrixvectormultiplication identifier="matVec1" displayname="Matrix Vector Multiplication 1">
  <i:in>
    <i:matrixref identifier="A" ref="inputs.matrix1"/>
    <i:vectorref identifier="B" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</matrixvectormultiplication>
```

transpose

Description: Performs the transpose of a matrix. The input 'A' must be a matrix and the output must be a matrix with the identifier "result".

Inputs:

Identifier	Description
A	Matrix to be transposed

Outputs:

Identifier	Description
result	Transposed matrix

The operation can be used for the following types of inputs and outputs:

A	result
matrix	matrix

Example Usage:

```
<i:transpose identifier="transpose1" displayname="Transpose 1">
 <i:in>
  <i:matrixref identifier="A" ref="inputs.matrix1"/>
 </i:in>
 <i:out>
  <i:matrix identifier="result"/>
 </i:out>
</i:transpose>
```

inverse

Description: Computes the inverse of a matrix. The input "A" must be a matrix and the output must be a matrix with the identifier "result".

Inputs:

Identifier	Description
A	Matrix to compute inverse

Outputs:

Identifier	Description
result	Inverse of matrix A

The operation can be used for the following types of inputs and outputs:

A	result
matrix	matrix

Example Usage:

```
<i:inverse identifier="inverse1" displayname="Inverse 1">
   <i:in>
      <i:matrixref identifier="A" ref="inputs.matrix1"/>
   </i:in>
   <i:out>
      <i:matrix identifier="result"/>
   </i:out>
</i:inverse>
```

sin

Description: Computes sin(A) and writes the result to the output "result".

Inputs:

Identifier	Description
A	Input for sine function

Outputs:

Identifier	Description
result	Result of the sine function

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Sine of each component of the vector

Example usage:

```
<i:sin identifier="sinus1" displayname="Sinus 1">
  <i:in>
    <i:scalarref identifier="A" ref="inputs.scalar1"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:sin>
```

cos

Description: Computes cos(A) and writes the result to the output "result".

Inputs:

Identifier	Description
A	Input for cosine function

Outputs:

Identifier	Description
result	Cosine of A

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Cosine of each component of the vector

Example Usage:

```
<i:cos identifier="cosinus1" displayname="Cosinus 1">
  <i:in>
    <i:scalarref identifier="A" ref="inputs.scalar1"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:cos>
```

tan

Description: Computes tan(A) and writes the result to the output "result".

Inputs:

Identifier	Description
A	Input value

Outputs:

Identifier	Description
result	Tangent of A

The operation can be used for the following types of inputs and outputs:

A	result
scalar	scalar
vector	Tangent of each component of the vector

Example Usage:

```
<i:tan identifier="tan1" displayname="Tan 1">
  <i:in>
    <i:scalarref identifier="A" ref="inputs.value"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:tan>
```

asin

Description: Performs an arcsine (asin) function with a scalar or vector input and produces a scalar or vector output. The input must have the identifier "A", and the output must have the identifier "result".

Inputs:

Identifier	Description
A	Input for the arcsine function

Outputs:

Identifier	Description
result	Arcsine of the input

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Arcsine of each component of the vector

Example Usage:

```
<i:asin identifier="asin1" displayname="Asin 1">
  <i:in>
    <i:scalarref identifier="A" ref="inputs.scalar1"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:asin>
```

acos

Description: Performs an arccosine (acos) function with a scalar or vector input and produces a scalar or vector output. The input must have the identifier "A", and the output must have the identifier "result".

Inputs:

Identifier	Description
A	Input value

Outputs:

Identifier	Description
result	Arccosine of A

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Arccosine of each component of the vector

Example Usage:

```
<i:acos identifier="acos1" displayname="Acos 1">
 <i:in>
  <i:vectorref identifier="A" ref="inputs.vector1"/>
 </i:in>
 <i:out>
  <i:vector identifier="result"/>
 </i:out>
</i:acos>
```

atan

Description: Performs an arctangent (atan) function with a scalar or vector input and produces a scalar or vector output. The input must have the identifier "A", and the output must have the identifier "result".

Inputs:

Identifier	Description
scalarref	Scalar input
vectorref	Vector input

Outputs:

Identifier	Description
scalar	Scalar output
vector	Vector output (arctangent of each component)

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Arctangent of each component of the vector

Example Usage:

```
<i:atan identifier="atan1" displayname="Atan 1">
 <i:in>
  <i:vectorref identifier="A" ref="inputs.vector1"/>
 </i:in>
 <i:out>
  <i:vector identifier="result"/>
 </i:out>
</i:atan>
```

atan2

Description: Performs the two-argument arctangent (atan2) of the inputs "A" and "B" and writes the result to the output "result". The inputs can be scalar or vector and the output is scalar or vector.

Inputs:

Identifier	Description
A	First input
B	Second input

Outputs:

Identifier	Description
result	Result of the atan2 operation

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	Atan2 of each component of the vectors

Example Usage:

```
<i:atan2 identifier="atan21" displayname="Atan2 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
      <i:vectorref identifier="B" ref="inputs.vector2"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:atan2>
```

min

Description: Performs a minimum operation on the inputs "A" and "B" and writes the result to the output "result".

Inputs:

Identifier	Description
A	First operand
B	Second operand

Outputs:

Identifier	Description
result	Minimum value of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	Minimum value of each component of vectors
matrix	matrix	matrix	Minimum value of each component of matrices

Example Usage:

```
<i:min identifier="min1" displayname="Min 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
      <i:vectorref identifier="B" ref="inputs.vector2"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:min>
```

max

Description: Performs a max function on the inputs "A" and "B" and writes the result to the output "result".

Inputs:

Identifier	Description
A	First value
B	Second value

Outputs:

Identifier	Description
result	Maximum value of A and B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	Maximum of each component of the vectors
matrix	matrix	matrix	Maximum of each component of the matrices

Example Usage:

```
<i:max identifier="max1" displayname="Max 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
      <i:vectorref identifier="B" ref="inputs.vector2"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:max>
```

abs

Description: Performs the absolute value operation on input "A" and writes the result to output "result".

Inputs:

Identifier	Description
A	Input for absolute value function

Outputs:

Identifier	Description
result	Result of absolute value operation

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Absolute value is computed for each component of the vector
matrix	matrix	Absolute value is computed for each component of the matrix

Example Usage:

```
<i:abs identifier="abs1" displayname="Abs 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:abs>
```

fmod

Description: Computes the remainder of the divison of the inputs "A" / "B" and writes the result to the output "result", as known from C++ fmod. result = A - B * trunc(A/B) The result will have the same sign as A.

Inputs:

Identifier	Description
A	The dividend
B	The divisor

Outputs:

Identifier	Description
result	The remainder of A divided by B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	modulo operation of each component of the vectors
matrix	matrix	matrix	modulo operation of each component of the matrices

Example Usage:

```
<i:fmod identifier="fmod1" displayname="Fmod 1">
  <i:in>
    <i:scalarref identifier="A" ref="constant1.c1"/>
    <i:scalarref identifier="B" ref="inputs.radius"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:fmod>
```

mod

Description: Performs a modulo operation on the inputs "A" and "B" and writes the result to the output "result" as known from GLSL mod. result = A - B * floor(A/B)

Inputs:

Identifier	Description
A	The dividend
B	The divisor

Outputs:

Identifier	Description
result	The remainder of A divided by B

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	modulo operation of each component of the vectors
matrix	matrix	matrix	modulo operation of each component of the matrices

Example Usage:

```
<i:mod identifier="mod1" displayname="mod 1">
  <i:in>
    <i:scalarref identifier="A" ref="constant1.c1"/>
    <i:scalarref identifier="B" ref="inputs.radius"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:mod>
```

pow

Description: Performs a power function with the input "A" as the base and "B" as the exponent. The result is written to the output "result".

Inputs:

Identifier	Description
A	Base of the power
B	Exponent of the power

Outputs:

Identifier	Description
result	Result of the power

The operation can be used for the following types of inputs and outputs:

A	B	result	comment
scalar	scalar	scalar	
vector	vector	vector	Power of each component of the vectors
matrix	matrix	matrix	Power of each component of the matrices

Example Usage:

```
<i:pow identifier="pow1" displayname="Pow 1">
 <i:in>
  <i:vectorref identifier="A" ref="inputs.vector1"/>
  <i:vectorref identifier="B" ref="inputs.vector2"/>
 </i:in>
 <i:out>
  <i:vector identifier="result"/>
 </i:out>
</i:pow>
```

sqrt

Description: Performs the square root operation on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	The value to take the square root of

Outputs:

Identifier	Description
result	The square root of A

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	square root of each component of the vector
matrix	matrix	square root of each component of the matrix

Example Usage:

```
<i:sqrt identifier="sqrt1" displayname="Sqrt 1">
 <i:in>
  <i:vectorref identifier="A" ref="inputs.vector1"/>
 </i:in>
 <i:out>
  <i:vector identifier="result"/>
 </i:out>
</i:sqrt>
```

mesh

Description: Evaluates the signed distance to a mesh. The input must have the identifier "pos" and must be a vector. The output is a scalar with the identifier "distance". The mesh is assumed to be closed and watertight.

Inputs:

Identifier	Description
pos	Input vector
mesh	Resource identifier for the mesh

Outputs:

Identifier	Description
distance	Distance to the mesh

The operation can be used for the following types of inputs and outputs:

pos	mesh	distance	comment
vector	-	scalar	-

Example Usage:

```
<i:mesh identifier="distanceToMesh1" displayname="Distance to Mesh 1" objectid="1" >
 <i:in>
  <i:vectorref identifier="pos" ref="inputs.pos"/>
    <i:resourceref identifier="mesh" ref="resourceidnode.value"/>
 </i:in>
 <i:out>
  <i:scalar identifier="distance"/>
 </i:out>
</i:mesh>
```

unsignedmesh

Description: Evaluates the unsigned distance to a mesh. The input must have the identifier "pos" and must be a vector. The output is a scalar with the identifier "distance". The mesh does not need to be closed or watertight.

Inputs:

Identifier	Description
pos	Input vector
mesh	Resource identifier for the mesh

Outputs:

Identifier	Description
distance	Distance to the mesh

The operation can be used for the following types of inputs and outputs:

pos	mesh	distance	comment
vector	-	scalar	-

Example Usage:

```
<i:unsignedmesh identifier="UnsignedDistToMesh1" displayname="Unsigned distance to Mesh" objectid="1" >
 <i:in>
  <i:vectorref identifier="pos" ref="inputs.pos"/>
    <i:resourceref identifier="mesh" ref="resourceidnode.value"/>
 </i:in>
 <i:out>
  <i:scalar identifier="distance"/>
 </i:out>
</i:unsignedmesh>
```

length

Description: Calculates the length of the input vector and writes the result to the output "result".

Inputs:

Identifier	Description
A	Input Vector

Outputs:

Identifier	Description
result	Length of the input vector

The operation can be used for the following types of inputs and outputs:

A	result	comment
vector	scalar	length(vector) = scalar

Example Usage:

