



3D XML User's Guide



**Version 4.3 Edition 0
March 2009**

This documentation is governed by the terms and conditions of Dassault Systèmes of America license. This documentation is, and shall remain, the exclusive property of Dassault Systèmes. The information in this documentation is subject to change without notice and should not be considered as a commitment from Dassault Systèmes and any of its affiliates. The documentation is provided "as is" and Dassault Systèmes and its affiliates do not assume any responsibility for any reason whatsoever including, but not limited to, errors or omissions that may appear in the documentation.

W3C® is a trademark (registered in numerous countries) of the World Wide Web Consortium; marks of W3C are registered and held by its host institutions MIT, ERCIM, and Keio.

Table of Contents

| | |
|---|------------|
| PREFACE | VII |
| Who Should Read this Guide? | vii |
| How to Use this Guide | vii |
| Related Documentation | vii |
| 1. 3D XML OVERVIEW | 9 |
| 1.1 Why 3D XML? | 9 |
| 1.1.1 Easy Adoption..... | 9 |
| 1.1.2 Extensibility | 9 |
| 1.1.3 Low-Volume and Quick Loading..... | 9 |
| 1.2 3D XML Concepts | 10 |
| 1.2.1 References and Instances | 10 |
| 1.2.2 Multi-Representation Concept | 10 |
| 1.2.3 Product Structure..... | 10 |
| 1.2.4 Identifying Occurrences | 11 |
| 1.2.5 Default View | 12 |
| 1.2.6 Extensions..... | 12 |
| 1.2.8 Coordinates System | 18 |
| 1.2.9 Positioning Matrix..... | 18 |
| 2. GETTING STARTED | 19 |
| 2.1 Your First 3D XML Document | 19 |
| 2.1.1 Model Description | 19 |
| 2.1.2 Overall Structure of a Document | 20 |
| 2.2 Document Header | 21 |
| 2.3 Creating and Organizing Objects in an Assembly | 21 |
| 2.3.1 A First Glance into the Product Structure Container | 21 |
| 2.3.2 Representing Objects | 23 |
| 2.4 Recommendations for Creating a Document from Scratch | 24 |
| 3. PRODUCT STRUCTURE | 25 |
| 3.1 References and Instances..... | 25 |
| 3.2 Representations | 26 |
| 4. GRAPHIC PROPERTIES | 28 |
| 4.1 Overview..... | 28 |
| 4.2 Graphic Attributes | 29 |
| 4.2.1 Surface Attributes | 29 |
| 4.2.2 Line Attributes | 30 |
| 4.2.3 Point Attributes | 30 |
| 4.2.4 General Attributes | 31 |

| | |
|--|-----------|
| 4.3 Using Graphic Properties | 32 |
| 5. MATERIALS | 34 |
| 5.1 Material DAG | 34 |
| 5.2 Rendering Materials..... | 36 |
| 5.2.1 Basic materials: | 37 |
| 5.2.2 Textured materials:..... | 40 |
| 5.3 Texture Mapping | 44 |
| 5.3.1 Mapping Types | 44 |
| 5.3.2 Mapping Operators..... | 44 |
| 5.3.3 Texture transformation..... | 46 |
| 5.3.4 Additional information for specific texture behavior | 46 |
| 5.4 Material usage | 48 |
| 5.4.1 Material on Product Structure..... | 48 |
| 5.4.2 Material on a mesh (geometric representation)..... | 50 |
| 5.4.3 How to Apply Textured Materials onto a Surface | 51 |
| 6. IMAGES | 52 |
| 7. VIEWPOINTS | 54 |
| 7.1 2D Viewpoints..... | 54 |
| 7.2 3D Viewpoints..... | 54 |
| 7.2.1 Projection Viewpoint..... | 54 |
| 7.2.2 Perspective Viewpoint..... | 55 |
| 7.2.3 Parallel Viewpoint..... | 56 |
| 7.2.4 Customized 3D Viewpoint | 57 |
| 8. MESH..... | 59 |
| 8.1 Concepts..... | 59 |
| 8.2 Vertex Buffer | 60 |
| 8.3 Polygonal Representation | 61 |
| 8.3.1 Graphic Attributes in a Polygonal Representation | 61 |
| 8.3.2 Primitive Sets | 61 |
| 8.4 3D XML Graphic Primitives..... | 62 |
| 8.4.1 The Face Primitive..... | 62 |
| 8.4.3 The Polyline Primitive | 63 |
| 8.5 Level of Detail (LOD) | 64 |
| 8.5.1 LOD Mechanism..... | 64 |
| 8.5.2 LOD Accuracy | 64 |
| 8.6 Example of a Polygonal Representation with a Texture..... | 65 |
| 8.7 Example of Multiple Textures Applications | 65 |
| 8.8 Example of a Polygonal Representation with LOD | 67 |
| 9. PPR | 69 |
| 9.1 Process | 70 |
| 9.2 Resource..... | 72 |
| 9.3 PPR context revisited | 75 |
| 9.4 Manikin | 76 |
| 10. 3DXML ARCHIVE | 78 |

| | |
|--|-----------|
| 10.1 Packaging..... | 78 |
| 10.2 Specifying Links | 80 |
| 10.2.1 Local Links..... | 80 |
| 10.2.2 External Links | 80 |
| 11. VERSIONING & EXTENSIBILITY..... | 81 |
| TASK INDEX..... | 83 |
| APPENDIX | 85 |
| Complete Reference/Instance Graph of the Quad Sample | 85 |
| GLOSSARY | 87 |

Preface

This guide provides you with the essential information for understanding and using the 3D XML language. 3D XML is designed especially for 3D data. It allows developers to create their own documents by using a language based on XML (eXtensible Mark-up Language), a specification developed by the World Wide Web Consortium (W3C). The purpose of this guide is to help you understand and create 3D XML documents.

Who Should Read this Guide?

This guide is intended for developers writing 3D data in 3D XML. To get the most out of this guide, you must have:

- ✚ a prerequisite knowledge of XML. If need be, refer to the <http://www.w3.org> site for information on XML
- ✚ some knowledge of the technologies and concepts used in 3D applications (visualization, mesh, etc).

How to Use this Guide

This guide is intended to be an introduction to the 3D XML language as well as an intermediate-level tutorial.

If you are a beginner, it is recommended to start with the "3D XML Overview" section, and download the schema and the *Reference Documentation* published on the web-site (<http://www.3ds.com/3dxml>). Then, to get up and running, read the "Getting Started" section.

To speed up the search for information, a [Task Index](#) has been provided.

Related Documentation

- ✚ The *Reference Documentation* gives a comprehensive list of all the 3D XML tags together with their attributes and associated documentation. This documentation assumes that you are familiar with the basics of 3D XML. If you are just starting with 3D XML and have no preliminary knowledge of any similar language, you may be overwhelmed by the sheer mass of information provided in the *Reference Documentation*. If so, read the *3D XML User's Guide* first.

1. 3D XML Overview

The 3D XML format is used across all of Dassault Systèmes brands – CATIA, DELMIA, ENOVIA, SIMULIA, 3DVIA, SolidWorks and Virtools. 3D XML format specification is part of “3D For All” initiative for bringing 3D to the masses. Dassault Systemes currently offers a free 3D XML Player for use with the 3DXML format as well as tools to capture 3D from other sources (3D PrintScreen). The offering will continue to grow.

1.1 Why 3D XML?

3D XML is a universal lightweight XML-based format for quick and easy sharing of 3D data. With 3D XML, 3D information can be easily incorporated into technical documentation, maintenance manuals, marketing brochures, websites, email communications and many other everyday uses. This re-use of 3D information broadens the base of 3D users.

1.1.1 Easy Adoption

3D XML is self-describing. 3D XML data can be read or written using standard tools and can be easily interpreted by applications. The broad adoption of 3D XML reduces the cost of re-using 3D data everywhere.

1.1.2 Extensibility

The 3D XML format allows extension of its instance/reference model. A pre-defined set of tags are provided. These tags can be specialized to add user-data to the 3D XML.

1.1.3 Low-Volume and Quick Loading

Used in conjunction with solutions for compressing highly complex data, 3D XML enables users to capture and share live 3D data quickly and easily in a lightweight format with file sizes up to 95 percent smaller than those of existing formats. 3D XML files can be loaded and transferred rapidly through the Web and used in the context of a collaborative system.

1.2 3D XML Concepts

3D XML relies on the principles presented below and detailed in the various sections of this guide.

1.2.1 References and Instances

3D XML supports the reuse of resources through an reference/instance mechanism:

- ✚ A Reference is a standardized object meant to be reused.
- ✚ An Instance is an instantiation of a reference. An instance carries information about its relative positioning. This information is specified by a positioning matrix. An instance is always aggregated to a reference.

References and instances are the key entities of the 3D XML Product Structure. References and instances are used to represent any assembly or product as a Directed Acyclic Graph. The Directed Acyclic Graph provides an efficient way to describe a product structure by avoiding duplication of reused references.

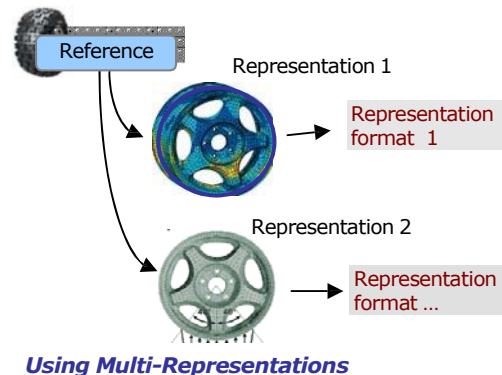
1.2.2 Multi-Representation Concept

A reference does not contain any geometric data. The way a reference appears is defined by one or more representations.

Representations can be made up of 3D data or text data. They are XML entities containing links to geometry containers. Several representations, of various types, can be associated with a reference. This capability can be used to model the applicative representation of a reference.

For example, you can represent a wheel according to its temperature characteristics or according to the pressure levels around the wheel. Another example: suppose your field is hydraulics, you can use one representation to describe the circuit diagram with its symbols, and another representation to describe mechanically the hydropumps and the pipes. Different representations can use different formats, if needed.

A reference is not necessarily associated with a representation. References that aggregate instances do not need representations. They use the representations of their aggregated instances.

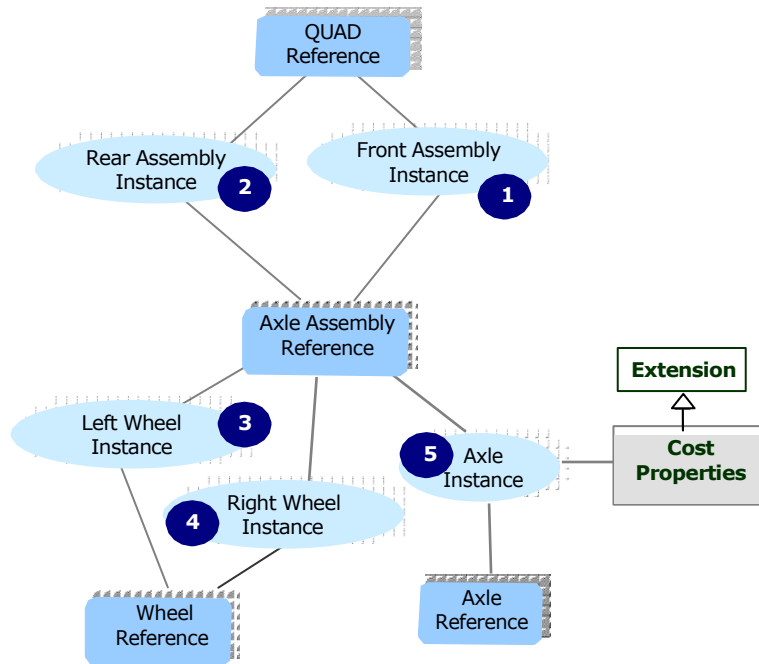


1.2.3 Product Structure

The Product Structure is the entity which aggregates the instances, the references and their representations. A Directed Acyclic Graph (DAG) or reference/instance graph describes the arrangement of entities within the Product Structure.

1.2.4 Identifying Occurrences

The Directed Acyclic Graph does not show explicitly all the objects making up an assembly. This reference/instance mechanism is a good way to describe a Product Structure without duplicating data, but it does not provide you with a complete tree structure. For example, the entities in a Directed Acyclic Graph can describe that an Axle assembly has 2 wheels and that a car has 2 Axle assemblies; but it does not allow to specifically identifying the Right wheel on the front Axle assembly. This is done through the use of instance path.



Example of a reference/instance graph & Extension

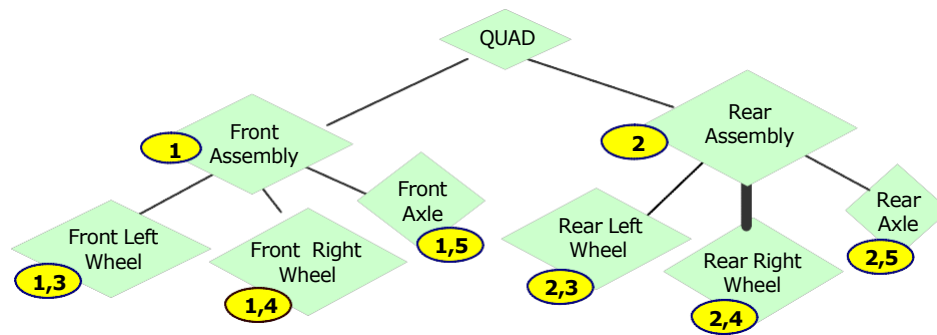
A Means to Access Each Object in a Model

Every reused reference appears only once in the instance/reference graph. In the above example, a wheel is used four times, but only appears once. In order to arrive at all the four usages of the wheel references, one has to traverse the above graph from the root to the wheel through the different available paths. The output of such a traversal is an unfolded tree. A node in the unfolded tree is called an *occurrence*.

Analogous to how an Extension is used to assign additional attributes to specific references/instances, an Occurrence Property is used to assign attributes to an occurrence. In order to assign an occurrence property to an occurrence, you must use an instance path to identify the specific occurrence (for example, the Front Left Wheel).

In the following depiction of the instance/reference model, each instance has a unique identifier. By using a list of these identifiers, it is possible to uniquely and explicitly identify each occurrence. For example, the Front Left Wheel is identified using 2 instance ids (1,3) – one for the Front Assembly (1) and the other for Left Wheel (3)

The following is a pictorial view of the occurrences in the Quad sample along with the *instance path* for each occurrence. Note that the deeper the assembly structure, longer is the path to uniquely identify each occurrence of the leaf nodes.



The Occurrence Tree with instance Paths

1.2.5 Default View

The default view, defines the default display of the model. In a default view, the user can indicate the default positions, color and visibility for each occurrence in the assembly. This information can be used by the viewing application for initial display of the model. The Default view reuses the concept of occurrence property and instance path explained above.

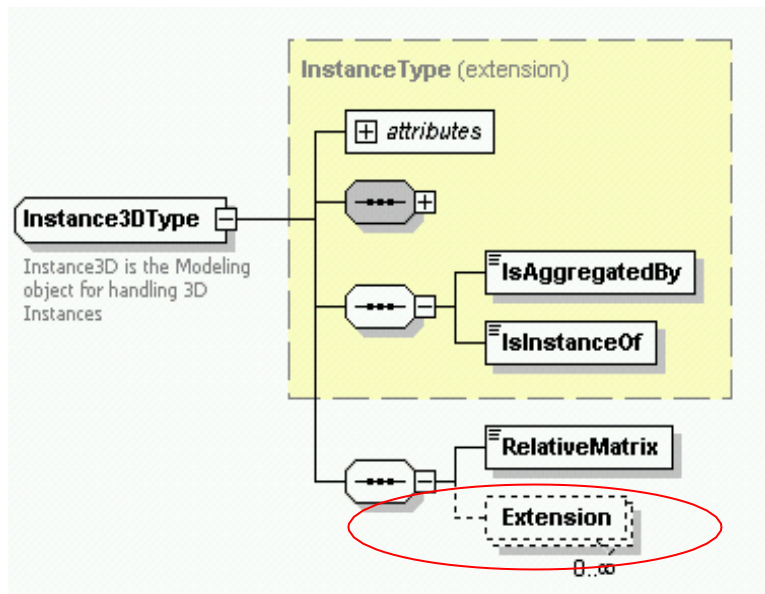
Default View Properties

A Default View property is a 3D XML element with which one can assign graphic properties to an occurrence. This property is used in Default View in order to build a initial view for display in the viewing application. Default View properties can include graphic properties, transformation matrices and visibility.

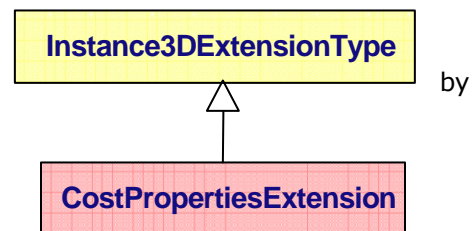
1.2.6 Extensions

If you intend to extend/enrich the published 3D XML specification for your use, this is done through the use of Extensions. The extension mechanism allows users to enrich the existing entities adding new attributes or sub-elements to a given entity. You can use this mechanism to attach any kind of data to references or instances. In the above example, the Cost Properties are added to the Axle Instance using a specialization of the `<Extension>`.

This is accomplished using an `<Extension>`. `<Reference3D>`, `<Instance3D>`, `<ReferenceRep>` and `<InstanceRep>` tags can be extended by specializing the extensions `<Reference3DExtensionType>`, `<Instance3DExtensionType>`, `<ReferenceRepExtensionType>`, `<InstanceRepExtensionType>` respectively. The following schema extract shows how Extensions are enabled for `<Instance3DType>`



For example, in order to add Cost information to an Instance3D, the user can extend the 3D XML schema specializing the `<Instance3DExtensionType>` and then instantiating this specialized extension into the `<Instance3D>` tag as shown below:



```
<xs:complexType name="CostPropertiesExtension" abstract="false">
  <xs:complexContent>
    <xs:extension base="Instance3DExtensionType">
      <xs:attribute name="Cost" type="xs:double" use="required"/>
      <xs:attribute name="Currency" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```

<Instance3D xsi:type="Instance3DType" id="29" name="Axle_Instance">
    .....
    <Extension xsi:type="CostPropertiesExtension">
        <Cost>60</Cost>
        <Currency>Euros</Currency>
    </Extension>
</ Instance3D >

```

1.2.7 Customization

From the version 4.2 of 3D XML schema you are able to customize 3D XML schema. It is the capability to specialized existing specification by creating new types based on existing one and by adding new attributes.

The specifications (ordered by containers) which support customization are:

ProductStructure

```

<xs:complexType name="Reference3DType" abstract="false" final="restriction">
<xs:complexType name="Instance3DType" abstract="false" final="restriction">
<xs:complexType name="ReferenceRepType" abstract="false" final="restriction">
<xs:complexType name="InstanceRepType" abstract="false" final="restriction">

```

Process

```

<xs:complexType name="DELFmiFunctionReferenceType" abstract="false" final="restriction">

```

CATMaterialRef

```

<xs:complexType name="CATMatReferenceType" abstract="false" final="restriction">
<xs:complexType name="MaterialDomainType" abstract="false" final="restriction">
<xs:complexType name="MaterialDomainInstanceType" abstract="false" final="restriction">

```

3D XML schema add-on, how to?

