

---

---

**Document management — 3D use of  
Product Representation Compact (PRC)  
format —**

**Part 1:  
PRC 10001**

*Gestion de documents — Utilisation en 3D du format compact de  
représentation de produit (PRC) —*

*Partie 1: PRC 10001*





**COPYRIGHT PROTECTED DOCUMENT**

© ISO 2014

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office  
Case postale 56 • CH-1211 Geneva 20  
Tel. + 41 22 749 01 11  
Fax + 41 22 749 09 47  
E-mail [copyright@iso.org](mailto:copyright@iso.org)  
Web [www.iso.org](http://www.iso.org)

Published in Switzerland

| <b>Contents</b>                            | <b>Page</b> |
|--------------------------------------------|-------------|
| <b>Contents .....</b>                      | <b>iii</b>  |
| <b>Foreword.....</b>                       | <b>v</b>    |
| <b>1 Scope.....</b>                        | <b>1</b>    |
| <b>2 Normative references.....</b>         | <b>1</b>    |
| <b>3 Terms and definitions .....</b>       | <b>2</b>    |
| <b>4 Document syntax conventions .....</b> | <b>2</b>    |
| 4.1 Conventions .....                      | 2           |
| 4.2 Example Structure .....                | 2           |
| <b>5 PRC file concepts .....</b>           | <b>3</b>    |
| 5.1 The PRC file .....                     | 3           |
| 5.2 Versioning .....                       | 5           |
| 5.3 Unique identifiers.....                | 6           |
| 5.4 Current data values .....              | 7           |
| 5.5 Userdata .....                         | 7           |
| 5.6 Units.....                             | 8           |
| 5.7 Tolerances .....                       | 8           |
| 5.8 Compressed file sections.....          | 9           |
| 5.9 Compressed geometry .....              | 9           |
| 5.10 Compressed tessellation .....         | 9           |
| <b>6 PRC file contents.....</b>            | <b>9</b>    |
| 6.1 Fileheader .....                       | 9           |
| 6.2 Filestructure .....                    | 11          |
| 6.3 PRC Schema.....                        | 13          |
| <b>7 PRC basic types .....</b>             | <b>13</b>   |
| 7.1 General.....                           | 13          |
| 7.2 Uncompressed types.....                | 14          |
| 7.3 Compressed types.....                  | 15          |
| <b>8 Base entities.....</b>                | <b>21</b>   |
| 8.1 General.....                           | 21          |
| 8.2 Abstract root types .....              | 21          |
| 8.3 Structure and assembly .....           | 25          |
| 8.4 Miscellaneous Data.....                | 45          |
| 8.5 Graphics.....                          | 56          |
| 8.6 Representation items .....             | 72          |
| 8.7 Markup.....                            | 77          |
| 8.8 Tessellation.....                      | 83          |
| 8.9 Topology.....                          | 114         |
| 8.10 Curve.....                            | 150         |
| 8.11 Surface .....                         | 182         |
| 8.12 Mathematical Operator.....            | 209         |
| <b>9 Schema Definition.....</b>            | <b>213</b>  |
| 9.1 General.....                           | 213         |
| 9.2 Enumeration Of Schema Tokens .....     | 214         |

|                       |                                                 |     |
|-----------------------|-------------------------------------------------|-----|
| 9.3                   | Schema Processing .....                         | 216 |
| 9.4                   | Schema Requirements and Examples.....           | 222 |
| 10                    | I/O Algorithms .....                            | 225 |
| 10.1                  | Getnumberofbitsusedtostoreunsignedinteger ..... | 225 |
| 10.2                  | Makeportable32bitsunsigned .....                | 225 |
| 10.3                  | Writebits .....                                 | 225 |
| 10.4                  | Writestring.....                                | 226 |
| 10.5                  | Writefloatasbytes.....                          | 226 |
| 10.6                  | Writecharacterarray.....                        | 227 |
| 10.7                  | Writeshortarray .....                           | 228 |
| 10.8                  | Writecompressedintegerarray .....               | 229 |
| 10.9                  | Writecompressedindicearray .....                | 229 |
| 10.10                 | Writeunsignedinteger .....                      | 230 |
| 10.11                 | Writeinteger .....                              | 230 |
| 10.12                 | Writeintegerwithvariablebitnumber .....         | 230 |
| 10.13                 | Writeunsignedintegerwithvariablebitnumber.....  | 231 |
| 10.14                 | Writedoublewithvariablebitnumber.....           | 231 |
| 10.15                 | Writenumberofbitsthenunsignedinteger .....      | 232 |
| 10.16                 | Writecompressedentitytype .....                 | 232 |
| 10.17                 | Writedouble .....                               | 233 |
| 10.18                 | Procedure For Writedouble .....                 | 270 |
| 11                    | Tessellation Compression Support .....          | 274 |
| 11.1                  | General.....                                    | 274 |
| 11.2                  | Huffman Algorithm.....                          | 275 |
| 11.3                  | Basis Pseudocode.....                           | 277 |
| Annex A (informative) | Example: Triangle .....                         | 281 |
| Annex B (informative) | List of figures and tables .....                | 283 |
| Bibliography          | .....                                           | 284 |

## Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see [www.iso.org/directives](http://www.iso.org/directives)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](http://www.iso.org/patents)).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: [Foreword - Supplementary information](#)

The committee responsible for this document is ISO/TC 171, *Document management applications*, Subcommittee SC 2, *Application issues*.

## Introduction

The data representations in PRC allows 3D design data, typically created in CAD and PLM systems, to be viewed and interrogated by visualization applications and to be integrated into complex documents.