```
<i:length identifier="length1" displayname="Length 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:length>
```

beamlattice

Description: Evaluates the signed distance to a beam lattice. The input MUST have the identifier "pos" and MUST be a vector. A resource identifier MUST specify the beam lattice. The output is a scalar with the identifier "distance". The distance is positive if the point is outside the beam lattice and negative if the point is inside the beam lattice.

Inputs:

Identifier	Description
pos	Input vector
beamlattice	Resource identifier for the beam lattice

Outputs:

Identifier	Description
distance	Signed distance to the beam lattice

The operation can be used for the following types of inputs and outputs:

pos	beamlattice	distance	comment
vector	-	scalar	-

Attributes:

Attribute	Type	Required	Default	Description
accuraterange	double	no	0.0	Non-negative distance band (in model units) around the beam lattice within which the returned signed distance MUST be accurate. Outside this band (i.e., for points p with

Example Usage:

```
<i:beamlattice identifier="SignedDistanceToBeamLattice1" displayname="Signed Distance to Beam Lattice 1" accuraterange="2.5">
  <i:in>
    <i:vectorref identifier="pos" ref="inputs.pos"/>
    <i:resourceref identifier="beamlattice" ref="resourceidnode.value"/>
  </i:in>
  <i:out>
    <i:scalar identifier="distance"/>
  </i:out>
</i:beamlattice>
```

functiongradient

Description: Computes the spatial gradient of a referenced function output using central finite differences. The inputs MUST include a resource reference with the identifier "functionID", a scalar input named "step", and all argument references required by the referenced function. The attribute "scalaroutput" MUST name a scalar output of the referenced function. The attribute "vectorinput" MUST name a vector (float3) input of the referenced function with respect to which the gradient is computed. Consumers MUST clamp the effective step to >= 1e-8.

The outputs MUST have the identifiers "vector", "gradient" and "magnitude" where:

- "vector" is the normalized gradient (float3)
- "gradient" is the raw gradient (float3)
- "magnitude" is the length of the gradient (scalar)

Central finite differences are used: for each component c ∈ {x,y,z}, evaluate the referenced scalar output at the selected vector input offset by ±step along c and compute (f(x+h_c) - f(x-h_c)) / (2·step). If the provided step is missing or not finite (NaN/Inf), the result is undefined.

Inputs:

Identifier	Description
functionID	Resource reference to the function to differentiate
step	Scalar step size for central finite differences
defined by function	Provide references for all inputs of the referenced function; the identifier of the vector input to differentiate MUST match the value of the @vectorinput attribute

Outputs:

Identifier	Description
vector	Normalized gradient (float3)
gradient	Raw gradient (float3)
magnitude	Length of the gradient (scalar)

Attributes:

Attribute	Type	Required	Default	Description
scalaroutput	xs:string	yes		Name of the scalar output of the referenced function to differentiate
vectorinput	xs:string	yes		Name of the vector (float3) input of the referenced function used for differentiation

Example Usage:

```
<i:functiongradient identifier="grad1" displayname="Gradient of f" scalaroutput="distance" vectorinput="pos">
  <i:in>
    <i:resourceref identifier="functionID" ref="FunctionCall_5_functionID.value"/>
    <i:vectorref identifier="pos" ref="inputs.pos"/>
    <i:scalarref identifier="radius" ref="inputs.radius"/>
    <i:scalarref identifier="step" ref="inputs.step"/>
  </i:in>
  <i:out>
    <i:vector identifier="vector"/>
    <i:vector identifier="gradient"/>
    <i:scalar identifier="magnitude"/>
  </i:out>
</i:functiongradient>
```

normalizedistance

Description: Computes a normalized distance from a referenced scalar function output by dividing the function value by the magnitude of its spatial gradient, using central finite differences. The inputs MUST include a resource reference with the identifier "functionID", a scalar input named "step", and all argument references required by the referenced function. The attribute "scalaroutput" MUST name a scalar output of the referenced function. The attribute "vectorinput" MUST name a vector (float3) input of the referenced function with respect to which the gradient is computed. Consumers MUST clamp the effective step to >= 1e-8.

Formally, let f be the referenced scalar output and x the vector input. Using central finite differences with the provided step, compute ∇f(x) and its magnitude |∇f(x)|, then output f(x)/max(|∇f(x)|, 1e-8). If the provided step is missing or not finite (NaN/Inf), the result is undefined.

Inputs:

Identifier	Description
functionID	Resource reference to the function providing f(x)
step	Scalar step size for central finite differences
defined by function	Provide references for all inputs of the referenced function; the identifier of the vector input to differentiate MUST match the value of the @vectorinput attribute

Outputs:

Identifier	Description
result	Normalized distance f(x)/

Attributes:

Attribute	Type	Required	Default	Description
scalaroutput	xs:string	yes		Name of the scalar output of the referenced function to use as f(x)
vectorinput	xs:string	yes		Name of the vector (float3) input of the referenced function used for gradient computation

Example Usage:

```
<i:normalizedistance identifier="n1" displayname="Normalized Distance" scalaroutput="distance" vectorinput="pos">
  <i:in>
    <i:resourceref identifier="functionID" ref="FunctionCall_5_functionID.value"/>
    <i:vectorref identifier="pos" ref="inputs.pos"/>
    <i:scalarref identifier="step" ref="inputs.step"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:normalizedistance>
```

log

Description: Performs a natural logarithm of the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Value to compute natural logarithm for

Outputs:

Identifier	Description
result	Natural logarithm of input "A"

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Natural logarithm of each component
matrix	matrix	Natural logarithm of each component

Example Usage:

```
<i:log identifier="log_1" displayname="Log_n 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:log>
```

log2

Description: Performs a base 2 logarithm operation on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Value to calculate logarithm of

Outputs:

Identifier	Description
result	Result of the logarithm calculation

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Logarithm is calculated componentwise
matrix	matrix	Logarithm is calculated componentwise

Example Usage:

```
<i:log2 identifier="log21" displayname="Log2 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:log2>
```

log10

Description: Performs a base 10 logarithm of the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Number to calculate the logarithm of

Outputs:

Identifier	Description
result	Base 10 logarithm of input number A

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Logarithm is calculated componentwise
matrix	matrix	Logarithm is calculated componentwise

Example Usage:

```
<i:log10 identifier="log101" displayname="Log10 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:log10>
```

exp

Description: Performs an exponential function on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Input for exponential function

Outputs:

Identifier	Description
result	Result of exponential function

The operation can be used for the following types of inputs and outputs:

A	result	Comment
scalar	scalar	
vector	vector	Exponential of each component of the vector
matrix	matrix	Exponential of each component of the matrix

Example Usage:

```
<i:exp identifier="exp1" displayname="Exp 1">
 <i:in>
  <i:vectorref identifier="A" ref="inputs.vector1"/>
 </i:in>
 <i:out>
  <i:vector identifier="result"/>
 </i:out>
</i:exp>
```

cosh

Description: Performs a hyperbolic cosine function on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Input value for the hyperbolic cosine function

Outputs:

Identifier	Description
result	Result of the hyperbolic cosine function

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	cosh of each component of the vector
matrix	matrix	cosh of each component of the matrix

Example Usage:

```
<i:cosh identifier="cosh1" displayname="Cosh 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:cosh>
```

sinh

Description: Calculates the hyperbolic sine of the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Operand

Outputs:

Identifier	Description
result	Result

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	sinh of each component of the vector
matrix	matrix	sinh of each component of the matrix

Example Usage:

```
<i:sinh identifier="sinh1" displayname="Sinh 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:sinh>
```

tanh

Description: Calculates the hyperbolic tangent of the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Operand

Outputs:

Identifier	Description
result	Result

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	tanh of each component of the vector
matrix	matrix	tanh of each component of the matrix

Example Usage:

```
<i:tanh identifier="tanh1" displayname="Tanh 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:tanh>
```

clamp

Description: Performs the clamp function $max(min, min(A,max))$. The input must have the identifier "A" (value), "min", and "max". The output must have the identifier "result".

Inputs:

Identifier	Description
A	Value
min	Minimum value
max	Maximum value

Outputs:

Identifier	Description
result	Result of the clamp function

The operation can be used for the following types of inputs and outputs:

A	min	max	result
scalar	scalar	scalar	scalar
vector	vector	vector	Vector with clamped values component-wise
matrix	matrix	matrix	Matrix with clamped values component-wise

Example Usage:

```
<i:clamp identifier="clamp1" displayname="Clamp 1">
  <i:in>
    <i:scalarref identifier="A" ref="inputs.scalar1"/>
    <i:scalarref identifier="min" ref="inputs.scalar2"/>
    <i:scalarref identifier="max" ref="inputs.scalar3"/>
  </i:in>
  <i:out>
    <i:scalar identifier="result"/>
  </i:out>
</i:clamp>
```

select

Description: Derived node for a select function. The input must have the identifier "A", "B", "C", "D", and the output must have the identifier "result". If A < B then the result is C, otherwise D.

Inputs:

Identifier	Description
A	First value to compare
B	Second value to compare
C	Result if A < B
D	Result if A >= B

Outputs:

Identifier	Description
result	Result of A < B ? C : D

The operation can be used for the following types of inputs and outputs:

A	B	C	D	result
scalar	scalar	scalar	scalar	scalar
vector	vector	vector	vector	vector
matrix	matrix	matrix	matrix	matrix

Example Usage:

```
<i:select identifier="select1" displayname="Select 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
    <i:vectorref identifier="B" ref="inputs.vector2"/>
    <i:vectorref identifier="C" ref="inputs.vector3"/>
    <i:vectorref identifier="D" ref="inputs.vector4"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:select>
```

round

Description: Performs a rounding operation on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Value to be rounded

Outputs:

Identifier	Description
result	Rounded value of A

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Round of each component of the vector
matrix	matrix	Round of each component of the matrix

Example Usage:

```
<i:round identifier="round1" displayname="Round 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:round>
```

ceil

Description: Performs the ceil function on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Value to be rounded up

Outputs:

Identifier	Description
result	Rounded up value

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	Round up a single scalar value
vector	vector	Round up each component of the vector
matrix	matrix	Round up each component of the matrix

Example Usage:

```
<i:ceil identifier="ceil1" displayname="Ceil 1">
   <i:in>
      <i:vectorref identifier="A" ref="inputs.vector1"/>
   </i:in>
   <i:out>
      <i:vector identifier="result"/>
   </i:out>
</i:ceil>
```

floor

Description: Performs a floor function on the input "A" and writes the result to the output "result".

Inputs:

Identifier	Description
A	Value to be rounded down

Outputs:

Identifier	Description
result	Rounded down value of A

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Floor of each component of the vector
matrix	matrix	Floor of each component of the matrix

Example Usage:

```
<i:floor identifier="floor1" displayname="Floor 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:floor>
```

sign

Description: Performs the sign function on the input "A" and writes the result to the output "result". If the input is less than zero this function returns -1, if the input is greater than zero this function returns 1, and if the input is equal to zero this function returns 0.

Inputs:

Identifier	Description
A	Input value

Outputs:

Identifier	Description
result	Sign of the input

The operation can be used for the following types of inputs and outputs:

A	result	comment
scalar	scalar	
vector	vector	Sign of each component of the vector
matrix	matrix	Sign of each component of the matrix

Example Usage:

```
<i:sign identifier="sign1" displayname="Sign 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:sign>
```

frac

fract

Description: Returns the fractional part of the input (A - floor(A)).

Inputs:

Identifier	Description
A	Input value

Outputs:

Identifier	Description
result	Fractional part of A

The operation can be used for the following types of inputs and outputs:

A	result	Comment
scalar	scalar	
vector	vector	Sign of each component of the vector
matrix	matrix	Sign of each component of the matrix

Example Usage:

```
<i:sign identifier="sign1" displayname="Sign 1">
  <i:in>
    <i:vectorref identifier="A" ref="inputs.vector1"/>
  </i:in>
  <i:out>
    <i:vector identifier="result"/>
  </i:out>
</i:sign>
```

functionCall

Description: Calls a function with the given identifier. The input must have the identifier "functionID". The rest of the inputs and outputs must match the inputs and outputs of the function.

Inputs:

Identifier	Description
functionID	Identifier of the function to call
defined by function	

Outputs:

Identifier	Description
defined by function	

Example Usage:

```
<i:functioncall identifier="functionCall1" displayname="Function Call 1">
 <i:in>
  <i:resourceref identifier="functionID" ref="resourceidnode.value"/>
 </i:in>
 <i:out>
  <i:scalar identifier="result"/>
 </i:out>
</i:functioncall>
```

Calling the sphere function from the example in Chapter 2. Function Implicit could look like this:

```
<i:constant identifier="ConstantScalar_3" displayname="Radius" tag="" value="10">
  <i:out>
    <i:scalar identifier="value" displayname="value"></i:scalar>
  </i:out>
</i:constant>
<i:constresourceid identifier="FunctionCall_5_functionID" displayname="sphere_functionID" tag="" resourceid="5">
  <i:out>
    <i:resourceid identifier="value" displayname="value"></i:resourceid>
  </i:out>
</i:constresourceid>
<i:functioncall identifier="FunctionCall_6" displayname="sphere" tag="">
  <i:in>
    <i:resourceref identifier="functionID" displayname="functionID" ref="FunctionCall_5_functionID.value"></i:resourceref>
    <i:vectorref identifier="pos" displayname="pos" ref="Subtraction_5.result"></i:vectorref>
    <i:scalarref identifier="radius" displayname="radius" ref="ConstantScalar_3.value"></i:scalarref>
  </i:in>
  <i:out>
    <i:scalar identifier="shape" displayname="shape"></i:scalar>
  </i:out>
</i:functioncall>
```

The inputs of the sphere function are added as inputs to the <i:functioncall> node. The output of the <i:functioncall> node is the output of the sphere function. A <i:constresourceid> node is used to define the functionID of the sphere function. Note that a resourceid can also be used as a function input.

Chapter 6. Implicit Evaluation

6.1 Valid Graphs

The native nodes provided are used to create arbitrary graphs. In order for these graphs to be evaluatable they must meet the following criteria and any graph that fails to meet this criteria MUST be rejected.

- Nodes that compose a Function form a subgraph that must be acyclic and directed.
- References between Functions that form the graph MUST be acyclic and directed.
- A Function MUST NOT reference itself.
- Inputs must only reference outputs of the same DataType.

6.2 Undefined Results and Fallback Values

The native nodes provided can create graphs that have regions that will evaluate to an undefined value. This undefined value presents a problem when trying to evaluate a <levelset> object or a volume data element such as <color> . Such undefined results make the result of the function undefined and the volumetric data element MUST be evaluated to the volumetric data element's fallback value.

Chapter 7. Notes

Part III. Appendices

Appendix A. Glossary

See the standard 3MF Glossary.

Appendix B. 3MF XSD Schema for the Volumetric and Implicit Extensions

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://schemas.3mf.io/3dmanufacturing/volumetric/2022/01" xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xml="http://www.w3.org/XML/1998/namespace" targetNamespace="http://schemas.3mf.io/3dmanufacturing/volumetric/2022/01"
 elementFormDefault="unqualified" attributeFormDefault="unqualified" blockDefault="#all">

 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Schema notes:

  Items within this schema follow a simple naming convention of appending a prefix indicating the type of element for references:

  Unprefixed: Element names
  CT_: Complex types
  ST_: Simple types

  ]]>
  </xs:documentation>
 </xs:annotation>

 <!-- Complex Types -->
 <xs:complexType name="CT_Resources">
  <xs:sequence>
   <xs:choice minOccurs="0" maxOccurs="2147483647">
    <xs:element ref="volumedata" minOccurs="0" maxOccurs="2147483647" />
    <xs:element ref="image3d" minOccurs="0" maxOccurs="2147483647" />
    <xs:element ref="functionfromimage3d" minOccurs="0" maxOccurs="2147483647" />
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>
   </xs:choice>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>

 <xs:complexType name="CT_Object">
  <xs:sequence>
   <xs:choice>
    <xs:element ref="levelset" />
   </xs:choice>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="CT_Image3D">
  <xs:sequence>
   <xs:choice>
    <xs:element ref="imagestack" />
    <xs:any namespace="##other" processContents="lax" />
   </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="ST_ResourceID" use="required" />
  <xs:attribute name="name" type="xs:string" />
 </xs:complexType>

 <xs:complexType name="CT_ImageStack">
  <xs:sequence>
   <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647" />
   <xs:element ref="imagesheet" minOccurs="1" maxOccurs="1073741824" />
  </xs:sequence>
  <xs:attribute name="rowcount" type="xs:positiveInteger" use="required" />
  <xs:attribute name="columncount" type="xs:positiveInteger" use="required" />
  <xs:attribute name="sheetcount" type="xs:positiveInteger" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>

 <xs:complexType name="CT_ImageSheet">
  <xs:sequence>
   <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647" />
```

```
 </xs:sequence>
 <xs:attribute name="path" type="ST_UriReference" use="required" />
 <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:complexType name="CT_Function">
 <xs:attribute name="id" type="ST_ResourceID" use="required" />
 <xs:attribute name="displayname" type="xs:string" />
 <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:complexType name="CT_FunctionFromImage3D">
 <xs:complexContent>
  <xs:extension base="CT_Function">
   <xs:attribute name="image3did" type="ST_ResourceID" use="required" />
   <xs:attribute name="filter" type="ST_Filter" default="linear" />
   <xs:attribute name="valueoffset" type="ST_Number" default="0.0" />
   <xs:attribute name="valuescale" type="ST_Number" default="1.0" />
   <xs:attribute name="tilestyleu" type="ST_TileStyle" default="wrap" />
   <xs:attribute name="tilestylev" type="ST_TileStyle" default="wrap" />
   <xs:attribute name="tilestylew" type="ST_TileStyle" default="wrap" />
   <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="CT_VolumeData">
 <xs:sequence>
  <xs:element ref="composite" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="color" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="property" minOccurs="0" maxOccurs="2147483647"/>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>
 </xs:sequence>
 <xs:attribute name="id" type="ST_ResourceID" use="required" />
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="CT_Mesh">
 <xs:attribute name="volumeid" type="ST_ResourceID" />
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="CT_LevelSet">
 <xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>
 </xs:sequence>
 <xs:attribute name="functionid" type="ST_ResourceID" use="required" />
 <xs:attribute name="channel" type="xs:QName" use="required" />
 <xs:attribute name="transform" type="ST_Matrix3D" />
 <xs:attribute name="minfeaturesize" type="ST_PositiveNumber" default="0" />
 <xs:attribute name="meshbboxonly" type="xs:boolean" default="false" />
 <xs:attribute name="fallbackvalue" type="ST_Number" default="0" />
 <xs:attribute name="meshid" type="ST_ResourceID" use="required"/>
 <xs:attribute name="volumeid" type="ST_ResourceID" />
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="CT_Color">
 <xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647" />
 </xs:sequence>
 <xs:attribute name="functionid" type="ST_ResourceID" use="required" />
 <xs:attribute name="channel" type="xs:QName" use="required" />
 <xs:attribute name="transform" type="ST_Matrix3D" />
```

```
 <xs:attribute name="minfeaturesize" type="ST_PositiveNumber" default="0" />
 <xs:attribute name="fallbackvalue" type="ST_Number" default="0" />
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="CT_Composite">
 <xs:sequence>
  <xs:element ref="materialmapping" minOccurs="1" maxOccurs="2147483647"/>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>
 </xs:sequence>
 <xs:attribute name="basematerialid" type="ST_ResourceID" use="required"/>
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="CT_MaterialMapping">
 <xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647" />
 </xs:sequence>
 <xs:attribute name="functionid" type="ST_ResourceID" use="required" />
 <xs:attribute name="channel" type="xs:QName" use="required" />
 <xs:attribute name="transform" type="ST_Matrix3D" />
 <xs:attribute name="minfeaturesize" type="ST_PositiveNumber" default="0" />
 <xs:attribute name="fallbackvalue" type="ST_Number" default="0" />
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="CT_Property">
 <xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647" />
 </xs:sequence>
 <xs:attribute name="functionid" type="ST_ResourceID" use="required" />
 <xs:attribute name="channel" type="xs:QName" use="required" />
 <xs:attribute name="transform" type="ST_Matrix3D" />
 <xs:attribute name="name" type="xs:QName" use="required" />
 <xs:attribute name="required" type="xs:boolean" use="required" />
 <xs:attribute name="minfeaturesize" type="ST_PositiveNumber" default="0" />
 <xs:attribute name="fallbackvalue" type="ST_Number" default="0" />
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Simple Types -->
<xs:simpleType name="ST_TileStyle">
 <xs:restriction base="xs:string">
  <xs:enumeration value="wrap"/>
  <xs:enumeration value="mirror"/>
  <xs:enumeration value="clamp"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ST_Filter">
 <xs:restriction base="xs:string">
  <xs:enumeration value="linear"/>
  <xs:enumeration value="nearest"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ST_CompositionSpace">
 <xs:restriction base="xs:string">
  <xs:enumeration value="raw"/>
  <xs:enumeration value="linearcolor"/>
 </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ST_ResourceID">
 <xs:restriction base="xs:positiveInteger">
  <xs:maxExclusive value="2147483648"/>
 </xs:restriction>
```

```
 </xs:simpleType>
 <xs:simpleType name="ST_UriReference">
  <xs:restriction base="xs:anyURI">
   <xs:pattern value="/.*"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="ST_Number">
  <xs:restriction base="xs:double">
   <xs:whiteSpace value="collapse"/>
   <xs:pattern value="((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?)"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="ST_PositiveNumber">
  <xs:restriction base="xs:double">
   <xs:whiteSpace value="collapse"/>
   <xs:pattern value="((\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?)"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="ST_Matrix3D">
  <xs:restriction base="xs:string">
   <xs:whiteSpace value="collapse"/>
   <xs:pattern value="((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?) ((\-|\+)?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))((e|E)(\-|\+)?[0-9]+)?)"/>
  </xs:restriction>
 </xs:simpleType>

 <!-- Elements -->
 <xs:element name="image3d" type="CT_Image3D"/>
 <xs:element name="imagestack" type="CT_ImageStack"/>
 <xs:element name="imagesheet" type="CT_ImageSheet"/>
 <xs:element name="functionfromimage3d" type="CT_FunctionFromImage3D"/>
 <xs:element name="volumedata" type="CT_VolumeData"/>
 <xs:element name="composite" type="CT_Composite"/>
 <xs:element name="materialmapping" type="CT_MaterialMapping"/>
 <xs:element name="color" type="CT_Color"/>
 <xs:element name="property" type="CT_Property"/>
 <xs:element name="mesh" type="CT_Mesh"/>
 <xs:element name="levelset" type="CT_LevelSet"/>
</xs:schema>
```