If you intend to create your own customization you have to create first an add-on of 3D XML XSD to describe your new customized type.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://www.3ds.com/xsd/3DXML/PRODUCT/DummyCustomization" xmlns:xs3d="http://www.3ds.com/xsd/3DXML" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.3ds.com/xsd/3DXML/PRODUCT/DummyCustomization" elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
  <xs:import namespace="http://www.3ds.com/xsd/3DXML" schemaLocation="3DXML.xsd"/>
  <xs:complexType name="CustoOfReference3D">
    <xs:complexContent>
      <xs:extension base="xs3d:Reference3DType">
        <xs:sequence>
          <xs:element name="InternalCodeProduct" type="xs:string" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="CustoOfRepReference">
    <xs:complexContent>
      <xs:extension base="xs3d:ReferenceRepType">
        <xs:sequence>
          <xs:element name="OrigineOfShape" type="xs:string" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>

```

<xs:schema> Element:

You have to declare as attributes of this element:

- the new target Namespace for this Customisation with the following pattern:
 - for specialization of types declared in ProductStructure Container
http://www.3ds.com/xsd/3DXML/**PRODUCT**/<<CustomizationName>
 - for specialization of types declared in Process Container
http://www.3ds.com/xsd/3DXML/**PROCESS**/<<CustomizationName>
 - for specialization of types declared in CATMaterialRef Container
http://www.3ds.com/xsd/3DXML/**CATMaterialRef**/<<CustomizationName>
- A prefix for this customization
xmlns:tns="http://www.3ds.com/xsd/3DXML/PRODUCT/<CustomizationName>"
- Warning:** The attribut version="1.0" is the version dedicated to the schema of how to declare an xsd for 3D XML customization. (this attribute is mandatory)

<xs:import> Element:

Following tag is mandatory

```
<xs:import namespace="http://www.3ds.com/xsd/3DXML" schemaLocation="3DXML.xsd"/>
```

<xs:ComplexType> Elements:

You can declare 0 or **1** new complex type for each existing complex type defined in 3D XML xsd (see above the available complexType which can be specialized)

- Specialization must be a complexType which is an extension⁽²⁾ with a sequence of 1..n element(s)⁽⁵⁾
- Specialization must have a name⁽¹⁾.

- The extension is based on one complexType declared in 3D XML xsd⁽³⁾ (see above the available complexType which can be specialized).
- The sequence⁽⁴⁾ tag must have minOccurs="1" and maxOccurs="1" or neither minOccurs nor maxOccurs (same signification)

```
<xs:complexType name(1)="CustoOfReference3D">
<xs:extension(2) base="xs3d:Reference3DType"(3)>
<xs:sequence(4)>
<xs:element(5) ... />
<xs:element(5) ... />
...
```

Each `<xs:element>` must have a name, a type (simple type only) and must have `maxOccurs="1"` or no attribute `maxOccurs` (Same signification)

```
<xs:element name="InternalCodeProduct" type="xs:string" minOccurs="0"/>
```

Warning: Only the above declaration is supported!

What about instance of 3D XML?

- Without customization

```
<?xml version="1.0" encoding="utf-8" ?>
<Model_3dxml xmlns="http://www.3ds.com/xsd/3DXML" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductStructure root="1">
    <Reference3D xsi:type="Reference3DType" id="1">
      <PLM_ExternalID>Quad</PLM_ExternalID>
    </Reference3D>
    <ReferenceRep xsi:type="ReferenceRepType" id="4" format="UVR" version="1.0" associatedFile="urn:3DXML:Chassis.3DRep">
      <PLM_ExternalID>ChassisRepr</PLM_ExternalID>
      <V_discipline>Design</V_discipline>
      <V_nature>1</V_nature>
    </ReferenceRep>
  </ProductStructure>
</Model_3dxml>
```

- With customization

```
<?xml version="1.0" encoding="utf-8" ?>
<Model_3dxml xmlns="http://www.3ds.com/xsd/3DXML" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.3ds.com/xsd/3DXML/PRODUCT/DummyCustomization DummyCustomization.xsd"
  xmlns:custo="http://www.3ds.com/xsd/3DXML/PRODUCT/DummyCustomization">
  <ProductStructure root="1">
    <Reference3D xsi:type="custo:CustoOfReference3D" id="1">
      <PLM_ExternalID>Quad</PLM_ExternalID>
      <custo:InternalCodeProduct>Prod#280475</custo:InternalCodeProduct>
    </Reference3D>
    <ReferenceRep xsi:type="custo:CustoOfRepReference" id="4" format="UVR" version="1.0" associatedFile="urn:3DXML:Chassis.3DRep">
      <PLM_ExternalID>ChassisRepr</PLM_ExternalID>
      <V_discipline>Design</V_discipline>
      <V_nature>1</V_nature>
      <custo:OrigineOfShape>Supplier AAA Shape</custo:OrigineOfShape>
    </ReferenceRep>
  </ProductStructure>
</Model_3dxml>
```

- What are the differences?

<Model_3dxml> Element:

New namespace⁽¹⁾ is added with a dedicated prefix⁽²⁾
Schema location is added (link between namespace and XSD)⁽³⁾

These information are mandatory but can be write at different place in 3D XML instance
(have a look on W3C documentation)

```
<Model_3dxml xmlns=http://www.3ds.com/xsd/3DXML  
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  
xsi:schemaLocation="http://www.3ds.com/xsd/3DXML/PRODUCT/DummyCustomization DummyCustomization.xsd"(3)  
xmlns:custo(2)="http://www.3ds.com/xsd/3DXML/PRODUCT/DummyCustomization"(1)>
```

<Reference3D> Element:

This element has new type⁽⁴⁾ with his dedicated prefix⁽²⁾.

```
<Reference3D xsi:type="custo(2):CustoOfReference3D"(4) id="1">
```

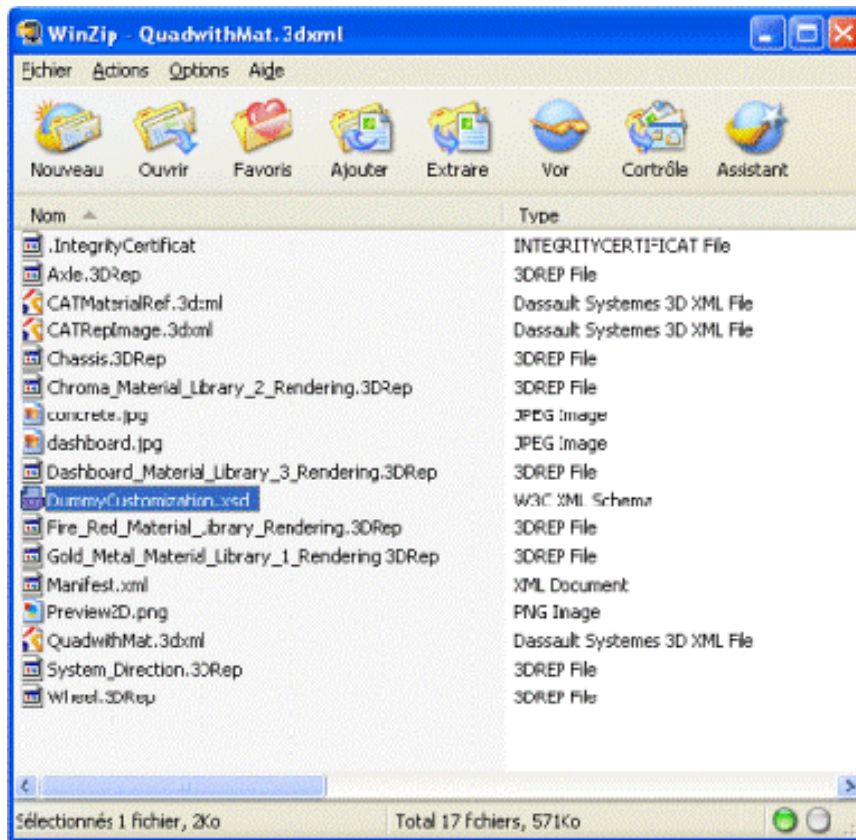
Element added with customization:

Thanks to xsd add-on and customized type of <Reference3D>⁽⁴⁾ you are able to add sub
element define in xsd add-on. Do not forget the prefix⁽²⁾.

```
<custo(2):InternalCodeProduct>Prd#280475</custo(2):InternalCodeProduct>
```

What about 3D XML archive?

3D XML schema add-on(s) must be included in 3D XML archive. Without it, the customization
is not readable and the 3D XML instance is not compliant with the 3D XML schema.



1.2.8 Coordinates System

3DXML uses a counter-clockwise coordinate system.

1.2.9 Positioning Matrix

To make up a consistent structure, the instances have to be positioned within the reference they are aggregated to. This is the role of the relative matrix which defines the relative coordinates and the rotation of a child object with respect to its parent. The relative matrix is a 3x4 matrix which includes a 3x1 transformation matrix and a 3x3 rotation matrix.

The positioning matrix is specified by using the `<RelativeMatrix>` sub-element of the `<Instance3D>` element.

2. Getting Started

This section is intended to get you started with 3D XML by using a sample document. This document illustrates the principles you need to understand before going any further in a 3D XML application. The comprehensive list of 3D XML tags is given in the *Reference Documentation*. In this section, you will learn how to create the objects making up a Product Structure and how to modify some graphic properties. Recommendations for creating a document from scratch are also given. To work with this document, you must install the 3DXML Player, download the Quad.3dxml sample and the schema; all of which are delivered on the 3D XML web-site (<http://www.3ds.com/3dxml>).

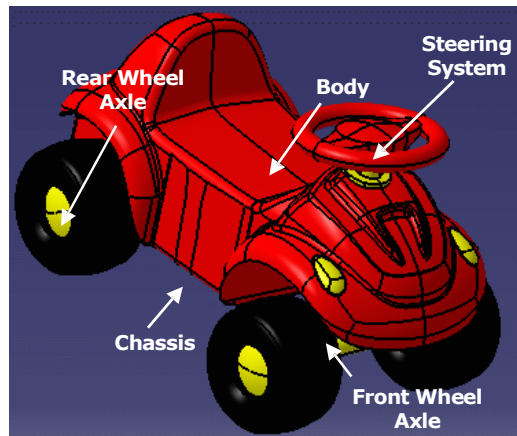
2.1 Your First 3D XML Document

The Quad.3dxml is the 3D XML description of the basic vehicle shown on the right. You can visualize this model in the 3DXML Player by double-clicking on the 3DXML file after you installed the 3DXML Player.

If you want to examine the XML content of this description, you will have to first extract the contents Quad.3DXML using any ZIP utility into directory. Then you can open the Quad.3DXML file using a text (or XML) viewer. complete description of the 3DXML document structure, please refer to chapter 10.

2.1.1 Model Description

To make things clearer and help you understand the 3D XML basics, this vehicle is just made up of the two wheel axles, a chassis, a body and a steering system.



have

a

For a

2.1.2 Overall Structure of a Document

A 3D XML document consists of a hierarchy of objects.

The <Model_3dxml> Tag

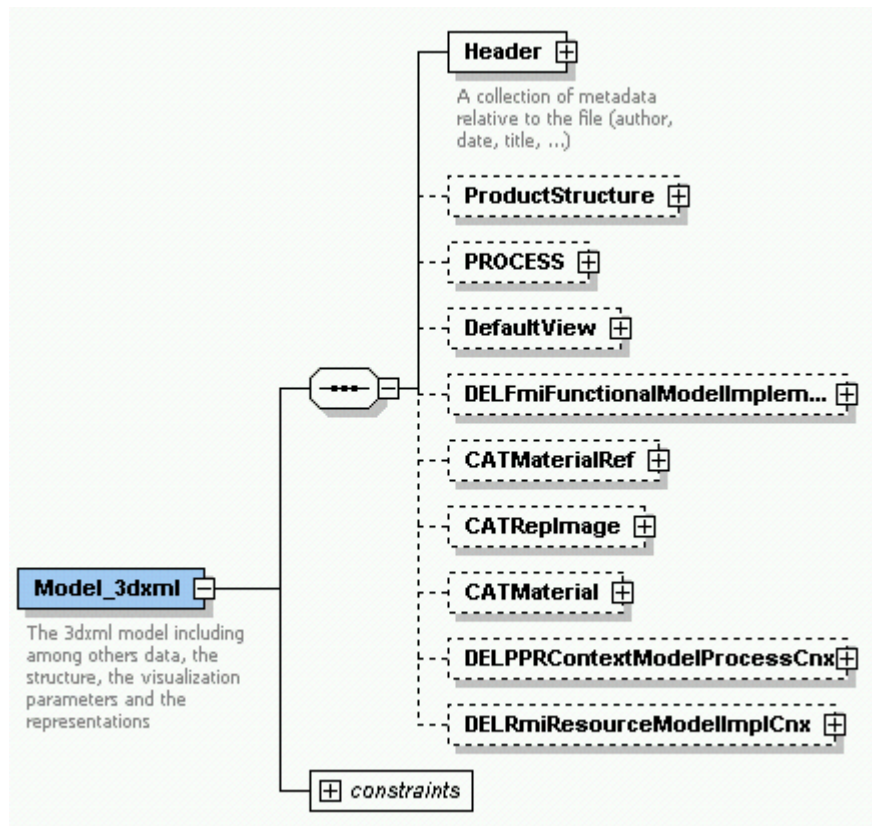
This root element federates all the containers to be included in the document. In a 3D XML document, there must be one and only one <Model_3dxml> element.

```
<?xml version="1.0" encoding="utf-8"?>
<Model_3dxml xmlns="http://www.3ds.com/xsd/3DXML"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
```

3D XML Containers

The sub-elements of a <Model_3dxml> element depend on the kind of data you want to store in your document. In a model, data is grouped into containers. A container is a high-level 3D XML element intended to wrap related data. The data in one container can refer to data specified in other containers. Any entity intended to be pointed to by another is assigned an identifier which must be unique within a given container.

Not all available containers are present in the accompanying sample. 3D XML defines several types of containers but not all are mandatory. Following is the list of containers which can exist in a model. We will look at these containers in detail in following chapters.



2.2 Document Header

The `<Header>` element describes the various properties of the document, including its title, schema version or author. The `<SchemaVersion>` is the only mandatory sub-element. However, it is recommended to include other sub-elements as it helps readability and document maintenance. The date as an "xs:date" data type is written in the following form "YYYY-MM-DD".

```
<Header>
  <SchemaVersion>4.0</SchemaVersion>
  <Title> Quad 3DXML File</Title>
  <Author>Someone</Author>
  <Created>2007-03-29</Created>
</Header>
```

The `<Header>` element describes the various properties of the document, including its title, schema version or author. The `<SchemaVersion>` is the only mandatory sub-element. This element is used to determine the compatibility of a given 3DXML document with the viewing application. The `<SchemaVersion>` follows a "major version.minor version" pattern. It is recommended, however, to include as much information as possible to improve readability and document maintenance. The `<Created>` and `<Modified>` elements ("xs:date" data type) are written using "YYYY-MM-DD" format.

2.3 Creating and Organizing Objects in an Assembly

All the data related to the creation of objects as well as their arrangement within a global structure is grouped into the Product Structure container. The Product Structure container is the most important part of a 3D XML document because it defines the way objects are organized to make up an assembly.

2.3.1 A First Glance into the Product Structure Container

If you take a look at the data within the `<ProductStructure>` element, you find a number of `<Reference3D>` and `<Instance3D>` elements.

Instances and References

What are these elements for? In the vehicle assembly, there are objects which are repeated. In the 3D XML model, repeating the geometry would amount to unnecessary data duplication. To avoid data duplication, 3D XML uses a reference/instance mechanism. The object geometry is defined only once in a reference. Every reuse of this reference therefore, reuses the geometry associated with that reference. A vehicle has four wheels, but there is just one reference which defines the wheel geometry.

```

<ProductStructure root="1">
.....
    <!--The Axle Assembly Reference -->
    <Reference3D xsi:type="Reference3DType" id="11" name="Axle Assembly"/>
    <!--The Wheel Reference -->
    <Reference3D xsi:type="Reference3DType" id="13" name="Wheel"/>

    <!--Details about the representation for the Wheel – only described once -->
    <ReferenceRep xsi:type="ReferenceRepType" id="14" name="Wheel_ ReferenceRep"
        format="UVR" version="1.0" associatedFile="urn:3DXML:Wheel.3DRep"/>
    <InstanceRep xsi:type="InstanceRepType" id="15" name="Wheel_InstanceRep">
        <IsAggregatedBy>13</IsAggregatedBy>
        <IsInstanceOf>14</IsInstanceOf>
    </InstanceRep>

    <!--2 instances of Wheel belonging to the Axle assembly -->
    <Instance3D xsi:type="Instance3DType" id="12" name="Wheel.1">
        <IsAggregatedBy>11</IsAggregatedBy>
        <IsInstanceOf>13</IsInstanceOf>
        <RelativeMatrix>1 0 0 0 1 0 0 0 1 0 1 0 </RelativeMatrix>
    </Instance3D>
    <Instance3D xsi:type="Instance3DType" id="16" name="Wheel.2">
        <IsAggregatedBy>11</IsAggregatedBy>
        <IsInstanceOf>13</IsInstanceOf>
        <RelativeMatrix>1 0 0 0 1 0 0 0 1 1 1 1 </RelativeMatrix>
    </Instance3D>
.....
</ProductStructure>

```

Now, if you look for the instances created from this reference, you find just two of them (with id 12 and 16). This is because the wheel instances are themselves aggregated to the Assembly reference which is instantiated twice. In your product, you actually have four wheel instances, but they are not all created directly from the wheel reference. Instances and references are described by the `<Instance3D>` and `<Reference3D>` objects respectively. Both objects are identified by an `id` (mandatory) and a `name` (optional).

The `<IsInstanceOf>` sub-element of an `<Instance3D>` element identifies its reference. In the example above, the resource identifier "urn:3DXML:Wheel.3DRep" refers to a reference to be located within the 3D XML archive. The 3DXML archive structure is explained in Chapter 10.

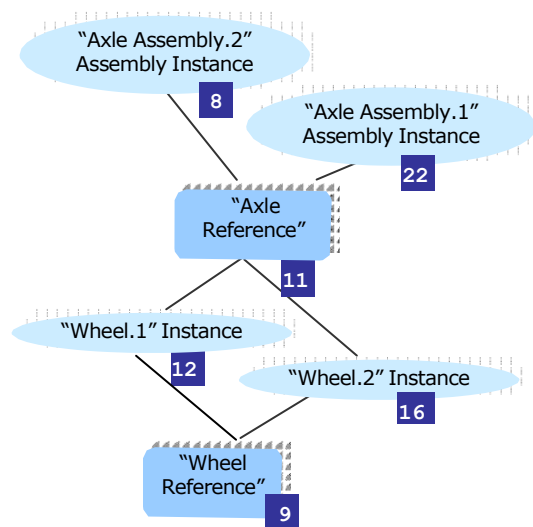
Aggregation Mechanism

The aggregation mechanism enables you to create references from instances which are in turn created from lower-level references.

The graph shown on the right is an extracted view of the complete vehicle reference/instance graph. It shows how the two axle assemblies (front and rear) are created from the wheel reference. Instances 12 and 16 are created from reference 9 and are aggregated to reference 11 which describes an axle assembly. Then reference 11 is instantiated twice.

See the [Appendix](#) for the complete reference/instance graph.

The `<IsAggregatedBy>` sub-element of the `<Instance3D>` element defines the reference to which the instance is aggregated.



Positioning Matrix

To make up a consistent structure, the instances have to be positioned within the reference they are aggregated to. This is the role of the relative matrix which defines the relative coordinates and the rotation of a child object with respect to its parent. The relative matrix is a 3x4 matrix which includes a 3x1 transformation matrix and a 3X3 rotation matrix.

The positioning matrix is specified by using the `<RelativeMatrix>` sub-element of the `<Instance3D>` element.

2.3.2 Representing Objects

References are 3D XML elements which are defined by just two attributes, the `id` and the `name`. You may wonder where the geometry of these objects is stored and how this geometry and the references are linked.

In 3D XML, the geometry is defined using representations. A representation contains a pointer to the data. The `<ReferenceRep>` element allows you to specify the reference owning the representation (the value of the `owner` attribute), and the pointer to the geometric data (the value of the `associatedFile` attribute). A `<Representation>` has an `id` which is referred to by the value of the `associatedFile` attribute. 3DXML allows the reuse of a Representation through the instance-reference mechanism explained above.

```

<!--Details about the representation for the Wheel – only described once -->
<ReferenceRep xsi:type="ReferenceRepType" id="14" name="Wheel_ReferenceRep" format="UVR"
version="1.0" associatedFile="urn:3DXML:Wheel.3DRep"/>
<InstanceRep xsi:type="InstanceRepType" id="15" name="Wheel_InstanceRep">
  <IsAggregatedBy>13</IsAggregatedBy>
  <IsInstanceOf>14</IsInstanceOf>
</InstanceRep>
  
```

The `<ReferenceRep>` element points to geometric data located within the current document. 3D XML supports two formats for representations. For more information, see [section 3.2](#).

2.4 Recommendations for Creating a Document from Scratch

1. Prior to creating a document, you must define your Product Structure. To do this, review the assembly to be described and determine what objects are repeated. Breaking down your assembly into instances and references is the first step.
2. Create a first Product Structure with all the objects to be instantiated only once (like the Chassis for example). These objects can be easily described in 3D XML. For each object, create the reference along with the representation and the related instance.
3. Check this initial document with respect to the 3D XML syntax. Then display it in the 3D XML Player. At this stage, you have a simple Product Structure in which all instances are directly aggregated to the root reference.
4. At this stage of your design, if you display your document in the 3D XML Player, you get a bare Product Structure. Now you can add representations and graphic properties.

3. Product Structure

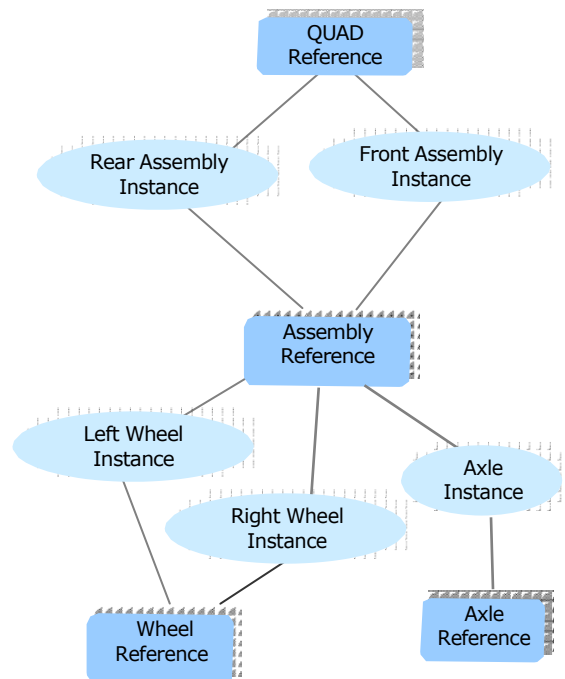
A 3D model consists of a hierarchy of objects. These objects are gathered into a container referred to as the Product Structure. To describe the Product Structure, 3D XML defines a lightweight Directed Acyclic Graph (DAG) that makes intensive use of references and instances.

3.1 References and Instances

A reference is a standardized object intended to be reused in several different products or within one given product. A reference is only specified once, but can be instantiated several times. Using references is a way to minimize data duplication and reduce the size of generated documents.

The reference concept is used to specify any objects located in the Product Structure. Assuming that the left and right wheels have the same geometry, there is no need to duplicate the wheel geometry. The wheel geometry is defined in the reference. The right and left wheels are created by instantiating the reference and assigning a specific position to the created objects within the Product Structure.

- ✚ An instance is an instantiation of a reference to be used somewhere in a specific product. From a single reference, you can create several 3D instances. Each created instance is to be aggregated to a single 3D reference. The right and left wheels are two instances created from the Front Wheel reference. They are both aggregated to the Front Axle Assembly reference. They cannot be aggregated to another reference.



An example of a reference/instance graph

How to Create References and Instances

You must use the `<Reference3D>` and `<Instance3D>` tags respectively. Both tags have a mandatory `id` attribute and an optional `name` attribute. `<Instance3D>` objects have children or sub-elements. These children are described in the *Reference Documentation*.

An `<Instance3D>` object can point to a `<Reference3D>` object defined in an external resource. In the example below, the instance and the reference are located in the same resource (file).

```
<Reference3D xsi:type="Reference3DType" id="2" name="Chassis"/>
<Instance3D xsi:type="Instance3DType" id="3" name="Chassis.1">
  <IsAggregatedBy>1</IsAggregatedBy>
  <IsInstanceOf>2</IsInstanceOf>
  <RelativeMatrix>1 0 0 0 1 0 0 0 1 -35.3964 202.211 0</RelativeMatrix>
</Instance3D>
```

In the example below, the instance and the reference are located in different resources (files).

```
<Instance3D xsi:type="Instance3DType" id="3" name="Chassis.1">
  <IsAggregatedBy>1</IsAggregatedBy>
  <IsInstanceOf>urn:3DXML:Reference:ext:Chassis.3dxml#1</IsInstanceOf>
  <RelativeMatrix>1 0 0 0 1 0 0 0 1 -35.3964 202.211 0</RelativeMatrix>
</Instance3D>
```

The URN value in the `<IsInstanceOf>` tag defines the external resource (Chassis.3dxml) along with a pointer (#1) which is located in this resource. In Chassis.3dxml, you must write the reference declaration like this:

```
<Reference3D xsi:type="Reference3DType" id="1" name="Chassis"/>
```

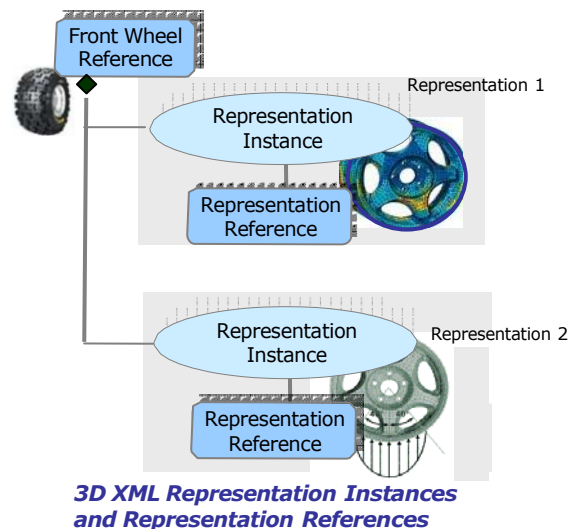
3.2 Representations

A reference can have multiple representations as explained in “1.2.2 Multi-Representation Concept” in [section 1.2.2](#).

The `<ReferenceRep>`/`<InstanceRep>` tags, are used to associate a representation with a reference.

How to Specify a Representation

You must use the `<ReferenceRep>` and `<InstanceRep>` tags as indicated in the following example. The `<IsAggregatedBy>` sub-element of the `<InstanceRep>` tag makes the link between the `<Reference3D>` and the representation. The `<ReferenceRep>` tag refers to the geometry using `associatedFile`. The value of the `associatedFile` attribute is an URN (Universal Resource Name) which is a predefined syntax with identifiers.



```

...
<Reference3D xsi:type="Reference3DType" id="13" name="Wheel"/>
<ReferenceRep xsi:type="ReferenceRepType" id="14" name="Wheel_ReferenceRep" format="UVR"
version="1.0" associatedFile="urn:3DXML:Wheel.3DRep"/>
<InstanceRep xsi:type="InstanceRepType" id="15" name="Wheel_InstanceRep">
  <IsAggregatedBy>13</IsAggregatedBy>
  <IsInstanceOf>14</IsInstanceOf>
</InstanceRep>
.....

```

You can attach several `<ReferenceRep>/<InstanceRep>` objects to the same reference. This is how multi-representation is supported in 3DXML. In the example above, you can declare other `<ReferenceRep>/<InstanceRep>` pairs and aggregate them all to `Reference3D`.

3D XML supports two major formats: the XML Tessellation and UVR as described below:

| FORMAT | TESSELLATED Format Version 1.2 | UVR Format Version 1.0 |
|--------------------|--|---|
| Description | Provides the representation of a part through a list of organized triangles (mesh) | Compressed Triangular mesh |
| Purpose | XML format - can be used for simple geometry exchange | Small file size, fast loading, low memory consumption |

The representation format and format type must be specified:

 in the Product Structure by using the Format attribute in `<ReferenceRep>` element

The `associatedData` attribute is used to point to the file within the 3DXML archive which contains the geometric representation. `<ReferenceRep>`, `<InstanceRep>` and `<Representation>` should always be declared in the same resource.

4. Graphic Properties

Graphic properties can be applied to highlight the different objects of a model, make more comprehensive the design of a Product Structure or provide a more attractive display of the model. With graphic properties, you can modify colors and manage the visibility and opacity.

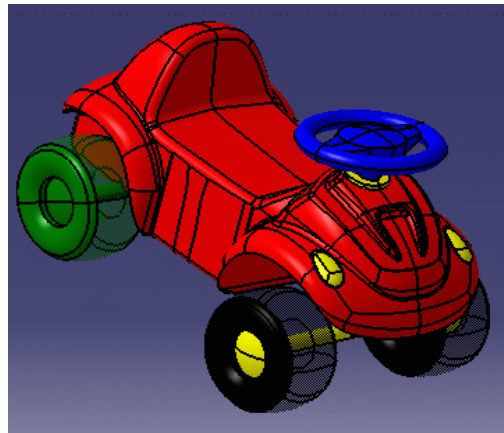
4.1 Overview

3D XML provides you with the graphic attributes usually encountered in graphic applications for surface, lines or points.

Graphic properties are not mandatory but all the objects of a model can be endowed with graphic properties.

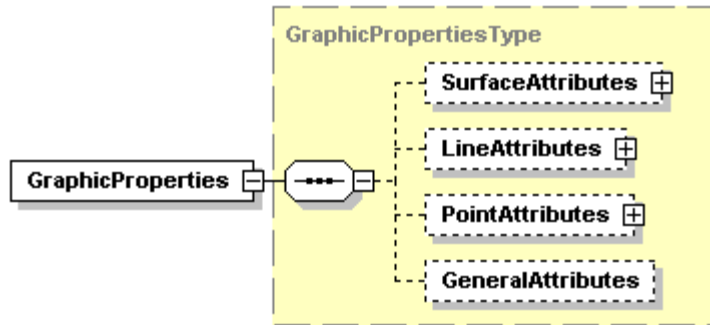
In 3D XML, there are two mechanisms to specify graphic properties:

1. The `<DefaultView>` which applies to occurrences (see section [1.2.4](#) and [1.2.5](#))
2. Graphic Attributes assigned within the XML Mesh to the geometric sub-elements.



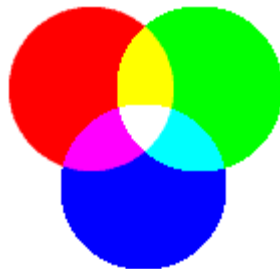
4.2 Graphic Attributes

Graphic attributes are divided into four categories: surface attributes, line attributes, general attributes and point attributes.



4.2.1 Surface Attributes

Surface attributes are basically colors. 3D XML supports RGBAColorType colors. The RGBA model is used to define a color by its four components: red, green, blue, and alpha. The RGBA color model is an additive color model in which Red, Green, and Blue light are combined in various ways to create other colors.



RGBA colors

The alpha channel is added to manage the transparency.

How to Specify Surface Attributes

Use the `<Color>` element as indicated below:









```
<SurfaceAttributes>
  <Color xsi:type="RGBAColorType" red="1" green="1" blue="1" alpha="1"/>
</SurfaceAttributes>
```

Red, green, blue and alpha channels are floating point value ranging from 0 to 1. For a fully opaque color, the alpha channel attribute should be set to 1. For a fully transparent color, you should specify 0.

4.2.2 Line Attributes

Line attributes control the color, thickness and styles of edges, wires and curves. 3D XML supports thicknesses from 0.1300mm to 2.600mm as well as the usual line types.

- A line color
 - Can be specified by a RGBA color.
- A line width
 - Defines the thickness of the line in millimeters.

| | |
|---|-----------|
|  | 0.1300 mm |
|  | 0.3500 mm |
|  | 0.7000 mm |
|  | 1.0000 mm |
|  | 1.4000 mm |
|  | 2.0000 mm |
|  | 2.3000 mm |
|  | 2.6000 mm |

- a line type
 - the following default line types are provided:

| | |
|---------------------|--|
| Solid |  |
| Dotted |  |
| Dashed |  |
| Dot-Dashed |  |
| Phantom |  |
| Small-Dotted |  |
| JIS Axis |  |

How to Specify Line Attributes

Use the `<LineAttributes>` as indicated below.

```
<GraphicProperties xsi:type="GraphicPropertiesType">
...
  <LineAttributes thickness="0.35" lineType="DASHED">
    <Color xsi:type="RGBAColorType" red="0" green="0" blue="0"/>
  </LineAttributes>
...
```

4.2.3 Point Attributes

Point Attributes are used to define graphic attributes of points. They include:

- A color
 - Can be specified by a RGBA color or an indexed one.
- A symbol type
 - The following default point symbols are provided:

| | |
|---|------------|
| × | CROSS |
| + | PLUS |
| ○ | CONCENTRIC |
| ◎ | COINCIDENT |
| ● | FULLCIRCLE |
| ■ | FULLSQUARE |
| ✳ | STAR |
| ▪ | DOT |
| . | SMALLDOT |

How to Specify Point Attributes

Use the `<PointAttributes>` tag as indicated below.

```
<GraphicProperties xsi:type="GraphicPropertiesType">
...
  <PointAttributes symbolType="STAR">
    <Color xsi:type="RGBAColorType" red="0" green="0" blue="0"/>
  </PointAttributes>
.....
</GraphicProperties>
```

4.2.4 General Attributes

With general attributes, you can control the way objects react upon selection and specify whether they are to be shown or not.

How to Specify General Attributes

Use the `<GeneralAttributes>` tag along with the `visible` and `selectable` attributes as shown below:

```
<GraphicProperties xsi:type="GraphicPropertiesType">
  <GeneralAttributes xsi:type="GeneralAttributesType" visible="false" selectable="true"/>
</GraphicProperties>
```

4.3 Using Graphic Properties

How to use Graphic Properties with Default View

Default view is used to assign colors to occurrences in product structure. Default view allows overriding the color, position and visibility of each occurrence. Surface attributes are basically colors. `<GraphicProperties>` is used in a `<DefaultView>`

to assign color, opacity and visibility to each occurrence. The following 3dxml extract demonstrates the usage.

```
<DefaultView>
  <Viewpoint xsi:type="ParallelViewpointType" visualizedHeight="2588.745866"
    targetDistance="9661.331055" nearPlaneDistance="8296.171875"
    farPlaneDistance="10898.97559">
    <Position>-5806.268555 4614.594727 6028.903809</Position>
    <Sight>0.6336148381 -0.4515056014 -0.628231585</Sight>
    <Right>-0.5478185415 -0.8352326155 0.0477630496</Right>
    <Up>0.5462847948 -0.3138935268 0.7765589356</Up>
  </Viewpoint>
  <DefaultViewProperty>
    <OccurrenceId>
      <id>urn:3DXML:Disassemble_Quad.3dxml#2</id>
      <id>urn:3DXML:Disassemble_Quad.3dxml#4</id>
      <id>urn:3DXML:Disassemble_Quad.3dxml#10</id>
      <id>urn:3DXML:Disassemble_Quad.3dxml#12</id>
    </OccurrenceId>
    <RelativePosition>1.000000000000005
      -5.89805981832114e-017
      .....
      0.001199999999996878
    </RelativePosition>
    <GraphicProperties xsi:type="GraphicPropertiesType">
      <SurfaceAttributes xsi:type="SurfaceAttributesType">
        <Color xsi:type="RGBAColortype" red="1"
          green="0" blue="0"/>
      </SurfaceAttributes>
    </GraphicProperties>
  </DefaultViewProperty>
  .....
</DefaultView>
```


How to use Graphic Properties with Mesh

Graphic properties can be applied directly inside the geometric representation described by Mesh (see Chapter 8 for more information about Mesh). In this case, it is possible to assign graphic properties either to the entire Representation (see extract below) or to individual Faces, Edges and graphic primitives.

```
.....
<Rep xsi:type="PolygonalRepType" accuracy="0.2">
  <!--graphic attributes of the polygonal rep-->
  <SurfaceAttributes>
    <Color xsi:type="RGBAColorType" red="0.6" green="0.6" blue="0.7"/>
  </SurfaceAttributes>
  <LineAttributes thickness="0.35" lineType="DASHED">
    <Color xsi:type="RGBAColorType" red="0" green="0" blue="0"/>
  </LineAttributes>
.....
</Rep>
```

5. Materials

Material property describes matter characteristics of a physical part with different applicative or physical domains.

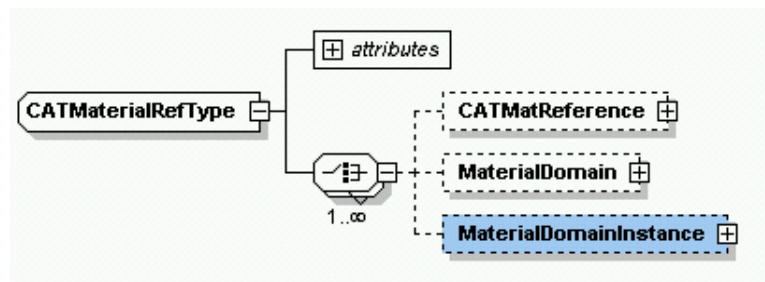
This version includes only the support for rendering domain which deals with material aspect visualization.

3D XML supports basic and textured rendering materials. Basic materials are those you can assign to an object so that it looks uniformly colored like plastic or rubber. You can modify the rendering of a basic material by specifying parameters such as the ambient, diffuse, specular lights or the reflectivity. Textured materials are irregular patterns that simulate a texture surface like marble or wood.



5.1 Material DAG

<CATMaterialRef> is the container for all material used in the 3DXML model. It collects the associated material references and their instances. Within the 3DXML archive, this container is located in a dedicated file named CATMaterialRef.3dxml. There can only be one **<CATMaterialRef>** in a model.



The CATMaterialRef.3dxml file contains:

- ¾ A specific element, named **<CATMatReference>**. This element represents the reference for each material used in the model. There can be any number of materials within **<CATMaterialRef>**. This element has following attributes:
 - id: 3DXML identifier
 - name: Descriptive name for the material.

A specific element, named **<MaterialDomain>** under **<CATMaterialRef>**. This element describes any specific resource such as an image file for the texture as well as the rendering properties.

Only one domain named "Rendering" is available and the purpose is for aspects visualization. This element has following attributes:

- id: 3DXML identifier (c.f. 3DXML documentation or related XSD)
- associatedFile: Specifies the embedded structure or an external link of the given rendering material rep.
- name: Descriptive name for the domain.
- format: Type of the domain is TECHREP.
- A [<PLMExternalID>](#) element
- A [<V_MatDomain>](#) element: type of the domain (Rendering)
- A [<PLM_Relation>](#) element: associate the domain to an image representation.

A specific element [<MaterialDomainInstance>](#) under [<CATMaterialRef>](#). It describes the usage of a Material Domain (as described in instance/reference concept). For a given model, Material can only be described with a single domain instance. Multiple domain instances per Material domain are not yet supported. [<MaterialDomainInstance>](#) has following attributes:

- id: 3DXML identifier (c.f. 3DXML documentation or related XSD)
- name: Descriptive name for the domain.
- format: Type of the domain is TECHREP.
- A [<PLMExternalID>](#) element
- A [<IsAggregatedBy>](#) element (pointing to a [<CATMatReference>](#) id)
- A [<InstanceOf>](#) element (pointing to a [<MaterialDomain>](#) id)

Here is a sample of material with a rendering domain used for Quad's wheels.