sheet0.png

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.3mf.io/3dmanufacturing/implicit/2023/12"
 elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all"
 xmlns="http://schemas.3mf.io/3dmanufacturing/implicit/2023/12"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:vol="http://schemas.3mf.io/3dmanufacturing/volumetric/2022/01">
 <xs:import namespace="http://schemas.3mf.io/3dmanufacturing/volumetric/2022/01"
  schemaLocation="volumetric.xsd" />

 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Schema notes:

  Items within this schema follow a simple naming convention of appending a prefix indicating the type of element for references:

  Unprefixed: Element names
  CT_: Complex types
```

```
 ST_: Simple types

 ]]>
 </xs:documentation>
</xs:annotation>

<!-- Complex Types -->
<xs:complexType name="CT_Resources">
 <xs:choice minOccurs="0" maxOccurs="2147483647">
  <xs:element ref="implicitfunction" />
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>
 </xs:choice>
 <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<!-- NormalizeDistance: normalized signed distance using referenced function and its gradient -->
<xs:complexType name="CT_NormalizeDistance">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
 Derived node for computing a normalized distance from a referenced scalar function output using its gradient magnitude for normalization.
 The inputs must include a resource reference with the identifier "functionID", a scalar input "step" for finite differences, and all
 argument references required by the referenced function. The attribute "scalaroutput" must name a scalar output of the referenced function.
 The attribute "vectorinput" must name a vector (float3) input of the referenced function with respect to which the gradient is computed.

 Let f be the referenced scalar output and x the vector input. Using central finite differences with the provided step, compute ∇f(x) and its
 magnitude |∇f(x)|. The node outputs the normalized distance defined as f(x) / max(|∇f(x)|, eps) where eps = 1e-8. Consumers MUST clamp
the effective
 step to >= 1e-8 and treat missing or non-finite step values as undefined results.

 Example:
 <normalizedistance identifier="n1" displayname="Normalized Distance" scalaroutput="distance" vectorinput="pos">
  <in>
   <resourceref identifier="functionID" ref="functionIDNode.value"/>
   <vectorref identifier="pos" ref="inputs.pos"/>
   <scalarref identifier="step" ref="inputs.step"/>
  </in>
  <out>
   <scalar identifier="result"/>
  </out>
 </normalizedistance>
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
 Inputs to the normalized distance. Must include a resourceref with identifier "functionID", a scalarref with identifier "step", and the
 argument references required by the referenced function. The identifier of the vector input must match the value of the @vectorinput attribute.
        ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice minOccurs="1" maxOccurs="2147483647">
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
       <xs:element ref="matrixref" />
       <xs:element ref="resourceref" />
      </xs:choice>
     </xs:complexType>
```

```
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
 Output normalized distance value "result" (scalar).
        ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="scalar" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
   <xs:attribute name="scalaroutput" type="xs:string" use="required" />
   <xs:attribute name="vectorinput" type="xs:string" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!-- signed distance to beam lattice -->
<xs:complexType name="CT_SignedDistanceToBeamLattice">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for evaluating the signed distance to a beam lattice. The input must have the identifier "pos" and must be a vector. The output is a scalar and must have the identifier "distance".
  The beam lattice is defined by a resource identifier.
  The distance is positive if the point is outside the beam lattice and negative if the point is inside the beam lattice.

     The optional attribute "accuraterange" specifies a non-negative distance (in model units, same space as the input "pos") around the beam lattice within which the returned signed distance MUST be accurate. Outside this band (i.e., for points p with |distance(p)| > accuraterange), consumers MAY return approximate values or implementation-defined extrapolations. Consumers MUST clamp negative values of @accuraterange to 0.0.

  Example:
    <beamlattice identifier="SignedDistanceToBeamLattice1" displayname="Signed Distance to Beam Lattice 1" accuraterange="2.5">
   <in>
    <vectorref identifier="pos" ref="inputs.pos"/>
    <resourceref identifier="beamlattice" ref="resourceidnode.value"/>
   </in>
   <out>
    <scalar identifier="distance"/>
   </out>
  </beamlattice>
  ]]>
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="CT_Node">
    <xs:all>
     <xs:element name="in" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
  Inputs to the distance to beam lattice function.
         ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:all>
        <xs:element ref="vectorref" />
        <xs:element ref="resourceref" />
```

```
      </xs:all>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
  distance to beam lattice of the inputs
        ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="scalar" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
   <xs:attribute name="accuraterange" type="xs:double" use="optional" default="0.0" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- node is the base type for all nodes in the implicit function tree. -->
<xs:complexType name="CT_Node">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   Base type for all nodes in the implicit function tree.
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="identifier" type="ST_Identifier" use="required" />
 <xs:attribute name="displayname" type="xs:string" use="optional" />
 <xs:attribute name="tag" type="xs:string" use="optional" />
</xs:complexType>


<xs:complexType name="CT_Addition">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for an addition. The inputs must have the identifier "A" and "B",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - scalar + scalar = scalar
  - vector + vector = vector (vector is added componentwise)
  - matrix + matrix = matrix (matrix is added componentwise)

  Example:
  <addition identifier="addition1" displayname="Addition 1">
   <in>
      <scalarref identifier="A" ref="constant1.c1"/>
      <scalarref identifier="B" ref="inputs.radius"/>
   </in>
   <out>
    <scalar identifier="result"/>
   </out>
  </addition>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
```

```
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the addition.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       sum of the added values
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice minOccurs="1" maxOccurs="1">
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Node for a constant scalar value -->
<xs:complexType name="CT_ConstantScalar">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for a constant scalar value. The output must have the identifier "value".
  Example:
  <constant identifier="constant1" displayname="Constant 1" value="1.0">
   <out>
    <scalar identifier="value"/>
   </out>
  </constant>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:sequence>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="scalar" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="value" type="xs:double" use="required" />
```

```
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Node for a constant vector value -->
<xs:complexType name="CT_ConstantVector">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for a constant vector value. The output must have the identifier "vector".
  Example:
  <constvec identifier="constant1" displayname="Constant 1" x="1.0" y="2.0" z="3.0">
   <out>
    <vector identifier="vector"/>
   </out>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:sequence>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="vector" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="x" type="xs:double" use="required" />
   <xs:attribute name="y" type="xs:double" use="required" />
   <xs:attribute name="z" type="xs:double" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Node for a constant 4x4 matrix value -->
<xs:complexType name="CT_ConstantMatrix">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for a constant 4x4 matrix value. The output must have the identifier "matrix".
  Example:
  <constmat identifier="constant1" displayname="identity"
   matrix="1.0 0.0 0.0 0.0
     0.0 1.0 0.0 0.0
     0.0 0.0 1.0 0.0
     0.0 0.0 0.0 1.0">
   <out>
    <matrix identifier="matrix"/>
   </out>
  </constmat>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:sequence>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="matrix" />
      </xs:sequence>
     </xs:complexType>
```

```
    </xs:element>
   </xs:sequence>
   <xs:attribute name="matrix" type="ST_Matrix4x4" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


<!-- CT_ConstResourceID -->
<xs:complexType name="CT_ConstResourceID">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node that defines a resource id that can be used as an input for a resource reference. The output must have the identifier "value".
  Example:
  <resourceid identifier="resourceid1" displayname="Resource Id 1" value="1">
   <out>
    <resourceid identifier="value"/>
   </out>
  </resourceid>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:sequence>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="resourceid" type="CT_ResourceID" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="value" type="ST_ResourceID" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


<!-- Node for composing a vector from 3 scalar values -->
<xs:complexType name="CT_ComposeVector">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for composing a vector from 3 scalar values. The inputs must have the identifiers x, y and z.
  Example:
  <composevector>
   <in>
    <scalarref identifier="x" ref="inputs.x"/>
    <scalarref identifier="y" ref="inputs.y"/>
    <scalarref identifier="z" ref="inputs.z"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </composevector>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
```

```
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="scalarref" minOccurs="3"
         maxOccurs="3" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="out" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="vector" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:all>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <!-- Node for creating a vector from a single scalar value -->
 <xs:complexType name="CT_VectorFromScalar">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
    Node for creating a vector from a single scalar value. The input must have the identifier "A".
    Example:
    <vectorfromscalar>
     <in>
      <scalarref identifier="A" ref="inputs.x"/>
     </in>
     <out>
      <vector identifier="result"/>
     </out>
    </vectorfromscalar>
    ]]>
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="CT_Node">
    <xs:all>
     <xs:element name="in" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="scalarref" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="out" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="vector" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:all>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <!-- Node for decomposing a vector -->
 <xs:complexType name="CT_DecomposeVector">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
```

```
 Node for decomposing a vector into 3 scalar values. The input must have the identifier "A", the outputs must have the identifiers x, y and z.
 Example:
 <decomposevector>
  <in>
   <vectorref identifier="A" ref="inputs.vector"/>
  </in>
  <out>
   <scalar identifier="x"/>
   <scalar identifier="y"/>
   <scalar identifier="z"/>
  </out>
 </decomposevector>
 ]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="vectorref" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="scalar" minOccurs="3"
       maxOccurs="3" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Node for composing a matrix from 16 scalar values -->
<xs:complexType name="CT_ComposeMatrix">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for composing a matrix from 16 scalar values. The inputs must have the following identifiers:
  m00, m01, m02, m03, m10, m11, m12, m13, m20, m21, m22, m23, m30, m31, m32, m33
  Example:
  <composematrix identifier="composeMatrix_0" displayname="composed matrix">
   <in>
    <scalarref identifier="m00" ref="constantM00.value"/>
    <scalarref identifier="m01" ref="constantM01.value"/>
    <scalarref identifier="m02" ref="constantM02.value"/>
    <scalarref identifier="m03" ref="constantM03.value"/>
    <scalarref identifier="m10" ref="constantM10.value"/>
    <scalarref identifier="m11" ref="constantM11.value"/>
    <scalarref identifier="m12" ref="constantM12.value"/>
    <scalarref identifier="m13" ref="constantM13.value"/>
    <scalarref identifier="m20" ref="constantM20.value"/>
    <scalarref identifier="m21" ref="constantM21.value"/>
    <scalarref identifier="m22" ref="constantM22.value"/>
    <scalarref identifier="m23" ref="constantM23.value"/>
    <scalarref identifier="m30" ref="constantM30.value"/>
    <scalarref identifier="m31" ref="constantM31.value"/>
    <scalarref identifier="m32" ref="constantM32.value"/>
    <scalarref identifier="m33" ref="constantM33.value"/>
```

```
  </in>
  <out>
   <matrix identifier="result"/>
  </out>
 </composematrix>
 ]]>

 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="scalarref" minOccurs="16"
       maxOccurs="16" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="matrix" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- Node for composing a Matrix from column vectors -->
<xs:complexType name="CT_MatrixFromColumns">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for composing a matrix from column vectors. The 4th row is set [0,0,0,1]. The inputs must have the following identifiers:
  A, B, C, D.
  Example:
  <matrixfromcolumns identifier="matrixfromcolumns" displayname="composed matrix">
   <in>
    <vectorref identifier="A" ref="vector0.vector"/>
    <vectorref identifier="B" ref="vector1.vector"/>
    <vectorref identifier="C" ref="vector2.vector"/>
    <vectorref identifier="D" ref="vector3.vector"/>
   </in>
   <out>
    <matrix identifier="result"/>
   </out>
  </matrixfromcolumns>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="vectorref" minOccurs="4"
        maxOccurs="4" />
      </xs:sequence>
     </xs:complexType>
```

```
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="matrix" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Node for composing a Matrix from row vectors -->
<xs:complexType name="CT_MatrixFromRows">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Node for composing a Matrix from row vectors. The 4th column is set to (0,0,0,1). The inputs must have the following identifiers:
  A, B, C, D.
  Example:
  <matrixfromrows identifier="matrixfromrows" displayname="composed matrix">
   <in>
    <vectorref identifier="A" ref="vector0.vector"/>
    <vectorref identifier="B" ref="vector1.vector"/>
    <vectorref identifier="C" ref="vector2.vector"/>
    <vectorref identifier="D" ref="vector3.vector"/>
   </in>
   <out>
    <matrix identifier="result"/>
   </out>
  </matrixfromrows>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="vectorref" minOccurs="4"
        maxOccurs="4" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="matrix" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


 <!-- Nodes for arithmetic operations -->
 <xs:complexType name="CT_Multiplication">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
```

```
  Performs the multiplication A x B = result.
  The inputs must have the identifier "A" and "B", the output must have the identifier "result". If only one of the inputs is a scalar, the operation is componentwise. The following combinations of inputs and outputs are allowed:
  - scalar * scalar = scalar
  - vector * vector = vector (vector is multiplied componentwise)
  - matrix * matrix = matrix (matrix is multiplied componentwise)

  Example:
  <multiplication identifier="multiplication1" displayname="Multiplication 1">
   <in>
    <scalarref identifier="A" ref="constant1.c1"/>
    <scalarref identifier="B" ref="inputs.radius"/>
   </in>
   <out>
    <scalar identifier="result"/>
   </out>
  </multiplication>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the multiplication.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       product of the multiplied values
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!--subtraction-->
<xs:complexType name="CT_Subtraction">
 <xs:annotation>
  <xs:documentation>
```

```
 <![CDATA[
 Derived node for a subtraction. The inputs must have the identifier "A" and "B",
 the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
 - scalar - scalar = scalar
 - vector - vector = vector (vector is subtracted componentwise)
 - matrix - matrix = matrix (matrix is subtracted componentwise)

 Example:
 <subtraction identifier="subtraction1" displayname="Subtraction 1">
  <in>
   <scalarref identifier="A" ref="constant1.c1"/>
   <scalarref identifier="B" ref="inputs.radius"/>
  </in>
  <out>
   <scalar identifier="result"/>
  </out>
 </subtraction>
 ]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      Inputs to the subtraction.
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
      <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
      <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      difference of the subtracted values
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element ref="scalar" />
      <xs:element ref="vector" />
      <xs:element ref="matrix" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- division-->
<xs:complexType name="CT_Division">
 <xs:annotation>
  <xs:documentation>
```

```
 <![CDATA[
Derived node for a division. The inputs must have the identifier "A" and "B",
the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
- scalar / scalar = scalar
- vector / vector = vector (vector is divided componentwise)
- matrix / matrix = matrix (matrix is divided componentwise)

Example:
<division identifier="division1" displayname="Division 1">
 <in>
  <scalarref identifier="A" ref="constant1.c1"/>
  <scalarref identifier="B" ref="inputs.radius"/>
 </in>
 <out>
  <scalar identifier="result"/>
 </out>
</division>
]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      Inputs to the division.
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
      <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
      <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      quotient of the divided values
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element ref="scalar" />
      <xs:element ref="vector" />
      <xs:element ref="matrix" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- dot product-->
<xs:complexType name="CT_DotProduct">
 <xs:annotation>
  <xs:documentation>
```

```
 <![CDATA[
 Derived node for a dot product with a scalar as output and two vector inputs. The inputs must have the identifier "A" and "B",
 the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
 - vector * vector = scalar (dot product of the two vectors)

 Example:
 <dot identifier="dotproduct1" displayname="Dot Product 1">
  <in>
   <vectorref identifier="A" ref="inputs.vector1"/>
   <vectorref identifier="B" ref="inputs.vector2"/>
  </in>
  <out>
   <scalar identifier="result"/>
  </out>
 </dot>
 ]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      Inputs to the dot product.
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:sequence>
      <xs:element ref="vectorref" />
      <xs:element ref="vectorref" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      dot product of the two vectors
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:sequence>
      <xs:element ref="scalar" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- cross product-->
<xs:complexType name="CT_CrossProduct">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a cross product. The inputs must have the identifier "A" and "B",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - vector x vector = vector (cross product of the two vectors)
```

```
  Example:
  <cross identifier="crossproduct1" displayname="Cross Product 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
    <vectorref identifier="B" ref="inputs.vector2"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </cross>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the cross product.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="vectorref" />
       <xs:element ref="vectorref" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       cross product of the two vectors
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="vector" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

 <!-- Matrix multiplication -->
 <xs:complexType name="CT_MatrixVectorMultiplication">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
   Performs the matrix vector multiplication A x B = result. The output must be a vector with the name "result". The following combinations of
 inputs and outputs are allowed:
   - matrix * vector = vector

   Example:
   <matrixvectormultiplication identifier="matVec1" displayname="Matrix Vector Multiplication 1">
    <in>
     <matrixref identifier="A" ref="inputs.matrix1"/>
     <vectorref identifier="B" ref="inputs.vector1"/>
```

```
  </in>
  <out>
   <vector identifier="result"/>
  </out>
 </matrixvectormultiplication>
 ]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      Inputs to the matrix vector multiplication.
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:all>
      <xs:element ref="matrixref" />
      <xs:element ref="vectorref" />
     </xs:all>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      product of the matrix and the vector
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:sequence>
      <xs:element ref="vector" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- Transpose -->
<xs:complexType name="CT_Transpose">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Performs the transpose of a matrix. The input 'A' must be a matrix and the output must be a matrix with the identifier "result".

  Example:
  <transpose identifier="transpose1" displayname="Transpose 1">
   <in>
    <matrixref identifier="A" ref="inputs.matrix1"/>
   </in>
   <out>
    <matrix identifier="result"/>
   </out>
  </transpose>
  ]]>
  </xs:documentation>
 </xs:annotation>
```

```
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the transpose.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="matrixref" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       transpose of the matrix
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="matrix" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Inverse -->
<xs:complexType name="CT_Inverse">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   Computes the inverse of a matrix. The input 'A' must be a matrix and the output must be a matrix with the identifier "result".

   Example:
   <inverse identifier="inverse1" displayname="Inverse 1">
    <in>
     <matrixref identifier="A" ref="inputs.matrix1"/>
    </in>
    <out>
     <matrix identifier="result"/>
    </out>
   </inverse>
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the inverse.
```

```
      ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="matrixref" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       inverse of the matrix
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:sequence>
       <xs:element ref="matrix" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- sinus function for scalars or componentwise for vectors -->
<xs:complexType name="CT_Sinus">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a sine function with a scalar or vector as output and a scalar or vector input. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - sin(scalar) = scalar
  - sin(vector) = vector (sine of each component of the vector)

  Example:
  <sin identifier="sinus1" displayname="Sinus 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </sin>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the sinus function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
```

```
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       sinus of the input
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- cosinus function for scalars or componentwise for vectors -->
<xs:complexType name="CT_Cosinus">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a cosinus function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - cos(scalar) = scalar
  - cos(vector) = vector (cosinus of each component of the vector)

  Example:
  <cos identifier="cosinus1" displayname="Cosinus 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </cos>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the cosine function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
```

```
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      cosine of the input
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalar" />
      <xs:element ref="vector" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- tan function for scalars or componentwise for vectors -->
<xs:complexType name="CT_Tan">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a tan function with a scalar or vector as output and a scalar or vector input. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - tan(scalar) = scalar
  - tan(vector) = vector (tan of each component of the vector)

  Example:
  <tan identifier="tan1" displayname="Tan 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </tan>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the tan function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
```

```
      tan of the input
      ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- asin function for scalars or componentwise for vectors -->
<xs:complexType name="CT_ASin">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   Derived node for an arcsine (asin) function with a scalar or vector as output and a scalar or vector input. The input must have the identifier
"A",
   the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
   - asin(scalar) = scalar
   - asin(vector) = vector (asin of each component of the vector)

   Example:
   <asin identifier="asin1" displayname="Asin 1">
    <in>
     <vectorref identifier="A" ref="inputs.vector1"/>
    </in>
    <out>
     <vector identifier="result"/>
    </out>
   </asin>
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the asin function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       asin of the input
       ]]>
       </xs:documentation>
```

```
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- acos function for scalars or componentwise for vectors -->
<xs:complexType name="CT_ACos">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for an arccosine (acos) function with a scalar or vector as output and a scalar or vector input. The input must have the
identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - acos(scalar) = scalar
  - acos(vector) = vector (acos of each component of the vector)

  Example:
  <acos identifier="acos1" displayname="Acos 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </acos>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the acos function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       acos of the input
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
```

```
      <xs:element ref="vector" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- atan function for scalars or componentwise for vectors -->
<xs:complexType name="CT_ATan">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for an arctangent (atan) function with a scalar or vector as output and a scalar or vector input. The input must have the
identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - atan(scalar) = scalar
  - atan(vector) = vector (atan of each component of the vector)