```
<Model_3dxml xmlns="http://www.3ds.com/xsd/3DXML" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <Header>
    <SchemaVersion>4.0</SchemaVersion>
    <Title>3DXml File</Title>
    <Author>MJJ</Author>
    <Generator>CATIA V5</Generator>
    <Created>2007-04-19</Created>
  </Header>
  <CATMaterialRef>
    <CATMatReference id="15" xsi:type="CATMatReferenceType" name="Rubber_15">
      <PLM_ExternalID>Rubber_15</PLM_ExternalID>
    </CATMatReference>
    <MaterialDomain version="1.0" id="17"
      associatedFile="urn:3DXML:Rubber_15_Rendering.3DRep"
      xsi:type="MaterialDomainType" format="TECHREP"
      name="Rubber_15_Rendering">
      <PLM_ExternalID>Rubber_15_Rendering</PLM_ExternalID>
      <PLMRelation>
        <C_Semantics>Reference</C_Semantics>
        <C_Role>CATMaterialReferenceToTextureLink</C_Role>
        <Ids><id>urn:3DXML:CATReplImage.3dxml#16</id></Ids>
      </PLMRelation>
      <V_MatDomain>Rendering</V_MatDomain>
    </MaterialDomain>
    <MaterialDomainInstance id="18" xsi:type="MaterialDomainInstanceType" name="Rendering.1">
      <PLM_ExternalID>Rendering.1</PLM_ExternalID>
      <IsAggregatedBy>15</IsAggregatedBy>
      <IsInstanceOf>17</IsInstanceOf>
    </MaterialDomainInstance>
  </CATMaterialRef>
</Model_3dxml>
```



5.2 Rendering Materials

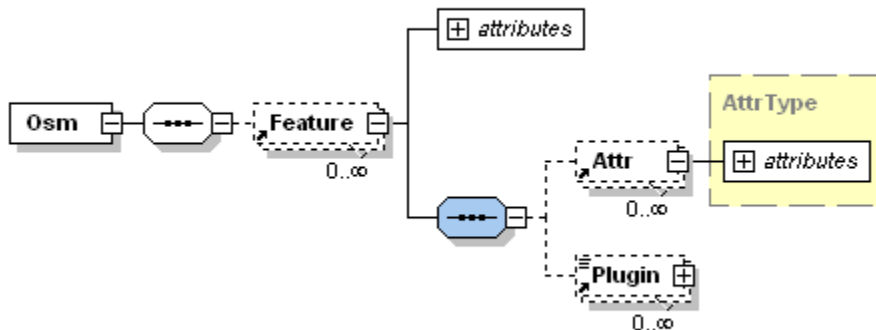
The material domains are describes in specific 3DRep file. This file is named "*DomainType Name.3DRep*".

In the case of the Rendering domain, this file contains all the necessary attributes to define both basic and textured materials.

Basic materials are those you can assign to an object so that it looks uniformly colored like plastic or rubber.

Textured materials are irregular patterns that simulate a texture surface like marble or wood. You can modify the rendering aspect by specifying parameters such as the ambient, diffuse, specular coefficient or the reflectivity.

For more details on attributes see the reference guide or XSD.

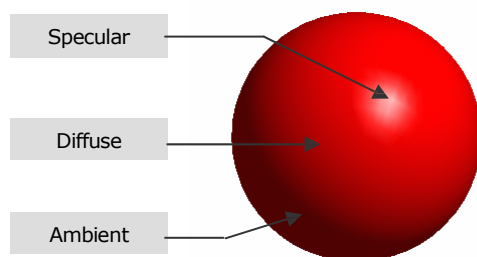


This file contains a "Feature" node with:

- ¾ An Alias equal to "RenderingRootFeature" with the following attributes exposed:
 - The Root Id (attribute name: "id")
- ¾ An Alias equal to "RenderingFeature" with optional attributes to describe the rendering behaviour for basic materials or textured materials

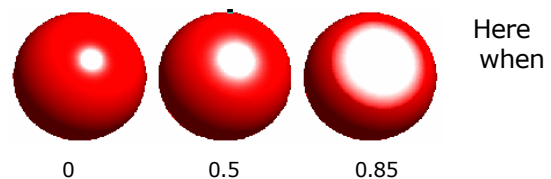
5.2.1 Basic materials:

A basic material is a material without texture. The material itself can be assigned with a simple color but the way this color appears to you depends on the percentage of incoming RGB light it reflects.



To describe this effect, materials can be assigned with ambient, diffuse and specular colors, which determine the ambient, diffuse and specular reflectance. The size of the specular light spot varies exponentially and can be adjusted by using a specular exponent.

is an example of what you are likely to get you modify the value of this specular exponent.



This kind of materials has also parameters defining other effects such as emissive color, blend color, transparency, reflectivity or refraction.

Here are examples of material with different colors, transparency and reflectivity.



Here is the complete list of attributes that define a basic material:

| Name | Type | Limit values | Mandatory | Comment | Default value |
|------------------|------------------|--------------|-----------|--------------------------|----------------|
| AmbientCoef | double | 0 to 1 | No | Ambient coefficient | 0.5 |
| AmbientColor | tab of 3 doubles | 0 to 1 | No | RGB ambient color | 1., 0.75, 0.07 |
| DiffuseCoef | double | 0 to 1 | No | Diffuse coefficient | 1. |
| DiffuseColor | tab of 3 doubles | 0 to 1 | No | RGB diffuse color | 1., 0.75, 0.07 |
| SpecularCoef | double | 0 to 1 | No | Specular coefficient | 1. |
| SpecularColor | tab of 3 doubles | 0 to 1 | No | RGB specular color | 1., 1., 1. |
| SpecularExponent | double | 0 to 1 | No | Specular exponent | 0. |
| EmissiveCoef | double | 0 to 1 | No | Emissive coefficient | 0. |
| EmissiveColor | tab of 3 doubles | 0 to 1 | No | RGB emissive color | 1., 1., 1. |
| BlendColor | tab of 3 doubles | 0 to 1 | No | RGB blend color | None |
| Transparency | double | 0 to 1 | No | Transparency coefficient | 0. |
| Reflectivity | double | 0 to 1 | No | Reflectivity coefficient | 0.18 |
| Refraction | double | 0 to 1 | No | Refraction coefficient | 1. |

A "Non-Mandatory" attribute means that even if this attribute is missing in 3dxml file, the material can be visualized. In that case, each application unstreaming the 3dxml could provide its own default value. Those mentioned for your information in the tables are those used by CATIA V6.

Note that specularExponent in 3DXML is not the one usually used in OpenGL. To have the equivalence, one needs first to bring the OpenGL value between 0 and 1, then applies the following formula:

$$(3dxml\ specular\ exponent) = 1 - (openGL\ specularExponent)^3$$

Here is a sample of basic material. This is the Red painting used for Quad's chasis. For more details see the XSD documentation.



the

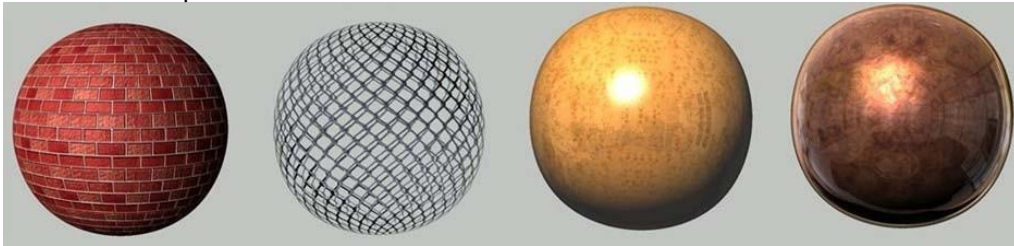
```
<Osm xmlns="http://www.3ds.com/xsd/osm.xsd">
  <Feature Alias="RenderingRootFeature" Id="1" StartUp="RenderingRootFeature"/>
  <Feature Alias="RenderingFeature" Id="2" StartUp="RenderingFeature" Aggregating="1">
    <Attr Name="AmbientCoef" Type="double" Value="0.4"/>
    <Attr Name="AmbientColor" Type="double" value="[0.829427,0.0455729,0.065169]"/>
    <Attr Name="DiffuseCoef" Type="double" Value="1"/>
    <Attr Name="DiffuseColor" Type="double" Value="[0.951667,0.0816666,0.125167]"/>
    <Attr Name="SpecularCoef" Type="double" Value="1"/>
    <Attr Name="SpecularColor" Type="double" Value="[1,0.641667,0.641667]"/>
    <Attr Name="EmissiveCoef" Type="double" Value="0"/>
    <Attr Name="EmissiveColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="BlendColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="SpecularExponent" Type="double" Value="0.3"/>
    <Attr Name="Transparency" Type="double" Value="0"/>
    <Attr Name="Reflectivity" Type="double" Value="0.15"/>
    <Attr Name="Refraction" Type="double" Value="1"/>
    <Attr Name="MappingType" Type="int" Value="2"/>
    <Attr Name="PreviewType" Type="int" Value="1"/>
    <Attr Name="TranslationU" Type="double" Value="0"/>
    <Attr Name="TranslationV" Type="double" Value="0"/>
    <Attr Name="Rotation" Type="double" Value="0"/>
    <Attr Name="ScaleU" Type="double" Value="300"/>
    <Attr Name="ScaleV" Type="double" Value="300"/>
    <Attr Name="FlipU" Type="boolean" Value="false"/>
    <Attr Name="FlipV" Type="boolean" Value="false"/>
    <Attr Name="TextureDimension" Type="int" Value="2"/>
    <Attr Name="TextureFunction" Type="int" Value="0"/>
    <Attr Name="AlphaTest" Type="boolean" Value="false"/>
    <Attr Name="Filtering" Type="int" Value="1"/>
    <Attr Name="WrappingModeU" Type="int" Value="1"/>
    <Attr Name="WrappingModeV" Type="int" Value="1"/>
  </Feature>
</Osm>
```

5.2.2 Textured materials:

A textured material is a material with a diffuse color which is not a simple color but an image or a complex variation of colors. A textured material reference an image in the CATRepImage.3dxml file included in the 3DXML archive. For more details on Images see chapter 6.

This material has the same parameters as a basic material but it has also some specific parameters to handle the image and position the texture on the geometry. Those parameters define a mapping operator.

Here are examples of material with different textures.



You can also see that some materials are reflecting the surrounding environment. A specific mapping is required for that.

To define a textured material, here is the complete attribute list to add to basic material one:

| Name | Type | Limit values | Mandatory | Comment | Default value |
|------------------|----------|--|-----------|--|---------------|
| MappingType | int | -2 π to +2 π greater than 0 greater than 0 | No | Texture mapping type | 2 |
| PreviewType | int | | No | Mapping operator type | 3 |
| TranslationU | double | | No | Texture position U | 0. |
| TranslationV | double | | No | Texture position V | 0. |
| Rotation | double | | No | Texture rotation | 0. |
| ScaleU | double | | No | Texture scale U | 100. |
| ScaleV | double | | No | Texture scale V | 100. |
| FlipU | boolean | | No | Texture flip U | False |
| FlipV | boolean | | No | Texture flip V | False |
| TextureDimension | int | | No | Texture dimension | 2 |
| TextureFunction | int | | No | Texture function (decals, blend, ...) | 0 |
| AlphaTest | boolean | | No | Alpha channel use | None |
| Filtering | int | | No | Texture filtering mode (linear, trilinear,...) | 1 |
| WrappingModeU | int | | No | Texture transformation U (clamp, repeat) | 1 |
| WrappingModeV | int | | No | Texture transformation V (clamp, repeat) | 1 |
| TextureImage | external | | Yes | Urn towards texture image file | None |
| ReflectionImage | external | | No | Urn towards environment reflection image file | None |

Here is a sample of textured material used for the Quad's wheels.
more details see the XSD documentation.



For

```
<Osm xmlns="http://www.3ds.com/xsd/osm.xsd" >
  <Feature Alias="RenderingRootFeature" Id="1" StartUp="RenderingRootFeature"/>
  <Feature Alias="RenderingFeature" Id="2" StartUp="RenderingFeature" Aggregating="1">
    <Attr Name="AmbientCoef" Type="double" Value="0.4"/>
    <Attr Name="AmbientColor" Type="double" Value="[0.804167,0.804167,0.804167]"/>
    <Attr Name="DiffuseCoef" Type="double" Value="0.8"/>
    <Attr Name="DiffuseColor" Type="double" Value="[0.804167,0.804167,0.804167]"/>
    <Attr Name="SpecularCoef" Type="double" Value="0"/>
    <Attr Name="SpecularColor" Type="double" Value="[0.804167,0.804167,0.804167]"/>
    <Attr Name="EmissiveCoef" Type="double" Value="0"/>
    <Attr Name="EmissiveColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="BlendColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="SpecularExponent" Type="double" Value="0"/>
    <Attr Name="Transparency" Type="double" Value="0"/>
    <Attr Name="Reflectivity" Type="double" Value="0"/>
    <Attr Name="Refraction" Type="double" Value="1"/>
    <Attr Name="MappingType" Type="int" Value="1"/>
    <Attr Name="PreviewType" Type="int" Value="1"/>
    <Attr Name="TranslationU" Type="double" Value="0"/>
    <Attr Name="TranslationV" Type="double" Value="0"/>
    <Attr Name="Rotation" Type="double" Value="0"/>
    <Attr Name="ScaleU" Type="double" Value="35"/>
    <Attr Name="ScaleV" Type="double" Value="35"/>
    <Attr Name="FlipU" Type="boolean" Value="false"/>
    <Attr Name="FlipV" Type="boolean" Value="false"/>
    <Attr Name="TextureDimension" Type="int" Value="2"/>
    <Attr Name="TextureFunction" Type="int" Value="1"/>
    <Attr Name="AlphaTest" Type="boolean" Value="false"/>
    <Attr Name="Filtering" Type="int" Value="1"/>
    <Attr Name="WrappingModeU" Type="int" Value="1"/>
    <Attr Name="WrappingModeV" Type="int" Value="1"/>
    <Attr Name="TextureImage" Type="external" Value="urn:3DXML:CATReplImage.3dxml#16"/>
  </Feature>
</Osm>
```

Here is a sample of a simple material using a specific image as reflection map.



```
<Osm xmlns="http://www.3ds.com/xsd/osm.xsd" >
  <Feature Alias="RenderingRootFeature" Id="1" StartUp="RenderingRootFeature"/>
  <Feature Alias="RenderingFeature" Id="2" StartUp="RenderingFeature" Aggregating="1">
    <Attr Name="AmbientCoef" Type="double" Value="0.36"/>
    <Attr Name="AmbientColor" Type="double" Value="[0.614218,0.827864,0.980469]"/>
    <Attr Name="DiffuseCoef" Type="double" Value="0.84"/>
    <Attr Name="DiffuseColor" Type="double" Value="[0.855709,0.921649,0.96875]"/>
    <Attr Name="SpecularCoef" Type="double" Value="1"/>
    <Attr Name="SpecularColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="EmissiveCoef" Type="double" Value="0"/>
    <Attr Name="EmissiveColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="BlendColor" Type="double" Value="[1,1,1]"/>
    <Attr Name="SpecularExponent" Type="double" Value="0"/>
    <Attr Name="Transparency" Type="double" Value="0"/>
    <Attr Name="Reflectivity" Type="double" Value="0.4"/>
    <Attr Name="Refraction" Type="double" Value="1"/>
    <Attr Name="ReflectionImage" Type="external" Value="urn:3DXML:CATReplImage.3dxml#7"/>
  </Feature>
</Osm>
```

5.3 Texture Mapping

3D XML supports three mapping types to manipulate:

- ¾ textures simulating reflections of an environment
- ¾ textures mapped directly on the mesh coordinates
- ¾ Complex mapping defined by a mapping operator.

The mapping operator defines the way a texture is applied onto a surface.

5.3.1 Mapping Types

The mapping type defines the way a texture is applied onto a surface. 3D XML supports three mapping types:

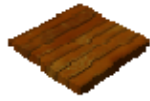
- ¾ ENVIRONMENT_MAPPING ("MappingType" attribute set to 0)
The way the texture is applied to surfaces is automatically computed to simulate reflection. The texture coordinates of the mesh are ignored and no mapping operator should be specified. This mapping type is mainly used to simulate reflection on shiny surfaces.
- ¾ IMPLICIT_MAPPING ("MappingType" attribute set to 1)
This mapping type supposes texture coordinates are already defined in the mesh vertex buffer. The mapping operator if any and all related attributes (scale, translation, rotation, flip,...) are ignored.
- ¾ MAPPING_OPERATOR ("MappingType" attribute set to 2)
The mapping operator must be provided. This operator defines the way texture coordinates are computed on the mesh. If texture coordinates are declared in the vertex buffer, they are redefined by the new ones.

5.3.2 Mapping Operators

Mapping operators are optional. They define how the bitmap is projected onto the surface. The mapping operator is only required when the mapping type is set to 2. If you don't specify any mapping operator (the mapping type being set to 2), a default planar mapping is applied.

In the 3DXML file, mapping operator type is defined through the "PreviewType" attribute.

This mapping operator carries information about the way (u,v) texture coordinates are computed before they are sent to the graphic adapter. Several mapping operators are supported.



PLANAR MAPPING
(PreviewType = 0)



SPHERICAL MAPPING
(PreviewType = 1)



CYLINDRICAL MAPPING
(PreviewType = 2)



CUBICAL MAPPING
(PreviewType = 3)



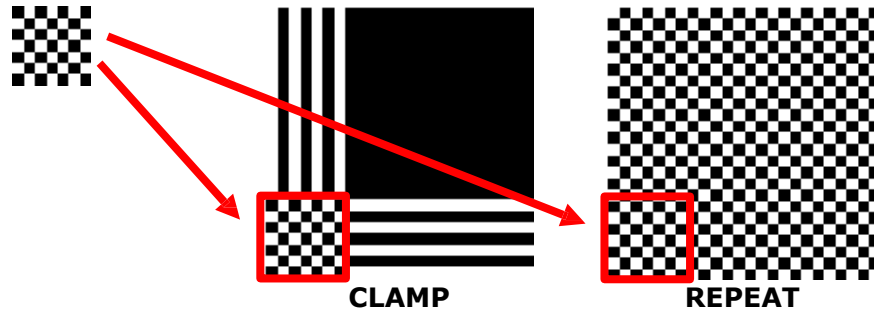
ADAPTIVE MAPPING
(PreviewType = 4)

🔧 NB: *ADAPTIVE MAPPING* is a kind of best fit mapping of a texture onto a geometry compromising between distortions and seams.

5.3.3 Texture transformation

Texture transformation describes how the bitmap is transformed prior to being applied to objects. By default, no transformation is applied on textures. The image can be translated, scaled, flipped, wrapped in both directions and rotated.

Here is an example of a wrapped texture.



Texture wrapping mode are define using "WrappingModeU" and "WrappingModeV" attributes:

- ¾ *CLAMP mode*: set "WrappingModeU" or "WrappingModeV" attributes to 0.
- ¾ *REPEAT mode*: set "WrappingModeU" or "WrappingModeV" attributes to 1.

5.3.4 Additional information for specific texture behavior

A texture is usually a rectangular pixel image (2D) but 3DXML also support 1D texture which are actually a line of pixels for specific purpose such as zebra mapping. So the texture's dimension has to be specified.

When applying a texture on a colored surface it is possible to define how this color and the texture color interact. This is called TextureFunction. The result can be:

- ¾ *DECAL*: The texture color is used as the final color. It is painted on its support like a decal would be applied ("TextureFunction" attribute should be set to 0).
- ¾ *MODULATE*: This technique combines the effects of lighting with texturing ("TextureFunction" attribute should be set to 1).
- ¾ *BLEND*: A constant color is blended with the texture ("Texture Function" attribute should be set to 2).
- ¾ *REPLACE*: The color values of the object to be textured are replaced with the texture color data ("TextureFunction" attribute should be set to 3).

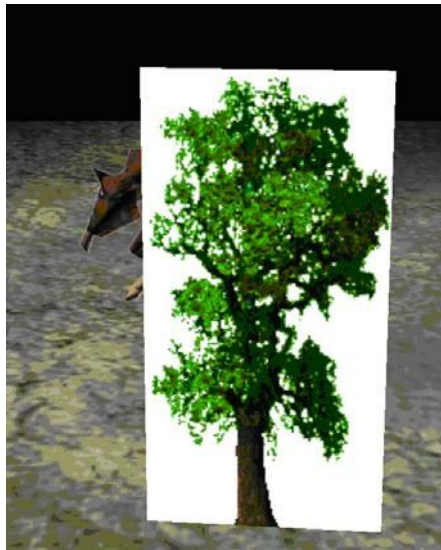
Applying texture images onto geometry can introduce visible noise due to sampling. To avoid such unwanted effect one can define a Filtering method.

Filtering defines the texel (fundamental unit of texture space) interpolation method. The default value is LINEAR. The texture image is a discrete array of texels, but the texture coordinates vary continuously. The following filtering methods are allowed:

- ¾ *NONE*: No filtering is done, the nearest texel is copied ("Filtering" attribute should be set to 0).

- ¾ **LINEAR**: Bilinear interpolation filtering is done. A weighted average of a 2×2 area of texels surrounding the desired pixel is used ("Filtering" attribute should be set to 1).
- ¾ **TRILINEAR**: Trilinear filtering method linearly interpolates pixel color, using the texels of the two nearest mipmap textures. The mipmap textures are automatically generated ("Filtering" attribute should be set to 2).
- ¾ **ANISOTROPIC**: The difference between the texture aspect ratio and the projected surface aspect ratio often results in more blur than needed. Anisotropic filtering method minimizes excessive blurring by correcting the ratio of the texture image ("Filtering" attribute should be set to 3).

An texture can have an alpha channel which specify pixels that can be transparent. This information can be taken into account or not. The "AlphaTest" attribute defines whether the alpha channel of texture is taken into account or not. If the alpha test parameter is valuated to **TRUE**, the alpha channel is used. Otherwise it is ignored.



Alpha test = FALSE



Alpha test = TRUE

5.4 Material usage

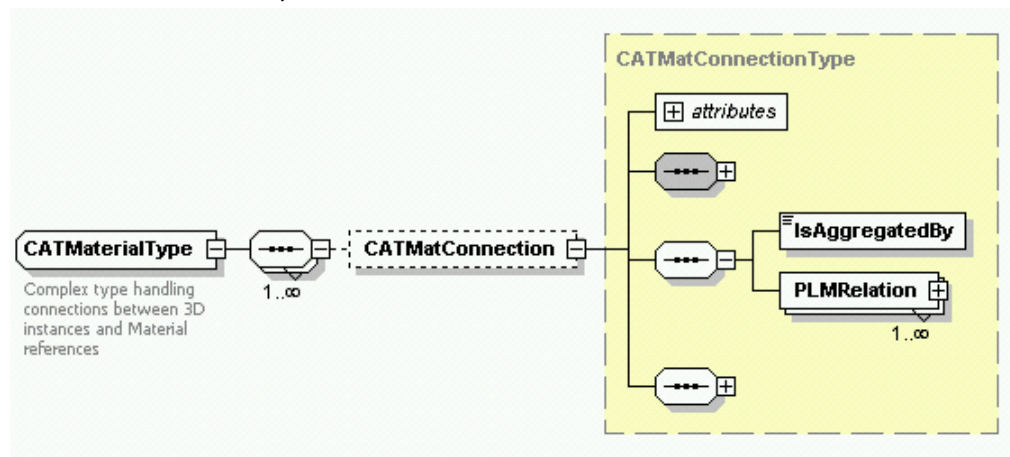
Material can be applied both on Product Structure and in a Mesh. But the ways to apply it are different.

5.4.1 Material on Product Structure

After the `<ProductStructure>` section, there is a `<CATMaterial>` section in which the relation between occurrences and Materials are described. Material application on an occurrence is described in the `<CATMatConnection>` element.

This element defines what material reference is used, how it is used, where it is applied and in which context.

- ¾ The “what” is a reference to a `<CATMatReference>` previously described.
- ¾ The “how” is a manner to qualify the material application. There are 2 kinds of materials: Core material also called “MadeOf” which is the constitutive material of a part and Covering material also called “DressBy” which is a layer applied on top of the part surface. For Rendering purpose the “DressBy” usage is the preferred one.
- ¾ The “where” is a reference to an occurrence in the product structure.
- ¾ The “context” is a notion helping to define a hierarchy of material when applied on sub-assemblies. For instance you can specify that a sub assembly is painted in red and that in a larger assembly context this sub assembly is partially colored in blue. So by definition the sub-assembly is still red but when displayed in its father context it will be covered with blue paint.



The `<CATMaterial>` section in the Product.3DXML file contains:

- ¾ A specific element, named `<CATMatConnection>`. This element will expose:
 - The connection Id
 - The connection Name
 - The discipline: Applied Material

- The layer number when several materials are stacked on the same occurrence. A material applied on top of another one is always covering it. There is no blend type management. One can consider the default behavior is equivalent to an Alpha blend mode.
- The material usage:
 1. Occurrence is "MadeOf" a Core material
 2. Occurrence is "DressBy" a Covering material
- A position matrix 3x4 (element names: <V_Matrix_1> to <V_Matrix_12>)
- Two <PLMRelation> elements:
 1. Defining a urn on an occurrence id toward the product structure section on which material will be applied
 2. Defining a pointer on a material reference inside the CATMaterialRef.3dxml file

Here is a sample of Red painting applied on the Quad's Chassis.