  Example:
  <atan identifier="atan1" displayname="Atan 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </atan>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the atan function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       atan of the input
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
      </xs:choice>
     </xs:complexType>
```

```
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


<!-- atan2 function -->
<xs:complexType name="CT_ATan2">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for an atan2 function. The inputs must have the identifier "A" and "B",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - atan2(scalar,scalar) = scalar
  - atan2(vector,vector) = vector (atan2 of each component of the vectors)

  Example:
  <atan2 identifier="atan21" displayname="Atan2 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
    <vectorref identifier="B" ref="inputs.vector2"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </atan2>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the atan2 function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
        atan2 of the input
        ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
```

```
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


<!-- min function -->
<xs:complexType name="CT_Min">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a min function. The inputs must have the identifier "A" and "B",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - min(scalar,scalar) = scalar
  - min(vector,vector) = vector (min of each component of the vectors)
  - min(matrix,matrix) = matrix (min of each component of the matrices)

  Example:
  <min identifier="min1" displayname="Min 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
    <vectorref identifier="B" ref="inputs.vector2"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </min>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the min function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       min of the inputs
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
```

```
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- max function -->
<xs:complexType name="CT_Max">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a max function. The inputs must have the identifier "A" and "B",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - min(scalar,scalar) = scalar
  - min(vector,vector) = vector (min of each component of the vectors)
  - min(matrix,matrix) = matrix (min of each component of the matrices)

  Example:
  <max identifier="max1" displayname="Max 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
    <vectorref identifier="B" ref="inputs.vector2"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </max>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the max function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       max of the inputs
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
```

```
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- absolute value  -->
<xs:complexType name="CT_Abs">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for an absolute value function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - abs(scalar) = scalar
  - abs(vector) = vector (absolute value of each component of the vector)
  - abs(matrix) = matrix (absolute value of each component of the matrix)

  Example:
  <abs identifier="abs1" displayname="Abs 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </abs>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the absolute value function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
       <xs:element ref="matrixref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       absolute value of the input
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
```

```
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- fmod(A,B) -->
<xs:complexType name="CT_Fmod">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a fmod function. The input must have the identifier "A" and "B",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - fmod(scalar,scalar) = scalar
  - fmod(vector,vector) = vector (fmod of each component of the vectors)
  - fmod(matrix,matrix) = matrix (fmod of each component of the matrices)

  Example:
  <fmod identifier="fmod1" displayname="Fmod 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
    <vectorref identifier="B" ref="inputs.vector2"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </fmod>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the fmod function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       fmod of the inputs
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
```

```
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- fmod(A,B) -->
<xs:complexType name="CT_Mod">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   Derived node for a mod function. While fmod is the 'C' fmod, mod has the same behaviour as the glsl mod function. The input must have the identifier "A" and "B",
   the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
   - fmod(scalar,scalar) = scalar
   - fmod(vector,vector) = vector (fmod of each component of the vectors)
   - fmod(matrix,matrix) = matrix (fmod of each component of the matrices)

   Example:
   <fmod identifier="mod1" displayname="mod 1">
    <in>
     <vectorref identifier="A" ref="inputs.vector1"/>
     <vectorref identifier="B" ref="inputs.vector2"/>
    </in>
    <out>
     <vector identifier="result"/>
    </out>
   </fmod>
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the mod function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       mod of the inputs
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
```

```
    </xs:all>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <!-- pow(base,exponent) -->
 <xs:complexType name="CT_Pow">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
   Derived node for a pow function. The input must have the identifier "A" (base) and "B" (exponent),
   the output must have the identifier "result". Inputs and output must have the same dimension. The following combinations of inputs and
outputs are allowed:
   - pow(scalar,scalar) = scalar
   - pow(vector,vector) = vector (pow of each component of the vectors)
   - pow(matrix,matrix) = matrix (pow of each component of the matrices)

   Example:
   <pow identifier="pow1" displayname="Pow 1">
    <in>
     <vectorref identifier="A" ref="inputs.vector1"/>
     <vectorref identifier="B" ref="inputs.vector2"/>
    </in>
    <out>
     <vector identifier="result"/>
    </out>
   </pow>
   ]]>
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="CT_Node">
    <xs:all>
     <xs:element name="in" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
        Inputs to the pow function.
        ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:choice>
        <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
        <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
        <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
       </xs:choice>
      </xs:complexType>
     </xs:element>
     <xs:element name="out" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
        pow of the inputs
        ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:choice>
        <xs:element ref="scalar" />
        <xs:element ref="vector" />
        <xs:element ref="matrix" />
       </xs:choice>
      </xs:complexType>
```

```
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- sqrt(A) -->
<xs:complexType name="CT_Sqrt">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a sqrt function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - sqrt(scalar) = scalar
  - sqrt(vector) = vector (sqrt of each component of the vector)
  - sqrt(matrix) = matrix (sqrt of each component of the matrix)

  Example:
  <sqrt identifier="sqrt1" displayname="Sqrt 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </sqrt>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the sqrt function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="vectorref" minOccurs="2" maxOccurs="2" />
       <xs:element ref="matrixref" minOccurs="2" maxOccurs="2" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       sqrt of the inputs
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
```

```
    </xs:all>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <!-- distance to mesh -->
 <xs:complexType name="CT_SignedDistanceToMesh">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
   Node for evaluating the signed distance to a mesh. The input must have the identifier "pos" and must be a vector. The output is a scalar and must have the identifier "distance".
   The mesh is defined by a resource identifier. The mesh is assumed to be closed and watertight.
   The distance is positive if the point is outside the mesh and negative if the point is inside the mesh.

   Example:
   <mesh identifier="signedDistanceToMesh1" displayname="Signed Distance to Mesh 1">
    <in>
     <vectorref identifier="pos" ref="inputs.pos"/>
     <resourceref identifier="mesh" ref="resourceidnode.value"/>
    </in>
    <out>
     <scalar identifier="distance"/>
    </out>
   </mesh>
   ]]>
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="CT_Node">
    <xs:all>
     <xs:element name="in" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
        Inputs to the distance to mesh function.
        ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:all>
        <xs:element ref="vectorref" />
        <xs:element name="resourceref" type="ST_ResourceID" />
       </xs:all>
      </xs:complexType>
     </xs:element>
     <xs:element name="out" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
        distance to mesh of the inputs
        ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:sequence>
        <xs:element ref="scalar" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:all>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

```
 <!-- unsigned distance to mesh -->
 <xs:complexType name="CT_UnsignedDistanceToMesh">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
   Node for evaluating the unsigned distance to a mesh. The input must have the identifier "pos" and must be a vector. The output is a scalar and must have the identifier "distance".
   The mesh is defined by a resource identifier. The mesh may be open and is not required to be watertight.
   The distance is always positive.

   Example:
   <unsignedmesh identifier="UnsignedDistanceToMesh1" displayname="Unsigned Distance to Mesh 1">
    <in>
     <vectorref identifier="pos" ref="inputs.pos"/>
     <resourceref identifier="mesh" ref="resourceidnode.value"/>
    </in>
    <out>
     <scalar identifier="distance"/>
    </out>
   </unsignedmesh>
   ]]>
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="CT_Node">
    <xs:all>
     <xs:element name="in" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
        Inputs to the distance to mesh function.
        ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:all>
        <xs:element ref="vectorref" />
        <xs:element name="resourceref" type="ST_ResourceID" />
       </xs:all>
      </xs:complexType>
     </xs:element>
     <xs:element name="out" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
        unsigned distance to mesh of the inputs
        ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:sequence>
        <xs:element ref="scalar" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:all>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <!-- length of a vector -->
 <xs:complexType name="CT_Length">
  <xs:annotation>
```

```
 <xs:documentation>
  <![CDATA[
 Derived node for a length function. The input must have the identifier "A",
 the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
 - length(vector) = scalar

 Example:
 <length identifier="length1" displayname="Length 1">
  <in>
   <vectorref identifier="A" ref="inputs.vector1"/>
  </in>
  <out>
   <scalar identifier="result"/>
  </out>
 </length>
 ]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      Inputs to the length function.
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:sequence>
      <xs:element ref="vectorref" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      length of the input vector
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:sequence>
      <xs:element ref="scalar" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- FunctionGradient: computes gradient of a referenced function -->
<xs:complexType name="CT_FunctionGradient">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
 Derived node for computing the spatial gradient of a referenced function. The inputs must include a resource reference with the
 identifier "functionID" and all argument references required by the referenced function. The attribute "scalaroutput" must name
 a scalar output of the referenced function. The attribute "vectorinput" must name a vector (float3) input of the referenced function
 with respect to which the gradient is computed.
```

```
  The outputs must have the identifiers "vector", "gradient" and "magnitude" where:
  - "vector" is the normalized gradient (float3)
  - "gradient" is the raw gradient (float3)
  - "magnitude" is the length of the gradient (scalar)

  The finite difference step is provided as a scalar input named "step". Consumers MUST clamp the effective step to
  >= 1e-8. If the input is missing or evaluates to NaN/Inf, the consumer MUST treat the result as undefined. Central finite differences are used:
for each component c ∈ {x,y,z}, evaluate the referenced scalar output at the selected
  vector input offset by ±step along c and compute (f(x+h_c) - f(x-h_c)) / (2·step).

  Example:
  <functiongradient identifier="grad1" displayname="Gradient of f" scalaroutput="distance" vectorinput="pos">
   <in>
    <resourceref identifier="functionID" ref="functionIDNode.value"/>
    <vectorref identifier="pos" ref="inputs.pos"/>
    <scalarref identifier="radius" ref="inputs.radius"/>
    <scalarref identifier="step" ref="inputs.step"/>
   </in>
   <out>
    <vector identifier="vector"/>
    <vector identifier="gradient"/>
    <scalar identifier="magnitude"/>
   </out>
  </functiongradient>
  ]]>
   </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="CT_Node">
    <xs:all>
     <xs:element name="in" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
  Inputs to the function gradient. Must include a resourceref with identifier "functionID", a scalarref with identifier "step", and the argument
references required
  by the referenced function. The identifier of the vector input to differentiate must match the value of the @vectorinput attribute.
         ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:choice minOccurs="1" maxOccurs="2147483647">
        <xs:element ref="scalarref" />
        <xs:element ref="vectorref" />
        <xs:element ref="matrixref" />
        <xs:element ref="resourceref" />
       </xs:choice>
      </xs:complexType>
     </xs:element>
     <xs:element name="out" minOccurs="1" maxOccurs="1">
      <xs:complexType>
       <xs:annotation>
        <xs:documentation>
         <![CDATA[
  Outputs of the function gradient: normalized gradient ("vector"), raw gradient ("gradient"), and gradient length ("magnitude").
         ]]>
        </xs:documentation>
       </xs:annotation>
       <xs:sequence>
        <xs:element ref="vector" />
        <xs:element ref="vector" />
        <xs:element ref="scalar" />
       </xs:sequence>
      </xs:complexType>
```

```
    </xs:element>
   </xs:all>
   <xs:attribute name="scalaroutput" type="xs:string" use="required" />
   <xs:attribute name="vectorinput" type="xs:string" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- base for log, exp etc. -->
<xs:complexType name="CT_BaseOneParameterFunc">
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the function.
       ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
       <xs:element ref="matrixref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
        result of the function
        ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice>
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- natural log-->
<xs:complexType name="CT_Log">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a natural log function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - log(scalar) = scalar
  - log(vector) = vector (ln of each component of the vector)
  - log(matrix) = matrix (ln of each component of the matrix)

  Example:
  <log identifier="log_1" displayname="Log_n 1">
   <in>
```

```
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </log>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- log2 -->
<xs:complexType name="CT_Log2">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a log2 function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - log2(scalar) = scalar
  - log2(vector) = vector (log2 of each component of the vector)
  - log2(matrix) = matrix (log2 of each component of the matrix)

  Example:
  <log2 identifier="log21" displayname="Log2 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </log2>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>


<!-- log10 -->
<xs:complexType name="CT_Log10">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a log10 function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - log10(scalar) = scalar
  - log10(vector) = vector (log10 of each component of the vector)
  - log10(matrix) = matrix (log10 of each component of the matrix)

  Example:
  <log10 identifier="log101" displayname="Log10 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </log10>
  ]]>
```

```
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- exp -->
<xs:complexType name="CT_Exp">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for an exponential function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - exp(scalar) = scalar
  - exp(vector) = vector (exp of each component of the vector)
  - exp(matrix) = matrix (exp of each component of the matrix)

  Example:
  <exp identifier="exp1" displayname="Exp 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </exp>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- cosh-->
<xs:complexType name="CT_Cosh">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a hyperbolic cosine function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - cosh(scalar) = scalar
  - cosh(vector) = vector (cosh of each component of the vector)
  - cosh(matrix) = matrix (cosh of each component of the matrix)

  Example:
  <cosh identifier="cosh1" displayname="Cosh 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </cosh>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- sinh-->
```

```
<xs:complexType name="CT_Sinh">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a hyperbolic sine function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - sinh(scalar) = scalar
  - sinh(vector) = vector (sinh of each component of the vector)
  - sinh(matrix) = matrix (sinh of each component of the matrix)

  Example:
  <sinh identifier="sinh1" displayname="Sinh 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </sinh>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- tanh-->
<xs:complexType name="CT_Tanh">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a hyperbolic tangent function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - tanh(scalar) = scalar
  - tanh(vector) = vector (tanh of each component of the vector)
  - tanh(matrix) = matrix (tanh of each component of the matrix)

  Example:
  <tanh identifier="tanh1" displayname="Tanh 1">
   <in>
      <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </tanh>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

 <!-- clamp -->
 <xs:complexType name="CT_Clamp">
  <xs:annotation>
   <xs:documentation>
    <![CDATA[
   Derived node for a clamp function max(min, min(A,max)). The input must have the identifier "A" (value), "min" and "max",
   the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
   - clamp(scalar,scalar,scalar) = scalar
   - clamp(vector,vector,vector) = vector (clamp of each component of the vectors)
```

```
- clamp(matrix,matrix,matrix) = matrix (clamp of each component of the matrices)
Example:
<clamp identifier="clamp1" displayname="Clamp 1">
 <in>
  <vectorref identifier="A" ref="inputs.vector1"/>
  <vectorref identifier="min" ref="inputs.vector2"/>
  <vectorref identifier="max" ref="inputs.vector3"/>
 </in>
 <out>
  <vector identifier="result"/>
 </out>
</clamp>
]]>
</xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
      Inputs to the clamp function.
      ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalarref" minOccurs="3" maxOccurs="3" />
      <xs:element ref="vectorref" minOccurs="3" maxOccurs="3" />
      <xs:element ref="matrixref" minOccurs="3" maxOccurs="3" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
       result of clamp(value,min,max)
       ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalar" />
      <xs:element ref="vector" />
      <xs:element ref="matrix" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>


<!-- select -->
<xs:complexType name="CT_Select">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a select function. The input must have the identifier "A", "B", "C", "D",
  the output must have the identifier "result". If A < B then the result is C, otherwise D. The following combinations of inputs and outputs are
allowed:
```

```
 - select(scalar,scalar,scalar,scalar) = scalar
 - select(vector,vector,vector,vector) = vector (select of each component of the vectors)
 - select(matrix,matrix,matrix,matrix) = matrix (select of each component of the matrices)

 Example:
 <select identifier="select1" displayname="Select 1">
  <in>
     <vectorref identifier="A" ref="inputs.vector1"/>
     <vectorref identifier="B" ref="inputs.vector2"/>
     <vectorref identifier="C" ref="inputs.vector3"/>
     <vectorref identifier="D" ref="inputs.vector4"/>
  </in>
  <out>
   <vector identifier="result"/>
  </out>
 </select>
 ]]>
 </xs:documentation>
</xs:annotation>
<xs:complexContent>
 <xs:extension base="CT_Node">
  <xs:all>
   <xs:element name="in" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
     Inputs to the select function.
     ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalarref" minOccurs="4" maxOccurs="4" />
      <xs:element ref="vectorref" minOccurs="4" maxOccurs="4" />
      <xs:element ref="matrixref" minOccurs="4" maxOccurs="4" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
   <xs:element name="out" minOccurs="1" maxOccurs="1">
    <xs:complexType>
     <xs:annotation>
      <xs:documentation>
       <![CDATA[
       result of A < B ? C : D
       ]]>
      </xs:documentation>
     </xs:annotation>
     <xs:choice>
      <xs:element ref="scalar" />
      <xs:element ref="vector" />
      <xs:element ref="matrix" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:all>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- round -->
<xs:complexType name="CT_Round">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
```

```
  Derived node for a round function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - round(scalar) = scalar
  - round(vector) = vector (round of each component of the vector)
  - round(matrix) = matrix (round of each component of the matrix)

  Example:
  <round identifier="round1" displayname="Round 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </round>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>


<!-- ceil-->
<xs:complexType name="CT_Ceil">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a ceil function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - ceil(scalar) = scalar
  - ceil(vector) = vector (ceil of each component of the vector)
  - ceil(matrix) = matrix (ceil of each component of the matrix)

  Example:
  <ceil identifier="ceil1" displayname="Ceil 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </ceil>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- floor -->
<xs:complexType name="CT_Floor">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a floor function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - floor(scalar) = scalar
  - floor(vector) = vector (floor of each component of the vector)
  - floor(matrix) = matrix (floor of each component of the matrix)

  Example:
```

```
  <floor identifier="floor1" displayname="Floor 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </floor>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- sign -->
<xs:complexType name="CT_Sign">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Derived node for a sign function. The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - sign(scalar) = scalar
  - sign(vector) = vector (sign of each component of the vector)
  - sign(matrix) = matrix (sign of each component of the matrix)