```

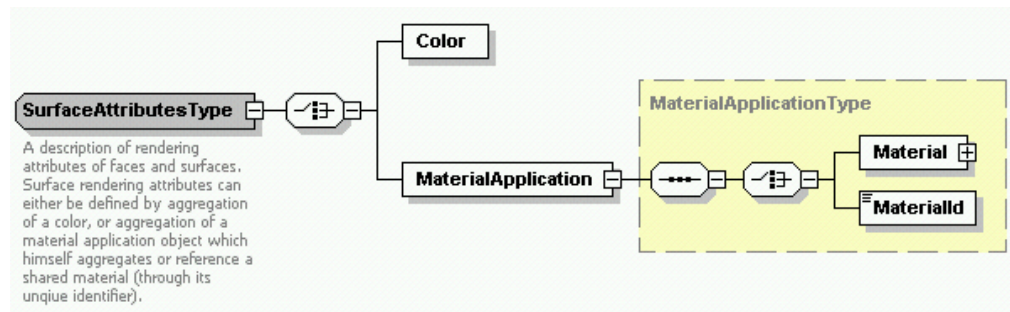
<CATMaterial>
  <CATMatConnection id="37" xsi:type="CATMatConnectionType"
    name="Fire Red_Material Library">
    <PLM_ExternalID>Fire Red_Material Library</PLM_ExternalID>
    <IsAggregatedBy>1</IsAggregatedBy>
    <PLMRelation>
      <C_Semantics>Reference3</C_Semantics>
      <C_Role>CATMaterialToReferenceLink</C_Role>
      <Ids><id>urn:3DXML:CATMaterialRef.3dxml#34</id></Ids>
    </PLMRelation>
    <PLMRelation>
      <C_Semantics>Reference5</C_Semantics>
      <C_Role>CATMaterialDressByLink</C_Role>
      <Ids>
        <id>urn:3DXML:Disassemble_Quad.3dxml#2</id>
        <id>urn:3DXML:Disassemble_Quad.3dxml#4</id>
        <id>urn:3DXML:Disassemble_Quad.3dxml#6</id>
      </Ids>
    </PLMRelation>
    <V_Layer>1</V_Layer>
    <V_Applied>2</V_Applied>
    <V_Matrix_1>1</V_Matrix_1>
    <V_Matrix_2>0</V_Matrix_2>
    <V_Matrix_3>0</V_Matrix_3>
    <V_Matrix_4>0</V_Matrix_4>
    <V_Matrix_5>1</V_Matrix_5>
    <V_Matrix_6>0</V_Matrix_6>
    <V_Matrix_7>0</V_Matrix_7>
    <V_Matrix_8>0</V_Matrix_8>
    <V_Matrix_9>1</V_Matrix_9>
    <V_Matrix_10>0</V_Matrix_10>
    <V_Matrix_11>0</V_Matrix_11>
    <V_Matrix_12>0</V_Matrix_12>
  </CATMatConnection>
</CATMaterial>

```

5.4.2 Material on a mesh (geometric representation)

For tessellated format, a material can be applied directly onto surface elements. The way the material is used is contained in the 3DRep file of the part itself (see Mesh chapter for more information).

In this case there is no need to qualify the type of usage or the context. The material is referenced by the mesh itself. In this version material is used for rendering purpose only.



Within tessellated geometry definition, the `<SurfaceAttributes>` can have a `<MaterialApplication>` element in order to allow application of material onto a geometric element



Here is an example:

```
<SurfaceAttributes>
  <MaterialApplication mappingChannel="0"
    blendType="REPLACE" >
    <MaterialId id="urn:3DXML:CATMaterialRef.3dxml#3"/>
  </MaterialApplication>
</SurfaceAttributes>
```

5.4.3 How to Apply Textured Materials onto a Surface

In the sections above, we have seen how to apply Textured Material on Product Structure using `<CATMaterial>`. Another way is to define the Textured Material directly on the Mesh geometry. Texture coordinates can be directly defined in the `<VertexBuffer>`. For more information see the Mesh chapter (Chapter 8).

The surface attributes may contain multiple texture coordinate channels of materials to be mapped on the mesh. Multi-texturing allows obtaining specific texture effects such as light maps, textures or image incrustation. This requires a specific attribute to define the way the textures are blended.

Within tessellated geometry definition, the SurfaceAttributes section can have several MaterialApplication sections. Section order defines the staking order of the materials. Each MaterialApplication section contains:

- ¾ Attributes:
 - The channel number which refers to the channel of a `<VertexBuffer>`.
 - The optional blend mode controlling how the materials are blended together to give the final result.
 - The optional side attribute controlling on which face side the texture will be mapped (two faces by default)
- ¾ A `<MaterialId>` element to identify the material used Thanks to an URN toward the CATMaterialRef.3dxml file

Here is an example with 2 layers:

```
<SurfaceAttributes>
  <MaterialApplication mappingChannel="0" blendType="REPLACE" >
    <MaterialId id="urn:3DXML:CATMaterialRef.3dxml#3"/>
  </MaterialApplication>
  <MaterialApplication mappingChannel="1" blendType="ALPHA_TRANSPARENCY">
    <MaterialId id="urn:3DXML:CATMaterialRef.3dxml#7"/>
  </MaterialApplication>
</SurfaceAttributes>
```



Single texture



Multi-texture

6. Images

3DXML allows references to images. The referenced images however are part of the 3DXML archive. Image files are self contained in and have their own separate document object model. All the references to images are inside in a dedicated CATRepImage.3dxml file.

Images in 3DXML are defined using the `<CATRepImage>` tag. It is collection of `<CATRepresentationImage>`. Each `<CATRepresentationImage>` identifies an image which can be referenced from else where in the 3DXML document using a URN notation (described below).



```
<Model_3dxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.3ds.com/xsd/3DXML">
  <Header>
    <SchemaVersion>4.0</SchemaVersion>
    <Title>3D XML File</Title>
    <Author/>
    <Generator>DXP</Generator>
    <Created>2007-03-28</Created>
  </Header>
  <CATReplImage>
    <CATRepresentationImage id="16"
      associatedFile="urn:3DXML:concrete.jpg"
      xsi:type="CATRepresentationImageType" name="concrete" format="jpg">
      <PLM_ExternalID>concrete</PLM_ExternalID>
      <V_PrimaryMimeType>jpg</V_PrimaryMimeType>
      <V_PrimaryFileName>concrete.jpg</V_PrimaryFileName>
    </CATRepresentationImage>
    <CATRepresentationImage id="59"
      associatedFile="urn:3DXML:dashboard.jpg" xsi:type="CATRepresentationImageType"
      name="dashboard" format="jpg">
      <PLM_ExternalID>dashboard</PLM_ExternalID>
      <V_PrimaryMimeType>jpg</V_PrimaryMimeType>
      <V_PrimaryFileName>dashboard.jpg</V_PrimaryFileName>
    </CATRepresentationImage>
  </CATReplImage>
</Model_3dxml>
```

The following URN is used to access the image representation:
"urn:3DXML:CATRepImage.3dxml#12"

Attributes for image representation:

- id: 3DXML identifier (c.f. 3DXML documentation or related XSD)
- associatedFile: Specifies the embedded structure or an external link of the given Image.
- name: Descriptive name for the image.
- format: Type of image. E.g. jpg, bmp, tiff, etc

Please note that the 2 tags "V_PrimaryFileName" and "V_PrimaryMimeType" are mandatory for each "CATRepresentationImage" node, and should contain the image name and the image format of the associated image file.

7. Viewpoints

Viewpoints, also referred to as cameras, define the way virtual objects are displayed on the two dimensional screen. 3D XML supports a number of 2D and 3D viewpoints which are described in this section.

7.1 2D Viewpoints

A 2D viewpoint defines the way 2D objects located in a 2D planar space are projected onto a virtual screen. The attributes which control a 2D viewpoint are:

- ✚ the eye position
- ✚ the roll angle that is the angle between the x-axis of the 2D virtual scene and the x-axis of the virtual screen
- ✚ the height of the 2D area to be displayed on the virtual screen.

The width of the visualized area depends on the graphic characteristics of the device onto which the projection is made (numbers of pixels and pixel dimensions).

7.2 3D Viewpoints

A 3D viewpoint describes how the 3D virtual space is projected onto the 2D space of the screen.

7.2.1 Projection Viewpoint

A projection viewpoint is defined by the following attributes:

- ✚ the eye position which is the point where the observer's eye is located
- ✚ the sight axis which is the direction in which the eye is looking
- ✚ the right and up axes which complete the orthonormalized trihedron (right axis, sight axis, up axis)
- ✚ the target distance which is the point at which the observer is looking. The target point is along the sight axis at a distance from the eye position equal to the target distance
- ✚ the near and far planes which are used to define the objects to be seen. Only the objects belonging to the zone between these two planes are displayed. The 3D objects between the eye and the near plane are not seen. The 3D objects beyond the far plane are not seen. The definition of these planes is optional.

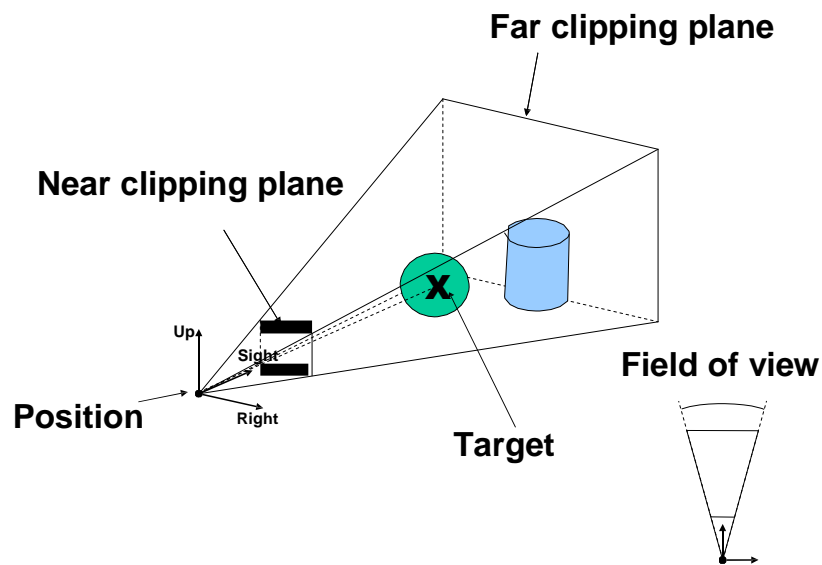
How to Specify a Projection Viewpoint

Use the `<Viewpoint>` tag with the `ProjectionViewpointType` type as indicated below.

```
<Viewpoint xsi:type="ProjectionViewpointType" nearPlaneDistance="1.0" farPlaneDistance="10000.0"
targetDistance="23.51">
  <Position>19.1636 8.9445 10.2740</Position>
  <Sight>-0.577350 -0.577350 -0.577350 </Sight>
  <Right>-0.707107 -0.707107 -0.707107</Right>
  <Up>-0.408248 -0.408248 -0.408248</Up>
</Viewpoint>
```

7.2.2 Perspective Viewpoint

A perspective viewpoint is a projection viewpoint with an additional attribute referred to as the "field of view" that defines the seen solid angle.



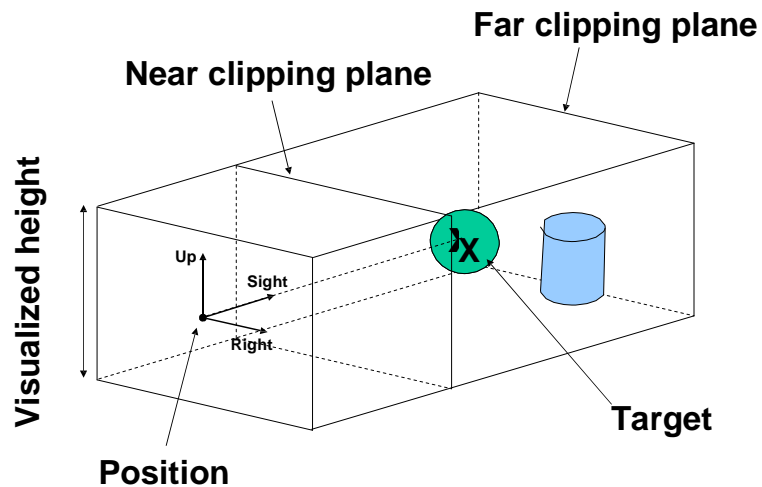
How to Specify a Perspective Viewpoint

Use the `<Viewpoint>` tag. The `PerspectiveViewpointType` type, an extension of the `ProjectionViewpointType` type, defines the `fieldOfView` parameter.

```
<Viewpoint xsi:type="PerspectiveViewpointType" nearPlaneDistance="1.0" farPlaneDistance="10000.0"
targetDistance="23.51" fieldOfView="0.268">
  <Position>19.1636 8.9445 10.2740</Position>
  <Sight>-0.577350 -0.577350 -0.577350 </Sight>
  <Right>-0.707107 -0.707107 -0.707107</Right>
  <Up>-0.408248 -0.408248 -0.408248</Up>
</Viewpoint>
```

7.2.3 Parallel Viewpoint

A parallel viewpoint is a parallel projection along the sight axis. The additional attribute, the visualized height, defines the zone of the 3D virtual world to be projected onto the virtual screen. The width of the visualized area depends on the graphic characteristics of the device onto which the projection is made (numbers of pixels and pixel dimensions).



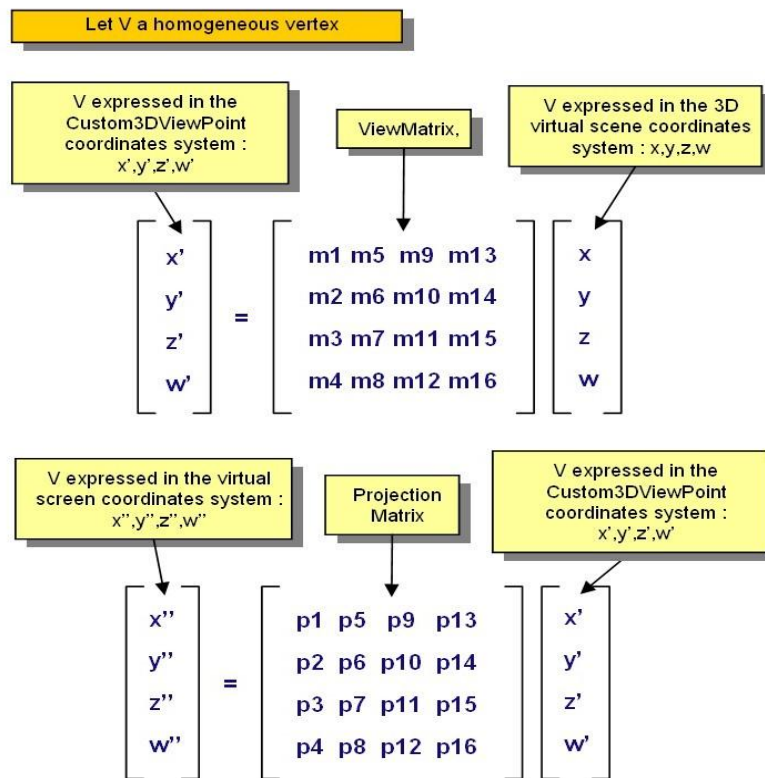
How to Specify a Parallel Viewpoint

The `ParallelViewpointType` type, an extension of the `ProjectionViewpointType` type, defines the `visualizedHeight` parameter.