  Example:
  <sign identifier="sign1" displayname="Sign 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </sign>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- fract(A) -->
<xs:complexType name="CT_Fract">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Returns the fractional part of the input (A- floor(A)). The input must have the identifier "A",
  the output must have the identifier "result". The following combinations of inputs and outputs are allowed:
  - sign(scalar) = scalar
  - sign(vector) = vector (sign of each component of the vector)
  - sign(matrix) = matrix (sign of each component of the matrix)

  Example:
  <sign identifier="sign1" displayname="Sign 1">
   <in>
    <vectorref identifier="A" ref="inputs.vector1"/>
   </in>
   <out>
    <vector identifier="result"/>
   </out>
  </sign>
```

```
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_BaseOneParameterFunc"> </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Scalar input value -->
<xs:complexType name="CT_Scalar">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Scalar output value
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="identifier" type="ST_Identifier" use="required" />
 <xs:attribute name="displayname" type="xs:string" use="optional" />
</xs:complexType>


<!-- Vector input value -->
<xs:complexType name="CT_Vector">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   Vector output value
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="identifier" type="ST_Identifier" use="required" />
 <xs:attribute name="displayname" type="xs:string" use="optional" />
</xs:complexType>

<!-- Matrix input value -->
<xs:complexType name="CT_Matrix">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   Matrix output value
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="identifier" type="ST_Identifier" use="required" />
 <xs:attribute name="displayname" type="xs:string" use="optional" />
</xs:complexType>

<!-- ResourceId input -->
<xs:complexType name="CT_ResourceID">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
   ResourceId output
   ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="identifier" type="ST_Identifier" use="required" />
 <xs:attribute name="displayname" type="xs:string" use="optional" />
</xs:complexType>

<xs:complexType name="CT_Ref">
 <xs:annotation>
  <xs:documentation>
```

```
   <![CDATA[
  Base for reference types
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:attribute name="identifier" type="ST_Identifier" use="required" />
 <xs:attribute name="displayname" type="xs:string" use="optional" />
</xs:complexType>

<xs:complexType name="CT_ScalarRef">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Reference to a scalar output.
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Ref">
   <xs:attribute name="ref" type="ST_ScalarID" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="CT_VectorRef">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Reference to a vector output
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Ref">
   <xs:attribute name="ref" type="ST_VectorID" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="CT_MatrixRef">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Reference to a 4x4 Matrix
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Ref">
   <xs:attribute name="ref" type="ST_MatrixID" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="CT_ResourceRef">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Reference to a resource
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Ref">
```

```
   <xs:attribute name="ref" type="ST_ResourceOutputID" use="required" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="CT_Input">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
 Inputs to the function.
 ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:choice minOccurs="0" maxOccurs="2147483647">
  <xs:element ref="scalar"/>
  <xs:element ref="vector"/>
  <xs:element ref="matrix"/>
  <xs:element ref="resourceid"/>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>

 </xs:choice>
</xs:complexType>

<xs:complexType name="CT_Output">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
    References to the outputs of the function.
    ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:choice minOccurs="1" maxOccurs="2147483647">
  <xs:element ref="scalarref" minOccurs="0" maxOccurs="2147483647" />
  <xs:element ref="vectorref" minOccurs="0" maxOccurs="2147483647" />
  <xs:element ref="matrixref" minOccurs="0" maxOccurs="2147483647" />
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>

 </xs:choice>
</xs:complexType>

<xs:group name="BasicNodeTypes">
 <xs:choice>
  <!-- nodes -->
  <xs:element ref="addition" />
  <xs:element ref="subtraction" />
  <xs:element ref="multiplication" />
  <xs:element ref="division" />
  <xs:element ref="constant" />
  <xs:element ref="constvec" />
  <xs:element ref="constmat" />
  <xs:element ref="composevector" />
  <xs:element ref="vectorfromscalar" />
  <xs:element ref="decomposevector" />
  <xs:element ref="composematrix" />
  <xs:element ref="matrixfromcolumns" />
  <xs:element ref="matrixfromrows" />
  <xs:element ref="dot" />
  <xs:element ref="cross" />
  <xs:element ref="matvecmultiplication" />
  <xs:element ref="transpose" />
  <xs:element ref="inverse" />
  <xs:element ref="sin" />
  <xs:element ref="cos" />
  <xs:element ref="tan" />
```

```
  <xs:element ref="asin" />
  <xs:element ref="acos" />
  <xs:element ref="atan" />
  <xs:element ref="atan2" />
  <xs:element ref="min" />
  <xs:element ref="max" />
  <xs:element ref="abs" />
  <xs:element ref="fmod" />
  <xs:element ref="pow" />
  <xs:element ref="sqrt" />
  <xs:element ref="exp" />
  <xs:element ref="log" />
  <xs:element ref="log2" />
  <xs:element ref="log10" />
  <xs:element ref="select" />
  <xs:element ref="clamp" />
  <xs:element ref="cosh" />
  <xs:element ref="sinh" />
  <xs:element ref="tanh" />
  <xs:element ref="round" />
  <xs:element ref="ceil" />
  <xs:element ref="floor" />
  <xs:element ref="sign" />
  <xs:element ref="fract" />
  <xs:element ref="functiongradient" />
  <xs:element ref="normalizedistance" />
  <xs:element ref="functioncall" />
  <xs:element ref="mesh" />
  <xs:element ref="unsignedmesh" />
  <xs:element ref="beamlattice" />
  <xs:element ref="length" />
  <xs:element ref="resourceid" />
  <xs:element ref="constresourceid" />
  <xs:element ref="mod" />
  <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>

 </xs:choice>
</xs:group>

<!-- implicit function -->
<xs:complexType name="CT_ImplicitFunction">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Implicit function with an arbitrary number of inputs and outputs. The inputs and outputs are defined by the user.
  No forward references allowed.

  Example:
  <implicitfunction identifier="sphere" displayname="Sphere">
   <in>
    <vector identifier="coordinates" displayname="Coordinates"/>
    <scalar identifier="radius" displayname="radius"/>
   </in>

   <length identifier="length1" displayname="length1">
    <in>
     <vectorref identifier="coordinates" ref="inputs.coordinates"/>
    </in>
    <out>
     <scalar identifier="value" displayname="length"></scalar>
    </out>
   </length>

   <subtraction identifier="sub1" displayname="subtraction">
```

```
    <in>
     <scalarref identifier="A" ref="length1.value"/>
     <scalarref identifier="B" ref="inputs.radius"/>
    </in>
    <out>
     <scalar identifier="difference" displayname="difference"></scalar>
    </out>
   </subtraction>
   <out>
    <vectorref identifier="distance" ref="sub1.difference"/>
   </out>

  </implicitfunction>

  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="vol:CT_Function">
   <xs:sequence>
    <xs:element name="in" type="CT_Input" />
    <xs:choice minOccurs="0" maxOccurs="2147483647">
     <xs:group ref="BasicNodeTypes"  />
    </xs:choice>
    <xs:element name="out" type="CT_Output" />
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="2147483647"/>

   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- node for calling a function -->
<xs:complexType name="CT_FunctionCall">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Calls a function. It must at least have an resourceref input with the identifier "functionID".
  Example:
  <functioncall identifier="sphere" displayname="Sphere">
   <in>
    <resourceref identifier="functionID" ref="functionIDNode.value" />
    <scalarref identifier="pos" ref="othernode_0.result" />
    <scalarref identifier="radius" ref="othernode_1.result" />
   </in>
   <out>
    <scalarref identifier="result" ref="mySphereFunction.distance"/>
   </out>
  </functioncall>
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="CT_Node">
   <xs:all>
    <xs:element name="in" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
       Inputs to the function.
       ]]>
       </xs:documentation>
      </xs:annotation>
```

```
      <xs:choice minOccurs="1" maxOccurs="2147483647">
       <xs:element ref="scalarref" />
       <xs:element ref="vectorref" />
       <xs:element ref="matrixref" />
       <xs:element ref="resourceref" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="out" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:annotation>
       <xs:documentation>
        <![CDATA[
        Outputs of the function.
        ]]>
       </xs:documentation>
      </xs:annotation>
      <xs:choice minOccurs="1" maxOccurs="2147483647">
       <xs:element ref="scalar" />
       <xs:element ref="vector" />
       <xs:element ref="matrix" />
       <xs:element ref="resourceid" />
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- Simple Types -->
<xs:simpleType name="ST_ResourceID">
 <xs:restriction base="xs:positiveInteger">
  <xs:maxExclusive value="2147483648" />
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ST_Matrix4x4">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  String containing 16 space separated floating numbers.
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="xs:QName">
  <xs:pattern
   value="^(-?\d+(\.\d+)?((e|E)[+-]?\d+)?)((\s+-?\d+(\.\d+)?((e|E)[+-]?\d+)?) {15})$" />
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ST_Identifier">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Identifier for referencing a node or a node output.
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="xs:QName">
  <xs:pattern value="[a-zA-Z0-9_]+" />
 </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="ST_NodeOutputIdentifier">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Identifier for a node output of the form "nodename.outputname".
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="xs:QName">
  <!-- pattern allowing refs of the form "nodename.outputname" -->
  <xs:pattern value="[a-zA-Z0-9_]+(\.[a-zA-Z0-9_]+)?" />
 </xs:restriction>
</xs:simpleType>

<!-- Identifier for a scalar output -->
<xs:simpleType name="ST_ScalarID">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Identifier for a scalar output
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="ST_NodeOutputIdentifier" />
</xs:simpleType>

<!-- Identifier for a vector output -->
<xs:simpleType name="ST_VectorID">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Identifier for a vector output
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="ST_NodeOutputIdentifier" />
</xs:simpleType>

<!-- Identifier for a Matrix-->
<xs:simpleType name="ST_MatrixID">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Identifier for a 4x4 Matrix
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="ST_NodeOutputIdentifier" />
</xs:simpleType>

<xs:simpleType name="ST_ResourceOutputID">
 <xs:annotation>
  <xs:documentation>
   <![CDATA[
  Identifier of an output providing a resource id
  ]]>
  </xs:documentation>
 </xs:annotation>
 <xs:restriction base="ST_NodeOutputIdentifier" />
</xs:simpleType>

<!-- Elements -->
<xs:element name="resources" type="CT_Resources" />
```

```
<xs:element name="implicitfunction" type="CT_ImplicitFunction" />
<xs:element name="scalar" type="CT_Scalar" />
<xs:element name="vector" type="CT_Vector" />
<xs:element name="matrix" type="CT_Matrix" />
<xs:element name="resourceid" type="CT_ResourceID" />
<xs:element name="in" type="CT_Input" />
<xs:element name="out" type="CT_Output" />

<xs:element name="scalarref" type="CT_ScalarRef" />
<xs:element name="vectorref" type="CT_VectorRef" />
<xs:element name="matrixref" type="CT_MatrixRef" />
<xs:element name="resourceref" type="CT_ResourceRef" />

<xs:element name="addition" type="CT_Addition" />
<xs:element name="subtraction" type="CT_Subtraction" />
<xs:element name="multiplication" type="CT_Multiplication" />
<xs:element name="division" type="CT_Division" />
<xs:element name="constant" type="CT_ConstantScalar" />
<xs:element name="constvec" type="CT_ConstantVector" />
<xs:element name="constmat" type="CT_ConstantMatrix" />
<xs:element name="composevector" type="CT_ComposeVector" />
<xs:element name="vectorfromscalar" type="CT_VectorFromScalar" />
<xs:element name="decomposevector" type="CT_DecomposeVector" />
<xs:element name="composematrix" type="CT_ComposeMatrix" />
<xs:element name="matrixfromcolumns" type="CT_MatrixFromColumns" />
<xs:element name="matrixfromrows" type="CT_MatrixFromRows" />
<xs:element name="dot" type="CT_DotProduct" />
<xs:element name="cross" type="CT_CrossProduct" />
<xs:element name="matvecmultiplication" type="CT_MatrixVectorMultiplication" />
<xs:element name="transpose" type="CT_Transpose" />
<xs:element name="inverse" type="CT_Inverse" />
<xs:element name="sin" type="CT_Sinus" />
<xs:element name="cos" type="CT_Cosinus" />
<xs:element name="tan" type="CT_Tan" />
<xs:element name="asin" type="CT_ASin" />
<xs:element name="acos" type="CT_ACos" />
<xs:element name="atan" type="CT_ATan" />
<xs:element name="atan2" type="CT_ATan2" />
<xs:element name="min" type="CT_Min" />
<xs:element name="max" type="CT_Max" />
<xs:element name="abs" type="CT_Abs" />
<xs:element name="fmod" type="CT_Fmod" />
<xs:element name="pow" type="CT_Pow" />
<xs:element name="sqrt" type="CT_Sqrt" />
<xs:element name="exp" type="CT_Exp" />
<xs:element name="log" type="CT_Log" />
<xs:element name="log2" type="CT_Log2" />
<xs:element name="log10" type="CT_Log10" />
<xs:element name="select" type="CT_Select" />
<xs:element name="clamp" type="CT_Clamp" />
<xs:element name="cosh" type="CT_Cosh" />
<xs:element name="sinh" type="CT_Sinh" />
<xs:element name="tanh" type="CT_Tanh" />
<xs:element name="round" type="CT_Round" />
<xs:element name="ceil" type="CT_Ceil" />
<xs:element name="floor" type="CT_Floor" />
<xs:element name="sign" type="CT_Sign" />
<xs:element name="fract" type="CT_Fract" />
<xs:element name="functiongradient" type="CT_FunctionGradient" />
<xs:element name="normalizedistance" type="CT_NormalizeDistance" />
<xs:element name="functioncall" type="CT_FunctionCall" />
<xs:element name="mesh" type="CT_SignedDistanceToMesh" />
<xs:element name="unsignedmesh" type="CT_UnsignedDistanceToMesh" />
<xs:element name="beamlattice" type="CT_SignedDistanceToBeamLattice" />
```

```
 <xs:element name="length" type="CT_Length" />
 <xs:element name="constresourceid" type="CT_ConstResourceID" />
 <xs:element name="mod" type="CT_Mod" />

</xs:schema>```
```

sheet1.png


Appendix C. Namespaces

Volumetric http://schemas.3mf.io/3dmanufacturing/volumetric/2022/01

Implicit http://schemas.3mf.io/3dmanufacturing/implicit/2023/12

References

[1] Pasko, Alexander, et al. "Function representation in geometric modeling: concepts, implementation and applications." The visual computer 11.8 (1995): 429-446.

[2] Wyvill, Brian, Andrew Guy, and Eric Galin. "Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system." Computer Graphics Forum. Vol. 18. No. 2. Oxford, UK and Boston, USA: Blackwell Publishers Ltd, 1999.

See also [the standard 3MF References](https://github.com/3MFConsortium/spec_resources/blob/master/references.md).

Copyright 3MF Consortium 2025.