```
<Viewpoint xsi:type="ParallelViewpointType" nearPlaneDistance="1.0" farPlaneDistance="10000.0"
visualizedHeight="543.88" targetDistance="23.5119">
  <Position>23.5119 0 0</Position>
  <Sight>-1 0 0</Sight>
  <Right>0 1 0</Right>
  <Up>0 0 1</Up>
</Viewpoint>
```


7.2.4 Customized 3D Viewpoint

A customized 3D viewpoint describes a projection viewpoint by two 4x4 matrices. The view matrix defines the position of the viewpoint in the 3D space. The projection matrix describes the 3D transformation from the camera axis system to the virtual screen axis system. Each matrix is a list of 16 double values stored column by column.



How to Specify a Customized 3D Viewpoint

Use the CustomViewpoint3DType type as in the example below:

```
<Viewpoint xsi:type="CustomViewpoint3DType" >
  <ViewMatrix>.....</ViewMatrix>
  <ProjectionMatrix>.....</ProjectionMatrix>
</Viewpoint>
```

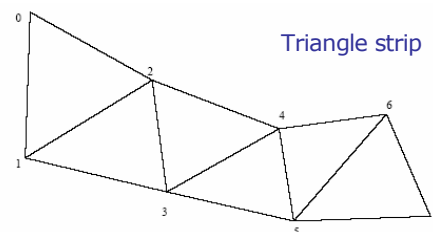
8. Mesh

In 3D XML Product Structure, geometric representations are associated with references. One of the formats authorized for representations is an XML-based faceted representation, made up of faces and edges. This section exposes the polygonal representation used to describe the faceted representation of a model, the graphic primitives making up its faces and edges and its geometrical organization through the *Vertex Buffer* object. This representation is contained in a separate file in the 3dxml archive, using the extension *3DRep*.

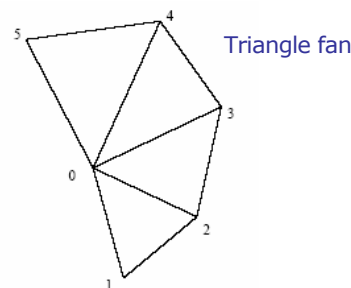
8.1 Concepts

A faceted representation is made up of vertices connected in triangles and which are themselves gathered into fans or strips to reduce the size of the polygonal mesh.

A triangle strip is a series of triangles organized so that each new triangle shares two vertices with the previous triangle. It is represented by a list of indices referencing vertices in the vertex buffer.

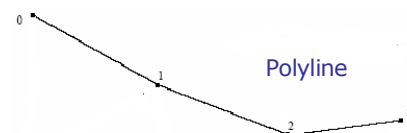


A triangle fan is a series of triangles that share a common central vertex. It is represented by a list of indices referencing vertices in the vertex buffer.



Independent triangles, triangle strips and triangle fans with the same graphic attributes and material are grouped in a higher level object called face primitive (FaceGPTYPE).

To represent edges of the mesh, vertices can also be connected by polylines (PolylineGPTYPE).



8.2 Vertex Buffer

The purpose of the vertex buffer is to store and optimize the definition of vertices used in faces.

A vertex buffer contains for each vertex:

- the coordinates x, y and z
- an optional normal (nx, ny, nz)
- one or more sets of texture coordinates.
- two optional sets of colors (diffuse and specular). Colors per vertex enable to obtain specific visual effects, or to render more rapidly some standard effects such as lighting, shadows, ambient occlusion, etc....

```
<VertexBuffer>
  <Positions>-50 -50 -50,-50 50 -50 50 -50 50 50 -50,-50 50 -50,-50 50 50 50 -50,50 50 50,-50 50 50,-50
50,50 50 50,50 -50 50,-50 -50 50,-50 -50,50 -50 50,50 -50 -50,-50 50 -50,-50 50 50,-50 -50 50,50 -50
-50,50 50 -50,50 -50 50,50 50 50 </Positions>
  <Normals>0 0 -1,0 0 -1,0 0 -1,0 0 -1,0 1 0,0 1 0,0 1 0,0 1 0,0 1 0,0 1 0,0 1 0,-1 0,0 -1 0,0 -1 0,0,-
1 0 0,-1 0 0,-1 0 0,1 0 0,1 0 0,1 0 0,1 0 0</Normals>
  <TextureCoordinates dimension="2D" channel="0">1 0,1 1,0 0,0 1,1 0,1 1,0 0,0 1,0 1,0 0,1 1,1 0,0 1,0 0,1 1,1 0,0
0,1 0,0 1,1 1,0 0,1 0,0 1,1 1</TextureCoordinates >
</VertexBuffer>
```

Texture coordinates allow to define the way a texture is mapped on the mesh.

The **dimension** attribute of the `<TextureCoordinates>` element defines whether this set of texture coordinates applies to a mono-dimensional or a bi-dimensional image.

The channel attribute is used by textured materials to define which set of texture coordinates of a mesh they are using.

For each vertex of the mesh, 1 or 2 coordinates (according to the dimension of the texture) specify which pixel of the texture is applied to the vertex.

For bi-dimensional textures, the convention used in 3DXML is the same as OpenGL : the (0,0) coordinates refers to the bottom left corner of the image and (1,1) coordinates to the top right corner of the the image, as shown in the following picture :

Usually, textures coordinates range from 0 to 1. However, lower and greater values are accepted, and allow to repeat the texture on the mesh when the wrapping modes of the material are set to "REPEAT" (see wrapping modes in Texture Transformation section).

8.3 Polygonal Representation

A polygonal representation is a `PolygonalRepType` object aggregating the following objects:

- graphic attributes
- optional level of details
- Primitive sets which are collections of graphic primitives (faces and polylines).
- A `VertexBuffer` object to store and optimize the definition of vertices used in faces and level of details

The global 3D XML writing of a polygonal representation is something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<XMLRepresentation version="1.2" xmlns="http://www.3ds.com/xsd/3DXML" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.3ds.com/xsd/3DXML">
  <Root xsi:type=" PolygonalRepType" id="828" accuracy="0.2" solid="1">
    <!-- 1 - graphic attributes -->
    ...
    <!-- 2 – level of details -->
    ...
    <!-- 3 - the primitive sets -->
    ...
    <!-- 4 - the vertex buffer -->
    ...
  </Root>
</XMLRepresentation>
```

The detailed writing of the polygonal representation has to be explained with respect to the type of primitives (faces and edges) used in the model.



8.3.1 Graphic Attributes in a Polygonal Representation

The graphic attributes (surface attributes, line attributes and materials) defined at the `PolygonalRep` level apply to all the primitive sets and primitives included in the `Polygonal Representation` unless they are themselves assigned graphic attributes.

8.3.2 Primitive Sets

Primitives are organized into sets so that they can share graphic attributes or material definitions.

The `PolygonalRepType` object can aggregate two kind of primitive sets: `FaceSetType`

-  which contains face representations (`FaceGP` object) `PolylineSetType`
-  which contains edge representations (`PolylineGP` object)

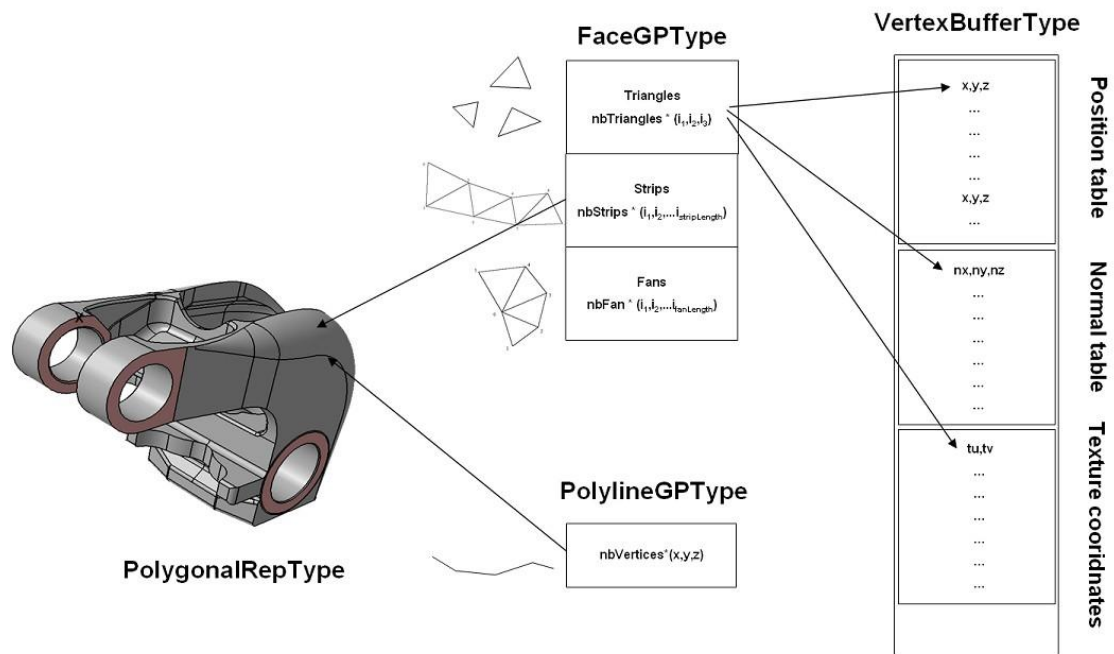
Graphic attributes can be optionally defined on primitive sets. If they are, these attributes are then inherited by aggregated primitives unless they are already assigned graphic attributes. Surface attributes can be set on a `FaceSet` object. Line attributes can be set on `PolylineSet` objects.

The polygonal representation can aggregate one or more sets of each type. Both categories are optional elements of the `<PolygonalRepType>` object.

Material versus simple colors :

When both material and color specification are available in a graph, the visualization can depend on the view mode. For example, when view mode is "simple shading", the color could be visualized, and when view mode is "shading with material", the material could be visualized.

8.4 3D XML Graphic Primitives



8.4.1 The Face Primitive

The Face primitive (**FaceGPTType**) is an indexed primitive. Indexed primitives are graphic primitives which can share their vertices. Their vertices are referenced into a vertex buffer object by their indices.

How to Describe Face Primitives in a Polygonal Representation

The description of face primitives requires a vertex buffer definition. The 3D XML description of the polygonal representation looks something like this:

```

<Rep xsi:type="PolygonalRepType" accuracy="0.2">
  <!--graphic attributes of the polygonal rep-->
  ...
  <!--a set of faces (FaceSetType)-->
  <Faces>
    <!--a face (FaceGPTType)-->
    <Face strips="0 1 4 5 8 9 12 13 14 15"/>
    <!--another face -->
    <Face triangles="12 4 8 " strips="4 12 0 14 2 10 6"/>
    ...
  </Faces>
  <!--the vertex buffer-->
  <VertexBuffer>
    <Positions> -10 0 -10, -10,0,10 ... 10,0,0 </Positions>
    <Normals> -1 0 0, 0,0,1 ... 1 0 0 </Normals>
  </VertexBuffer>
  ...
</Rep>

```

Turn to [section 8.8](#) for a complete example.

8.4.3 The Polyline Primitive

The Polyline primitive (PolylineGPType) is a non-indexed primitive, which means that it includes the definition of its vertices. Because no shading and texturing is applied on edges of a mesh, the definition of a vertex in a Polyline primitive only includes its position (no normal and texture coordinates are required).

How to Describe a Polyline primitive in a Polygonal Representation

Here is an example of edge description in a Polygonal Representation using the Polyline primitive.

```

<?xml version="1.0" encoding="utf-8" ?>
<XMLRepresentation version="1.2" xmlns="http://www.3ds.com/xsd/3DXML"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Root xsi:type="BagRepType" id="2">
    <Rep xsi:type="PolygonalRepType" id="72211" accuracy="0.2" solid="1">
      <Edges>
        <Polyline vertices="-50 -50 50,50 -50 50"/>
        <Polyline vertices="-50 -50 -50,50 -50 -50"/>
        <Polyline vertices="-50 -50 50,-50 -50 -50"/>
        <Polyline vertices="50 -50 50,50 -50 -50"/>
        <Polyline vertices="-50 50 -50,50 50 -50"/>
        <Polyline vertices="-50 -50 -50,-50 50 -50"/>
        <Polyline vertices="50 -50 -50,50 50 -50"/>
        <Polyline vertices="-50 50 50,50 50 50"/>
        <Polyline vertices="50 50 -50,50 50 50"/>
        <Polyline vertices="-50 50 50,-50 50 50"/>
      </Edges>
    </Rep>
  </Root>
</XMLRepresentation>

```

Definition of the vertices of the polyline is done using the **vertices** attribute of the **<PolylineGPType>** object.

8.5 Level of Detail (LOD)

A Level Of Detail is an approximation of the mesh intended to reduce the number of polygonal objects in modeling.

8.5.1 LOD Mechanism

When an object is so far away that it only occupies one pixel, there is very little use in modeling the object in great detail. Conversely, objects that are so close as to occupy the entire (or perhaps more) field of view need to be modeled with significant detail in the areas occupying the current field of view. A high level of detail is not necessary for the entire model because some portions may be obscured by a visible piece of the model or be far enough away to make the detail meaningless. The purpose of the LOD mechanism is to adjust the polygonal representation of an object to the

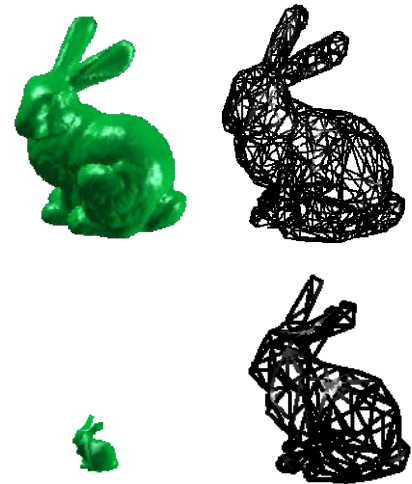
viewer distance. For information, if a mesh in LOD 2 can reveal details of 70m, a mesh in LOD 3 will reveal details of 35m. When increasing the value of the LOD by 1, the level of definition is multiplied by two (but the size of the file increases fourfold).

8.5.2 LOD Accuracy

LOD accuracy is the maximum distance between the set of triangles defined for the LOD and the initial geometry.

How to Specify a LOD

For a body made up of surfaces, specifying a LOD consists in declaring the restricted set of faces to be represented in the polygonal representation. Use the `<PolygonalLOD>` object as described in the example in [section 8.8](#). The set of faces to be represented is declared by the `<Faces>` tag.

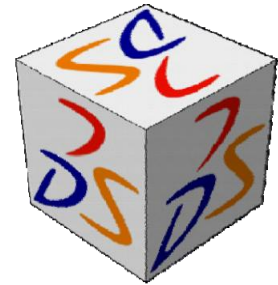


Top : the rabbit is near the observer, it occupies a large amount of pixels on the screen, a precise LOD is needed in order to have a nice rendering.

Bottom : the rabbit is far from the eye, only a few pixels are used on the screen, a rough LOD is enough to render the rabbit.

8.6 Example of a Polygonal Representation with a Texture

Here is an example of a mesh with a texture (a cube with the Dassault Systemes logo). See Material chapter for more informations.



```
<XMLRepresentation version="1.2" xmlns="http://www.3ds.com/xsd/3DXML"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Root xsi:type="BagRepType" id="2">
    <Rep xsi:type="BagRepType" id="3">
      <Rep xsi:type="BagRepType" id="33">
        <Rep xsi:type="PolygonalRepType"
          id="1010" accuracy="0.2" solid="1">
          <SurfaceAttributes>
            <MaterialApplication mappingChannel="0">
              <MaterialId id="urn:3DXML:CATMaterialRef.3dxml#3"/>
            </MaterialApplication>
          </SurfaceAttributes>
          <Faces>
            <Face strips="6 0 9 3,1 12 4 15,14 20 17 23,19 8 22 11,13 2 18 7,5 16 10 21">
              <SurfaceAttributes>
                <Color xsi:type="RGBAColorType"
                  red="0.823529" green="0.823529" blue="1" alpha="1"/>
              </SurfaceAttributes>
            </Face>
          </Faces>
          <Edges>
            <LineAttributes lineType="SOLID" thickness="2">
              <Color xsi:type="RGBAColorType" red="0" green="0" blue="0" alpha="1"/>
            </LineAttributes>
            <Polyline vertices="-50 50 -50,-50 50 50"/>
            <Polyline vertices="-50 -50 -50,-50 -50 50"/>
            <Polyline vertices="-50 50 -50,-50 50 -50"/>
            <Polyline vertices="-50 50 50,-50 50 50"/>
            <Polyline vertices="50 -50 -50,50 -50 50"/>
            <Polyline vertices="-50 -50 -50,50 -50 -50"/>
            <Polyline vertices="-50 -50 50,50 -50 50"/>
            <Polyline vertices="50 50 -50,50 50 50"/>
            <Polyline vertices="50 -50 -50,50 50 -50"/>
            <Polyline vertices="50 -50 50,50 50 50"/>
            <Polyline vertices="50 50 -50,50 50 -50"/>
            <Polyline vertices="50 50 50,-50 50 50"/>
          </Edges>
          <VertexBuffer>
            <Positions>-50 -50 -50,-50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50
            50,-50 50 -50,-50 50 -50,-50 50 -50,-50 50 50,-50 50 50,-50 50 50,-50 50 50,-50
            50,50 -50 50,50 50 -50,50 50 -50,50 50 -50,50 50 50,50 50 50,50 50 50</Positions>
            <Normals>-1 0 0,0 -1 0 0 0 -1,-1 0 0,0 -1 0 0,0 -1 0 0,0 0 -1,0 1 0,-1 0 0,0 0
            1,0 1 0,0 -1 0,0 0 -1,1 0 0,0 -1 0 0 0 1,1 0 0,0 0 -1,0 1 0,1 0 0,0 0 1,0 1 0,1 0 0</Normals>
            <TextureCoordinates dimension="2D" channel="0">1 0 0 0,1 0 1 1,0 1 0 0,0
            0,1 1,1 0 0 1,0 1 1,1 0 0 0 0 0 1,1 1 0 0 1,0 1 0 0 1,1 1 1 1</TextureCoordinates>
          </VertexBuffer>
        </Rep>
      </Rep>
    </Rep>
  </Root>
</XMLRepresentation>
```

8.7 Example of Multiple Textures Applications

Here is an example of a mesh with two textures. See Material chapter for more informations.



```
<XMLRepresentation version="1.2" xmlns="http://www.w3.org/xsd/3DXML"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <Root xsi:type="BagRepType" id="2">
        <Rep xsi:type="BagRepType" id="3">
            <Rep xsi:type="BagRepType" id="33">
                <Rep xsi:type="PolygonalRepType"
                    id="1010" accuracy="0.2" solid="1">

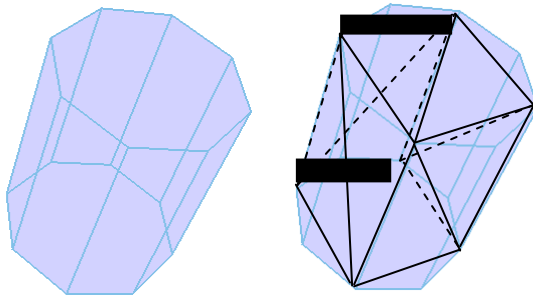
                        <SurfaceAttributes>
                            <MaterialApplication mappingChannel="0">
                                <MaterialId id="urn:3DXML:CATMaterialRef.3dxml#3"/>
                            </MaterialApplication>
                            <MaterialApplication mappingChannel="1"
blendType="ALPHA_TRANSPARENCY">
                                <MaterialId id="urn:3DXML:CATMaterialRef.3dxml#7"/>
                            </MaterialApplication>
                        </SurfaceAttributes>
                        <Faces>
                            <Face strips="6 0 9 3,1 12 4 15,14 20 17 23,19 8 22 11,13 2 18 7,5 16 10 21">
                                <SurfaceAttributes>
                                    <Color xsi:type="RGBAColorType"
                                        red="0.823529" green="0.823529" blue="1" alpha="1"/>
                                </SurfaceAttributes>
                            </Face>
                        </Faces>
                        <Edges>
                            <LineAttributes lineType="SOLID" thickness="2">
                                <Color xsi:type="RGBAColorType" red="0" green="0" blue="0" alpha="1"/>
                            </LineAttributes>
                            <Polyline vertices="-50 50 -50,-50 50 50"/>
                            <Polyline vertices="-50 -50 -50,-50 -50 50"/>
                            <Polyline vertices="-50 50 -50,-50 -50 -50"/>
                            <Polyline vertices="-50 50 50,-50 -50 50"/>
                            <Polyline vertices="50 -50 -50,50 -50 50"/>
                            <Polyline vertices="50 -50 -50,50 -50 -50"/>
                            <Polyline vertices="50 -50 50,50 -50 50"/>
                            <Polyline vertices="50 50 -50,50 50 50"/>
                            <Polyline vertices="50 -50 -50,50 50 -50"/>
                            <Polyline vertices="50 50 -50,50 50 50"/>
                            <Polyline vertices="50 -50 -50,50 50 -50"/>
                            <Polyline vertices="50 50 -50,-50 50 -50"/>
                            <Polyline vertices="50 50 50,-50 50 50"/>
                        </Edges>
                        <VertexBuffer>
                            <Positions>-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,-50 -50 -50,</Positions>
                            <Normals>-1 0 0,0 -1 0 0,0 -1,-1 0 0,0 -1 0 0,0 1,-1 0 0,0 0 -1,0 1 0,-1 0 0,0 0</Normals>
                            <TextureCoordinates dimension="2D" channel="0">1 0,0 1,0 1,1 0,1 0,0</TextureCoordinates>
                            <TextureCoordinates dimension="2D" channel="1">1 0,0 1,0 1,1 0,1 0,0</TextureCoordinates>
                        </VertexBuffer>
                    </Rep>
                </Rep>
            </Root>
        </XMLRepresentation>
```

A photograph of a weathered wooden crate or box. The surface of the wood is decorated with various colorful markings, including what appears to be stylized letters like 'B' and 'V' in blue, orange, and red paint or ink.

8.8 Example of a Polygonal Representation with LOD

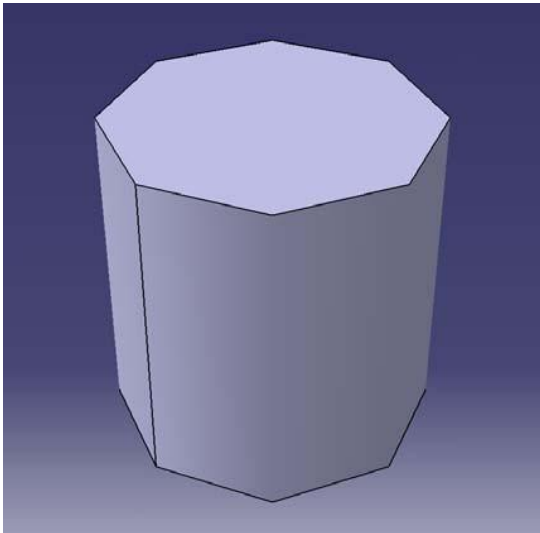
You will find below a complete example of a 3D XML polygonal representation which describes a cylinder. The cylinder is made of three faces and six polylines. It includes a level of detail in which the three faces have been merged and the number of triangles has been reduced.

The LOD described in this example is made of one face with a restricted number of vertices compared to the original mesh.



This example reduces to a minimum the number of strips and fans representing the cylinder. By so doing, the cylinder can be viewed as a rectangular prism.

The 3D XML model corresponding to this sample looks like this:



Following is the 3DXML Mesh content for above sample:

```
<?xml version="1.0" encoding="utf-8" ?>
<XMLRepresentation version="1.2" xmlns="http://www.3ds.com/xsd/3DXML" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xlink="http://www.w3.org/1999/xlink">
  <Root xsi:type="BagRepType">
    <!--the polygonal rep containing the cylinder definition-->
    <Rep xsi:type="PolygonalRepType" accuracy="0.2">
      <!--graphic attributes of the polygonal rep-->
      <SurfaceAttributes>
        <Color xsi:type="RGBAColorType" red="0.6" green="0.6" blue="0.7"/>
      </SurfaceAttributes>
      <LineAttributes thickness="0.35" lineType="DASHED">
        <Color xsi:type="RGBAColorType" red="0" green="0" blue="0"/>
      </LineAttributes>

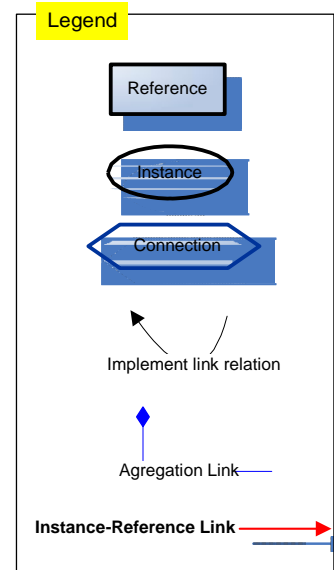
      <!--a level of detail with few triangles. The LOD is made of one face only -->
      <PolygonalLOD accuracy="0.5">
        <Faces>
          <Face fans="17 25 31 23, 16 24 30 22" strips="0 1 8 9 14 15 6 7 0 1"/>
        </Faces>
      </PolygonalLOD>

      <Faces>
        <!--middle face-->
        <Face strips="0 1 4 5 8 9 12 13 14 15 10 11 6 7 2 3 0 1"/>
        <!--bottom face-->
        <Face triangles="28 20 24 " strips="20 28 16 30 18 26 22 "/>
        <!--top face-->
        <Face triangles="25 21 29" strips="29 21 31 17 27 19 23 "/>
      </Faces>
      <!--edges of the cylinder-->
      <Edges>
        <Polyline vertices="-10 1.22461e-015 -10,-7.07107 7.07107 -10,6.12303e-016 10 -10,7.07107
7.07107 -10,10 0 -10"/>
        <Polyline vertices="-10 1.22461e-015 10,-7.07107 7.07107 10,6.12303e-016 10 10,7.07107
7.07107 10,10 0 10"/>
        <Polyline vertices="10 0 -10,7.07107 -7.07107 -10,-1.83691e-015 -10 -10,-7.07107 -7.07107 -
10,-10 1.22461e-015 -10"/>
        <Polyline vertices="10 0 10,7.07107 -7.07107 10,-1.83691e-015 -10 10,-7.07107 -7.07107 10,-
10 1.22461e-015 10"/>
        <Polyline vertices="10 0 10,10 0 -10"/>
        <Polyline vertices="-10 1.22461e-015 -10,-10 1.22461e-015 10"/>
      </Edges>
      <!--a vertex buffer containing all vertices of the cylinder-->
      <VertexBuffer>
        <Positions>-10 0 -10, -10 0 10, -7.07107 -7.07107 -10, -7.07107 -7.07107 10, -7.07107 7.07107
-10, -7.07107 7.07107 10, 0 -10 -10, 0 -10 10, 0 10 -10, 0 10 10, 7.07107 -7.07107 -10, 7.07107 -7.07107 10, 7.07107
7.07107 -10, 7.07107 7.07107 10, 10 0 -10, 10 0 10, -10 0 -10, -10 0 10, -7.07107 -7.07107 -10, -7.07107 -7.07107 10, -
7.07107 7.07107 -10, -7.07107 7.07107 10, 0 -10 -10, 0 -10 10, 0 -10 10, 0 10 -10, 0 10 10, 7.07107 -7.07107 -10, 7.07107 -7.07107
10, 7.07107 7.07107 -10, 7.07107 7.07107 10, 10 0 -10, 10 0 10</Positions>
        <Normals>-1 0 0, -1 0 0, -0.707083 -0.70713 0, -0.707083 -0.70713 0, -0.707083 0.70713 0, -
0.707083 0.70713 0, 0 -1 0, 0 -1 0, 0 1 0, 0 1 0, 0.707083 -0.70713 0, 0.707083 -0.70713 0, 0.707083 0.70713 0, 0.707083
0.70713 0, 1 0 0, 1 0 0, 0 0 -1, 0 0 1, 0 0 -1, 0 0 1, 0 0 -1, 0 0 1, 0 0 -1, 0 0 1, 0 0 -1, 0 0 1, 0 0 -1, 0 0 1, 0 0 -1,
0 0 1</Normals>
      </VertexBuffer>
    </Rep>
  </Root>
</XMLRepresentation>
```

9. PPR

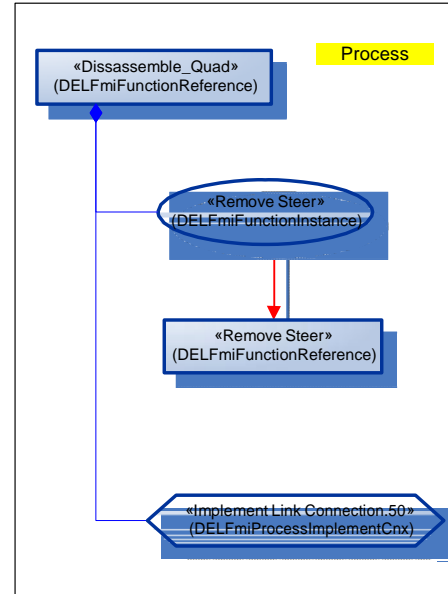
PPR is an acronym for Product, Process and Resource. The PPR Context is an extended Product reference that provides an integrated Manufacturing view of a consistent set of Products and Resources for a given set of manufacturing Processes.

Product, Process and Resource data are described in independent instance/reference trees in 3DXML. Product Structure data has been explained in detail in chapter 3. Process and Resource structures are described in the following sections of this chapter. We will use the following convention to explain the details of PPR

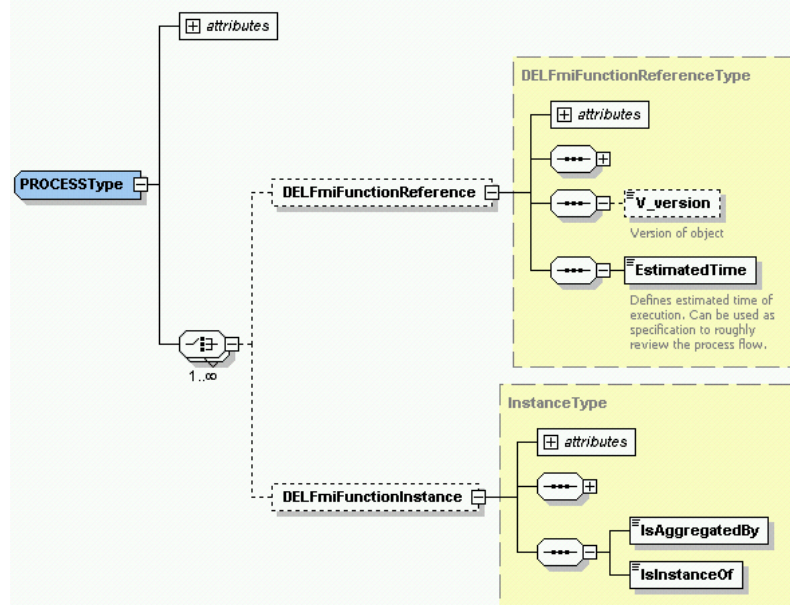


9.1 Process

The Process Structure is a instance/reference directed Acyclic Graph like the Product Structure. Process Reference gives a definition of a type of transformation for a Product Instance. In other words, a Process Instance implements a Product Instance. This "Implement" semantic is modeled using a Connection from the Process Instance to the Product Instance.



The <PROCESS> element, gathers a number of <DELFmiFunctionReference> (aka Process Reference), and <DELFmiFunctionInstance> (aka Process Instance) elements.



```

<PROCESS root="19">
  <DELFmiFunctionReference xsi:type="Reference3DType" id="21" name="MyActivity.1">
    <PLM_ExternalID>MyActivity.1</PLM_ExternalID>
    <PLMAttribute value="P1" name="V_ProcessType"/>
    <PLMAttribute value="MyActivityReference" name="V_description"/>
    <V_EstimatedTime>0</V_EstimatedTime>
    <Extension xsi:type="DELAsmGenericFunctionRefExtensionType"/>
  </DELFmiFunctionReference>
  <DELFmiFunctionInstance xsi:type="Instance3DType" id="22" name="MyActivity.1">
    <PLM_ExternalID>MyActivity.1</PLM_ExternalID>
    <PLMAttribute value="MyActivityInstance.1" name="V_description"/>
    <PLMAttribute value="A1" name="V_ProcessCategory"/>
    <IsAggregatedBy>19</IsAggregatedBy>
    <IsInstanceOf>urn:3DXML:Reference:loc:21</IsInstanceOf>
  </DELFmiFunctionInstance>
</PROCESS>

```

Connection:

```

<DELFmiFunctionalModelImplementCnx root="19">
  <DELFmiProcessImplementCnx xsi:type="" id="23" name="Implement Link Connection.23">
    <V_discipline>ProcessImplementLink</V_discipline>
    <PLM_ExternalID>Implement Link Connection.23</PLM_ExternalID>
    <IsAggregatedBy>urn:3DXML:Reference:ext:R17Simple.3dxml#19</IsAggregatedBy>
    <PLMRelation>
      <C_Role>PLM_ImplementLink_Source</C_Role>
      <C_Semantics>Reference5</C_Semantics>
      <Ids>
        <id>urn:3DXML:Reference:ext:R17Simple.3dxml#22</id>
      </Ids>
    </PLMRelation>
    <PLMRelation>
      <C_Role>PLM_ImplementLink_Target</C_Role>
      <C_Semantics>Reference3</C_Semantics>
      <Ids>
        <id>urn:3DXML:Reference:ext:R17Simple.3dxml#2</id>
        <id>urn:3DXML:Reference:ext:R17Simple.3dxml#4</id>
        <id>urn:3DXML:Reference:ext:R17Simple.3dxml#8</id>
      </Ids>
    </PLMRelation>
  </DELFmiProcessImplementCnx>
</DELFmiFunctionalModelImplementCnx>

```

The Process instance that realizes the transformation is identified thanks to following tags. Values are fixed.

```

<C_Role>PLM_ImplementLink_Source</C_Role>
<C_Semantics>Reference5</C_Semantics>

```

The product instance that realizes the transformation is identified thanks to following tags. Values are fixed.

```

<C_Role>PLM_ImplementLink_Target</C_Role>
<C_Semantics>Reference3</C_Semantics>

```

Resource is based on an extended product, and thus uses Product structure (Reference, Instance and Extension).

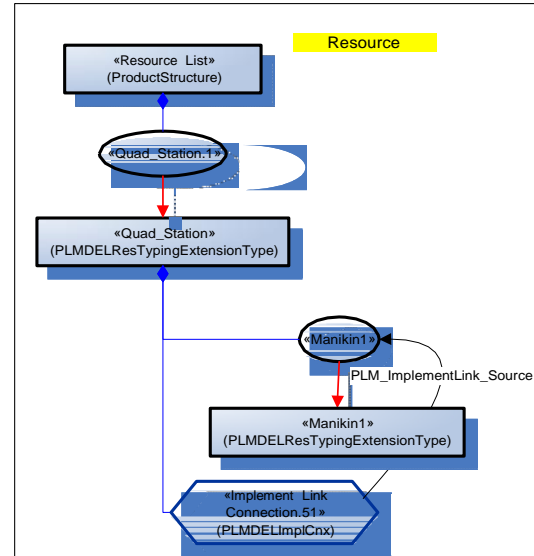
9.2 Resource

A Resource is a Reference Product used in a manufacturing context

- Resources are also modeled in 3DXML using instance/reference Directed Acyclic graph
- Resource related Extensions can be assigned to extend the product reference with attributes supporting the usage of the product in the manufacturing context.

Examples of Resource are Robot, Tool, Machine, Welding Gun, Production line, Human.

Resource implements Process. One Resource can implement multiple Processes, and One Process can be implemented by several Resources (through a PLM Connection).



An extension `<PLMDELResTypingExtensionType>` on the Product Reference defines a Resource, with a mandatory element `<MainType>` constrained by an enumerated numbers of values that describes precisely the kind of resource, for example a Robot, a Tool Device.

For connection between a resource instance and a process instance, the `<DELRmiResourceModelImplCnx>` gathers a number of `<PLMDELImpnCnx>`, including a `<PLM_ImplementLink_Target>` that is a Process and a `<PLM_ImplementLink_Source>` that is a Resource.


```

<ProductStructure root="1">
  <Reference3D xsi:type="Reference3DType" id="1" name="PPR">
    <Extension xsi:type="PLMDELPPRContextExtensionType"/>
  </Reference3D>
  <Instance3D xsi:type="Instance3DType" id="28" name="Resource List.1">
    <IsAggregatedBy>1</IsAggregatedBy>
    <IsInstanceOf>29</IsInstanceOf>
    <RelativeMatrix>1 0 0 0 1 0 0 0 1 0 0 0</RelativeMatrix>
    <Extension xsi:type="PLMDELPPRResourceExtensionType"/>
  </Instance3D>
  <Reference3D xsi:type="Reference3DType" id="29" name="Robot1">
    <Extension xsi:type="PLMDELResTypingExtensionType">
      <V_MainType>1</V_MainType>
    </Extension>
  </Reference3D>
  <Instance3D xsi:type="Instance3DType" id="30" name="Quad_Station.1">
    <IsAggregatedBy>29</IsAggregatedBy>
    <IsInstanceOf>31</IsInstanceOf>
    <RelativeMatrix>1 0 0 0 1 0 0 0 1 0 0 0</RelativeMatrix>
  </Instance3D>
  <Reference3D xsi:type="Reference3DType" id="31" name="Quad_Station">
    <Extension xsi:type="PLMDELResTypingExtensionType">
      <V_MainType>1</V_MainType>
    </Extension>
  </Reference3D>
</ProductStructure>

```

Resource extension on Product is PLMDELResTypingExtensionType, and the family of resource is set by the mandatory attribute V_ MainType

Connection Resource to Process:

```
<DELRmiResourceModellImplCnx root="1">
  <PLMDELLImplCnx xsi:type="" id="51" name="Implement Link Connection.51">
    <V_discipline>DELLImplementLinksDiscipline</V_discipline>
    <PLM_ExternalID>Implement Link Connection.51</PLM_ExternalID>
    <PLMAttribute value="PPR" name="V_description"/>
    <IsAggregatedBy>urn:3DXML:Reference:ext:Dissassemble_Quad.3dxml#31</IsAggregatedBy>
    <PLMRelation>
      <C_Role>PLM_ImplementLink_Target</C_Role>
      <C_Semantics>Reference3</C_Semantics>
      <Ids>
        <id>urn:3DXML:Reference:ext:Dissassemble_Quad.3dxml#46</id>
        <id>urn:3DXML:Reference:ext:Dissassemble_Quad.3dxml#49</id>
      </Ids>
    </PLMRelation>
    <PLMRelation>
      <C_Role>PLM_ImplementLink_Source</C_Role>
      <C_Semantics>Reference5</C_Semantics>
      <Ids>
        <id>urn:3DXML:Reference:ext:Dissassemble_Quad.3dxml#28</id>
        <id>urn:3DXML:Reference:ext:Dissassemble_Quad.3dxml#30</id>
        <id>urn:3DXML:Reference:ext:Dissassemble_Quad.3dxml#32</id>
      </Ids>
    </PLMRelation>
  </PLMDELLImplCnx>
</DELRmiResourceModellImplCnx>
```

The resource instance that realizes the transformation is identified thanks to following tags.
Values are fixed.

```
<C_Role>PLM_ImplementLink_Source</C_Role>
<C_Semantics>Reference5</C_Semantics>
```

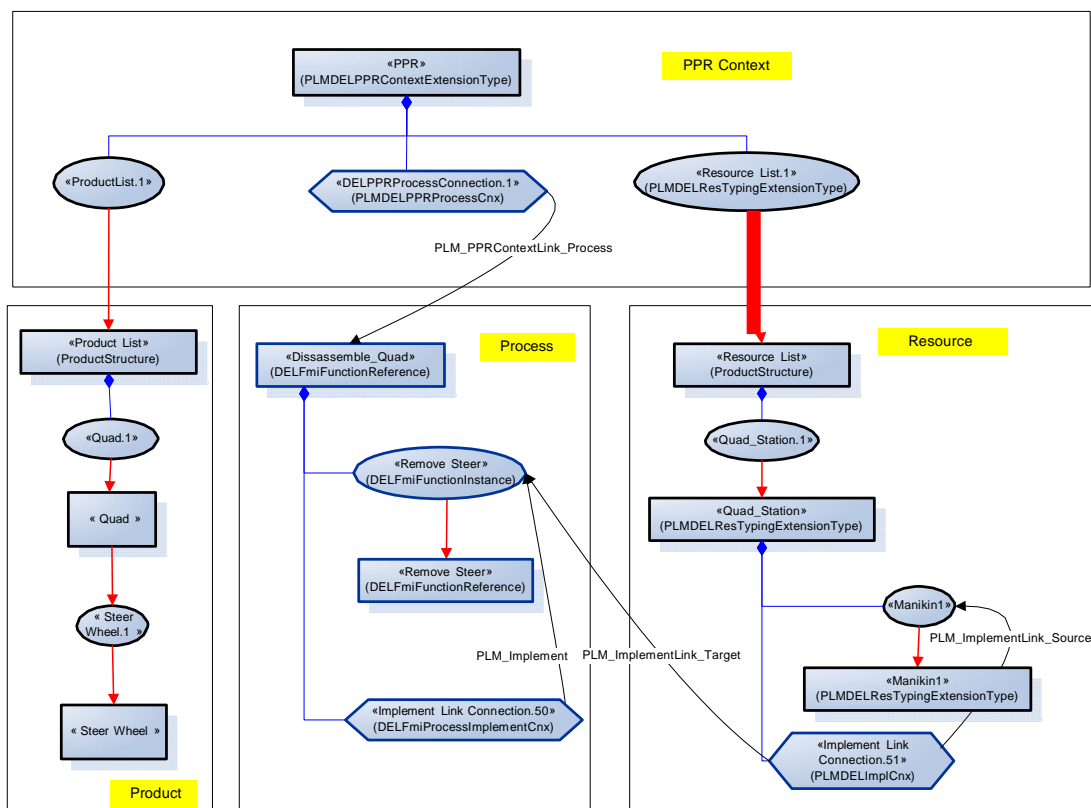
The process instance that realizes the transformation is identified thanks to following tags.
Values are fixed.

```
<C_Role>PLM_ImplementLink_Target</C_Role>
<C_Semantics>Reference3</C_Semantics>
```

9.3 PPR context revisited

PPR Context gathers Products and Resources using the standard product structure instantiation (refer to Product Structure section). The Resources are a specialization of Products and the extension <PLMDELResTypingExtensionType> identifies this specialization.

Processes are associated to the PPR Context using a PLM Connection <DELPPRProcessConnection>. This connection is aggregated in the PPR Context points to one Process Reference (one connection per Process linked).



9.4 Manikin

Manikins are used in the manufacturing world as well as in the design world for ergonomics analyses and for simulation of processes involving operators and workers.

Manikin can be modified with dedicated CATIA, ENOVIA or DELMIA Applications :

- anthropometry (size of segments, population, gender)
- posture (using Inverse Kinematics tools, Forward Kinematics, Place mechanism, standard postures, catalogs of postures).
- Analysis (ergonomic analysis can be applied and reused)
- Task (in dedicated DELMIA application, complete scenarios for human exercising products or realizing operations, can be described and simulated)

This manikin object is to be used everywhere human needs to be considered for validating a design (car cockpit for example) or realizing operation (in a manufacturing environment).

A manikin is a resource whose main type is Worker. It also has its own extension :

<PLMResourceManikinRefExtensionType>, with the following attributes :

- V_Population : the population
- V_Gender : the gender
- V_Weight : the weight percentile
- V_Stature : the stature percentile

Two references are aggregated by the manikin product reference :

- A tessellated representation that can be visualized by any viewer ;
- A technological representation that contains the detailed specifications of the manikin. The content of this representation cannot be read by the user.

Following is a sample of the 3dxml description of a manikin:

```

<ProductStructure root="1">
  <Reference3D xsi:type="Reference3DType" id="1" name="Product1"/>
  <Reference3D xsi:type="Reference3DType" id="2" name="Manikin1">
    <Extension xsi:type="PLMDELResTypingExtensionType">
      <V_MainType>Worker</V_MainType>
    </Extension>
    <Extension xsi:type="PLMResourceManikinRefExtensionType">
      <V_Population>American</V_Population>
      <V_Gender>1</V_Gender>
      <V_Weight>50</V_Weight>
      <V_Stature>50</V_Stature>
    </Extension>
  </Reference3D>
  <Instance3D xsi:type="Instance3DType" id="3" name="Manikin1">
    <IsAggregatedBy>1</IsAggregatedBy>
    <IsInstanceOf>2</IsInstanceOf>
    <RelativeMatrix>1 0 0 0 1 0 0 0 1 0 0 0</RelativeMatrix>
  </Instance3D>
  <ReferenceRep xsi:type="ReferenceRepType" id="4"
    name="Manikin1_Result_ReferenceRep" format="TESSELLATED" version="1.2"
    associatedFile="urn:3DXML:Manikin1_Result.3DRep">
    <PLM_ExternalID>Manikin1_Result_ReferenceRep_ReferenceRep</PLM_ExternalID>
    <V_discipline>Design</V_discipline>
    <V_usage>3DShape</V_usage>
    <V_nature>1</V_nature>
  </ReferenceRep>
  <InstanceRep xsi:type="InstanceRepType" id="5" name="Manikin1_Result_InstanceRep">
    <IsAggregatedBy>2</IsAggregatedBy>
    <IsInstanceOf>4</IsInstanceOf>
  </InstanceRep>
  <ReferenceRep xsi:type="ReferenceRepType" id="6"
    name="Human_Physical_Manikin1_Spec" format="TECHREP" version="1.0"
    associatedFile="urn:3DXML:Manikin1_Spec.3DRep">
    <PLM_ExternalID>Human_Physical_Manikin1_Spec</PLM_ExternalID>
    <V_discipline>Human</V_discipline>
    <V_nature>1</V_nature>
  </ReferenceRep>
  <InstanceRep xsi:type="InstanceRepType" id="7"
name="Human_Physical_Manikin1_InstanceRep">
    <IsAggregatedBy>2</IsAggregatedBy>
    <IsInstanceOf>6</IsInstanceOf>
  </InstanceRep>
</ProductStructure>

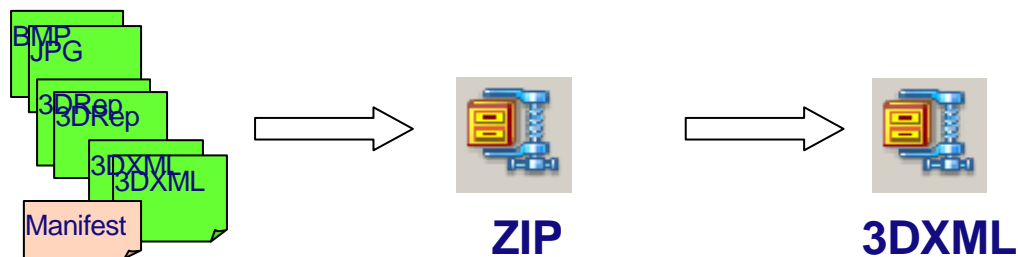
```

10. 3DXML Archive

The 3D XML file is a collection of different component that make up the various containers described previously in this document. This collection is managed as an integral unit by combining them into an archive. This section briefly describes the organization of the 3DXML archive.

10.1 Packaging

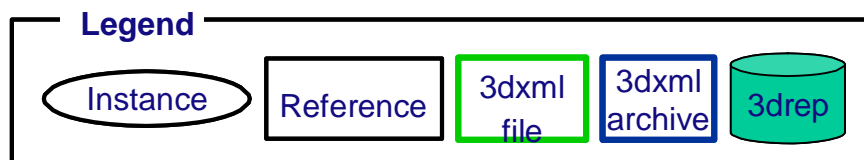
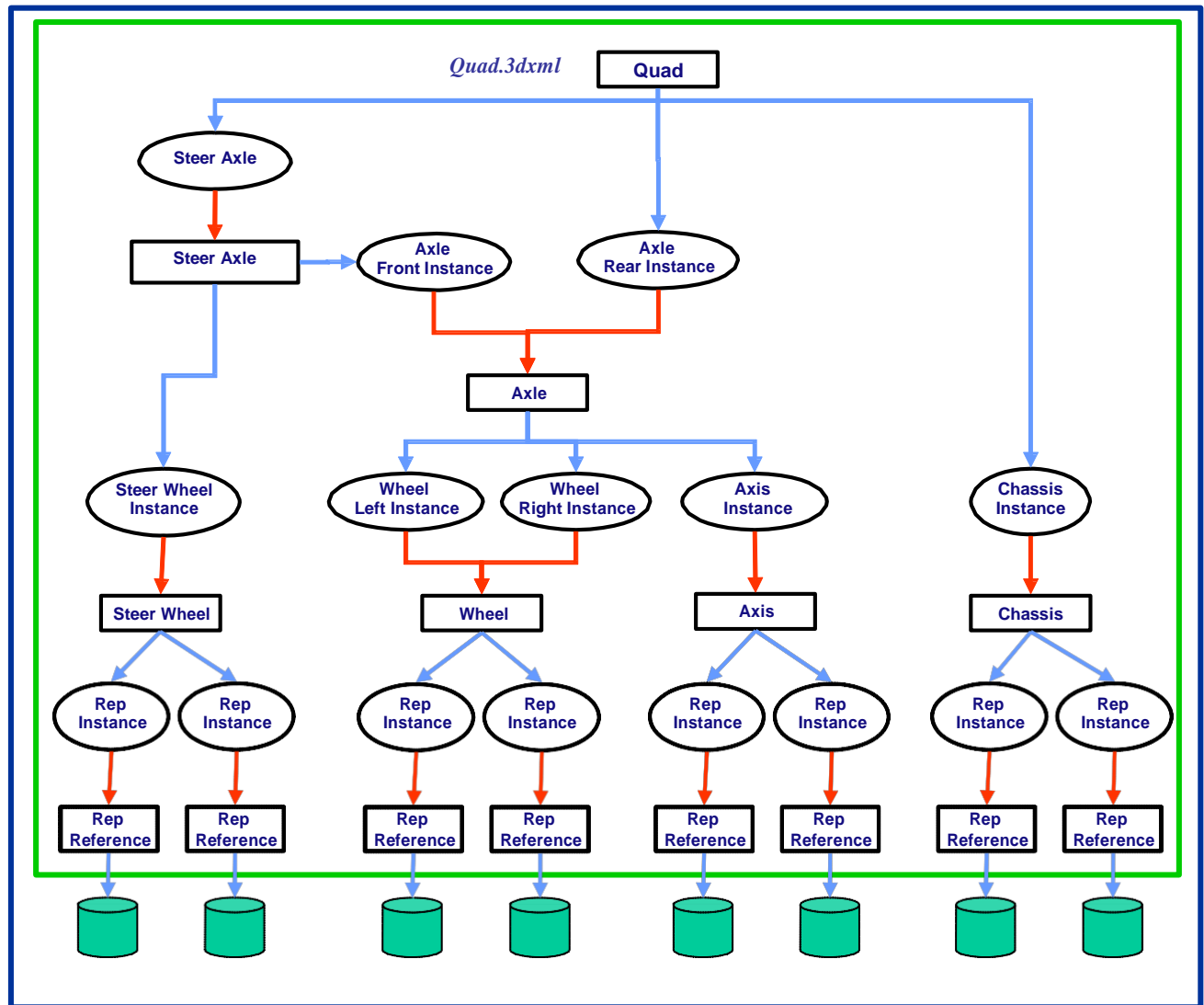
A 3DXML package is made up of various containers described previously in this document. For example, the following containers are packaged in different files inside a 3DXML package: Product Structure, Geometric representations, Images, Material, industry specific (applicative) data are each packaged in separate files within a 3DXML archive.



The ZIP compression used provides the necessary bundling of the 3D XML components as well as much needed compression for transmission of this data over network.

The Manifest.xml file within the archive identifies the starting point (file containing the root assembly). Applications needing to access and/or create 3dxml ZIP archives can use freely available ZIP toolkits.

Following is a decomposition of the sample accompanied with this user guide. Here you can see how the different containers and data are organized in the 3DXML archive.



10.2 Specifying Links

Following types of links are possible:

1. Local Links – for example an instance pointing to a reference within the same file.
2. External link – for example a ReferenceRep pointing to its physical representation data in another file, but inside the same archive.

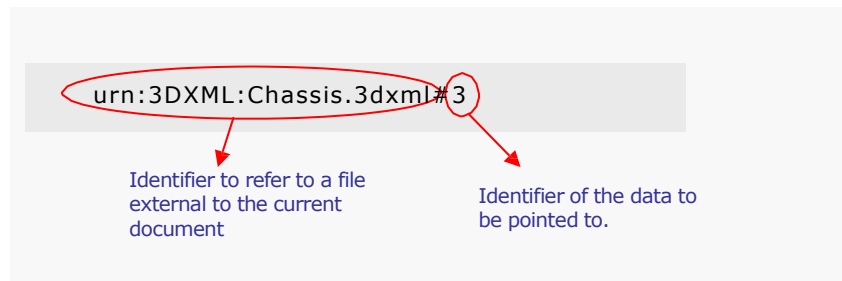
10.2.1 Local Links

The local links are to be used when the object to be pointed to is located within the current file. Local link is simply the identifier (**id**) of the object to be pointed to.

```
<ProductStructure root="1">
...
  <Reference3D xsi:type="Reference3DType" id="5" name="Quad"/>
  <Instance3D xsi:type="Instance3DType" id="6" name="Chassis.1">
    <IsAggregatedBy id="5"/>
    <IsInstanceOf id="7"/>
    <RelativeMatrix>0 1 0 1 0 1 0 1 0 1</RelativeMatrix>
  </Instance3D>
.....
```

10.2.2 External Links

The external links are to be used when the object to be pointed to is not located within the current file. The fourth string is the path to the resource where the data to be pointed to is located. The **id** of the data to be pointed to in the external resource is specified by the **#identifier**.



Following is an example of an external link. In this example, the 3D representation data exists in a dedicated file in the 3DXML archive and the ReferenceRep is pointing to this representation file.

```
<Reference3D xsi:type="Reference3DType" id="7" name="Chassis"/>
<ReferenceRep xsi:type="ReferenceRepType" id="8" name="Chassis_ReferenceRep" format="UVR"
  version="1.0" associatedFile="urn:3DXML:Chassis.3DRep"/>
```

11. Versioning & Extensibility

This chapter explains how the 3D XML schema will evolve and also how it can be extended by third parties to suit their needs. We treat these two topics together since they are related; as you will see below.

Versioning deals with the mechanisms which controls or limits backward and forward compatibility.

Extensibility deals with the mechanisms which makes it possible to extend the format by third parties. It allows the format to evolve without requiring central control of the format (it may harm interoperability because not all clients will support the same extensions).

As languages evolve, it becomes possible to speak of both backwards and forwards compatibility. There are two aspects of compatibility that are called out: software compatibility and schema compatibility. While it is often the case that they are directly related, sometimes they are not. We address below the aspect of software compatibility:

A language change is **backwards compatible** if newer processors can process all instances of the old language. Backwards compatible changes allow applications to behave properly if they receive an "older" version of the language.

A language change is **forwards compatible** if older processors can process all instances of the newer language. Forwards compatible changes allow existing applications to behave properly if they receive a "newer" version of the language.

In broad terms, backwards compatibility means that newer applications can continue to use existing files, and forwards compatibility means that existing applications can use newer files.

Our approach to versioning is to use **major.minor** version numbers, i.e. 1.0, 1.1, 2.0, etc. The definition of a major change is that it is incompatible, and the definition of a minor change is that it is forward and backward compatible.

In practice, a **minor** change represents relatively small changes to the standard. To enforce forward compatibility, it should consist only in new **optional** attributes or elements. Applications should use the **must ignore** rule: they must ignore any XML attributes or elements that they do not recognize.

A **major** change can be one of the following:

- f* A required information item is added.
- f* The semantics of existing information are changed.
- f* The maximum number of allowable items is reduced.

In general, a major version consolidates and incorporates all prior minor changes.

The consequences of this choice are:

- f* An application based on schema $x.0$ can process any instance of schema $x.y$, by simply ignoring attributes or elements that they do not recognize.
- f* An application based on schema $x.0$ can not process an instance based on schema $x'.0$.
- f* An application based on schema $x.y$ can only validate instances based on schema $x.y'$ with y' lower than y .

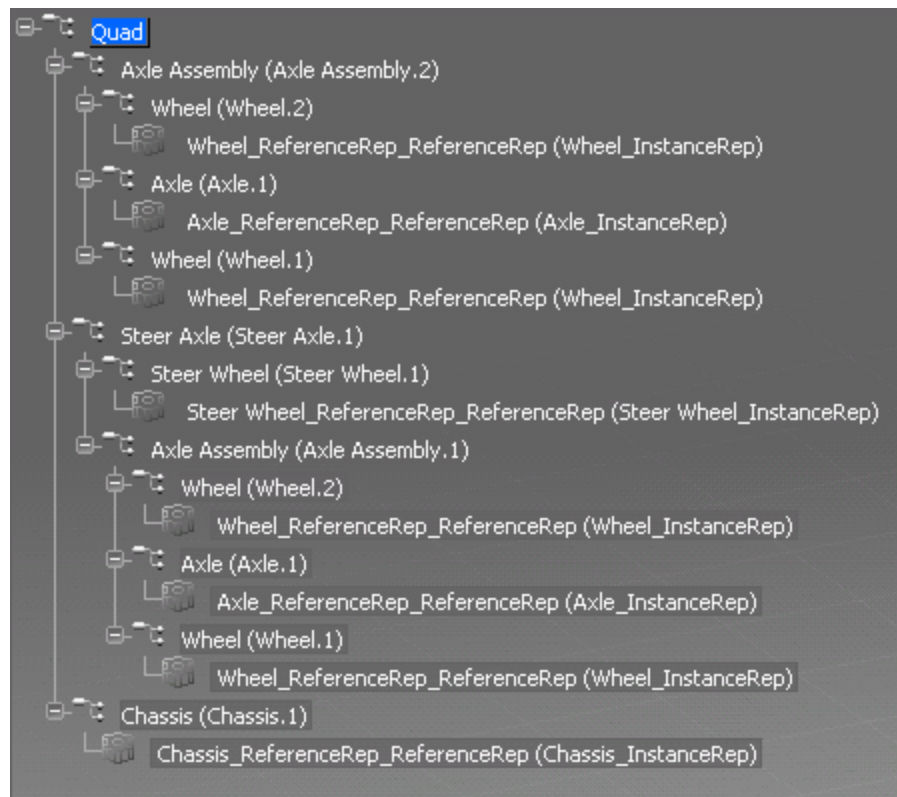
Third-parties can then extend the original schema by declaring their own extensions. The extension supported by 3DXML is addressed in [section 1.2.6](#).

Task Index

| | |
|---|----|
| How to Create References and Instances | 26 |
| How to Specify a Representation | 26 |
| How to Specify Surface Attributes | 29 |
| How to Specify Line Attributes | 30 |
| How to Specify Point Attributes | 31 |
| How to Specify General Attributes | 31 |
| How to use Graphic Properties with Default View | 32 |
| How to use Graphic Properties with Mesh | 33 |
| How to Specify a Projection Viewpoint..... | 55 |
| How to Specify a Perspective Viewpoint..... | 55 |
| How to Specify a Parallel Viewpoint | 56 |
| How to Specify a Customized 3D Viewpoint..... | 57 |
| How to Describe Face Primitives in a Polygonal Representation..... | 62 |
| How to Describe a Polyline primitive in a Polygonal Representation..... | 63 |
| How to Specify a LOD | 64 |

Appendix

Complete Reference/Instance Graph of the Quad Sample



Glossary

A

alpha channel

A portion of each pixel's data that is reserved for transparency information. 32-bit graphics systems contain four channels: three 8-bit channels for red, green, and blue (RGB) and one 8-bit alpha channel.

An alpha value of zero represents full transparency, and a value of 1 represents a fully opaque pixel. Intermediate values indicate partially transparent pixels.

ambient light

Light that has been scattered so much by the environment that its direction is impossible to determine.

C

container

A 3D XML element intended to gather data of similar types or data to be used together.

D

diffuse light

Light that comes from one direction. The diffuse light is bright if it comes squarely down on a surface while it is darker if it barely glances off the surface.

I

indexed primitive

A method of describing a group of 3D primitives. The vertices of the primitives are stored in a vertex list, and each primitive (a triangle or line segment) is described as a set of index values that point to particular vertices in the vertex list.

instance

In the 3D XML context, an object which is defined as an instantiation of a reference and is only meaningful when positioned within a Product Structure.

L

Level of Detail

A mechanism for achieving a high level of performance in a 3D virtual world. It balances the quantity of an object with its quality (detail). As some measure of the distance between the viewer and the object changes, a related change is made in the quantity and quality of the rendering of an object.

R

reference

In the 3D XML context, a standardized object intended to be reused.

RGBA color

A color which is defined by its four components: red, green, blue, and alpha. The RGBA color model is an additive color model in which Red, Green, and Blue light are combined in various ways to create other colors. The alpha channel is used to manage the transparency.

S

specular light

A light that comes from a particular direction and tends to bounce off the surface in a preferred direction.

specular exponent

A property of the specular light which defines the size of the highlight spot. Typical values for this property range from 1 to 500, with normal objects having values in the range 5 to 20.

V

vertex buffer

An object intended to store in high performance memory all vertices of a group of faces and adjacent edges and points. All vertices in a vertex buffer have the same description (coordinates, normal, texture).