This document specifies a wide range of data forms. The wide range is necessary to:

- Achieve a high fidelity, visually equivalent representation of 3D design data produced by an advanced CAD or PLM system without requiring the original application.
- Allow applications to compute high accuracy product shape measurements.

PRC is intended to complement native or open standard CAD and PLM formats as a compact, concise binary form for visualization and documentation. PRC is not intended as a data format for CAD interoperability or use in factory automation systems, e.g. automated manufacturing and inspection systems, which is addressed by the ISO 10303 standards.

# Document management — 3D use of Product Representation Compact (PRC) format —

## Part 1: PRC 10001

### 1 Scope

This International Standard describes PRC 10001 of a product representation compact (PRC) file format for three dimensional (3D) content data. This format is designed to be included in PDF (ISO 32000) and other similar document formats for the purpose of 3D visualization and exchange. It can be used for creating, viewing, and distributing 3D data in document exchange workflows. It is optimized to store, load, and display various kinds of 3D data, especially that coming from computer aided design (CAD) systems.

This International Standard does not apply to:

- Method of electronic distribution
- Converting CAD system generated datasets to the PRC format
- Specific technical design, user interface, implementation, or operational details of rendering
- Required computer hardware and/or operating systems

### 2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 12651:1999, *Electronic imaging — Vocabulary*

ISO 24517-1:2008, *Document management — Engineering document format using PDF — Part 1: Use of PDF 1.6 (PDF/E-1)*

ISO 32000, *Document management — Portable document format*

IEEE 754, *Floating-Point Arithmetic*

*The OpenGL Graphics System, A Specification, Version 4.1 (Core Profile), July 25, 2010*<sup>1</sup>

---

<sup>1</sup> Available at <http://www.opengl.org/registry/doc/glspec41.core.20100725.pdf>

### 3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 32000-1, ISO 24517-1 and ISO 12651 and the following apply.

#### 3.1

##### **PRC File Writer**

software application which writes a particular PRC file

#### 3.2

##### **PRC File Reader**

software application which reads a particular PRC file

#### 3.3

##### **byte**

group of eight bits processed as a single unit of data

### 4 Document syntax conventions

#### 4.1 Conventions

The following conventions are used within this document to describe data within a PRC File.

**Terms highlighted in bold** within this document signify field names in the description of entity types. Entity types are denoted in *italic*.

A table with three columns is used to describe the data within a contiguous portion of the file.

The first column indicates the name of the field. Field names are not unique and can be considered to have a scope limited to the data class.

The second column describes the type of the data. This might be

- A simple data type such as a Boolean, UnsignedInteger, or Double
- A simple class of data such as PRC\_TYPE\_TOPO\_Body or PtrTopology where the name of the class is used to define the data stored for that class
- An Array<data class>[<size>] which indicates an array of data of the specified class. An array has <size> elements. Elements of an array are referenced beginning at 0.

The third column indicates if the field is required or optional. If the field is optional a condition is described when the field is present. The field may also be described.

#### 4.2 Example Structure

Table 1 — PRC file structure example

| Name                         | Type                                      | Value                                                           |
|------------------------------|-------------------------------------------|-----------------------------------------------------------------|
| <b>flag1</b>                 | <i>Boolean</i>                            | (Required) describe <b>flag1</b>                                |
| <b>data_field1</b>           | <i>&lt;data class1&gt;</i>                | (Optional; if <b>flag1</b> is TRUE) describe data_field1        |
| <b>data_field2</b>           | <i>&lt;data class2&gt;</i>                | (Optional; if <b>flag1</b> is FALSE) describe data_field2       |
| <b>topological_body_data</b> | <i>PRC_TYPE_TOPO_Body</i>                 | (Required) describe <b>topological_body_data</b>                |
| <b>data_type</b>             | <i>&lt;enumerated type&gt; or Integer</i> | (Required) describe <b>data_type</b>                            |
| <b>data_of_type3</b>         | <i>&lt;data class3&gt;</i>                | (Optional; if <b>data_type</b> is 1) describe <b>data_type3</b> |
| <b>data_of_type4</b>         | <i>&lt;data class4&gt;</i>                | (Optional; if <b>data_type</b> is 2) describe <b>data_type4</b> |
| <b>Size</b>                  | <i>UnsignedInteger</i>                    | (Required) describe <b>size</b>                                 |
| <b>array_of_type5</b>        | <i>Array&lt;data class5&gt;[size]</i>     | (Required) describe the array of <data class5>                  |

## 5 PRC file concepts

### 5.1 The PRC file

A PRC File is a sequential binary file, written in a way to make the file portable across machine architectures and operating systems.

PRC is optimized to store various kinds of 3D data, especially those coming from computer-aided design (CAD) systems. Most of the main constructs of CAD systems are supported within the PRC File Format:

- Assemblies and parts
- Trees of 3D entities (coordinate systems, wireframes, surfaces, and solids)
- Exact geometry representation for curves, surfaces
- Tessellated (triangulated) representation
- Markup data

PRC is meant to be multipurpose. There are two ways to store exact geometry and tessellation depending on the usage of the file and on the original information:

- Regular compression is used to directly represent CAD data without loss or transformation from the originating CAD system.
- High compression is used to store very small files, which have a specified physical tolerance from the originating shape. The tolerance is typically 0,001 mm for exact geometric data and 0,01 mm for tessellation data.

Each PRC File corresponds to a single model file (see 8.3.3) which defines the root product occurrences within the FileStructures of the PRC File. A PRC File is a collection of FileStructures which are independent from each other and can come from various authoring PRC File Writers. A FileStructure is the representation of an independent physical file denoting an independent 3D part, assembly, etc. within a PRC file. Hence, there is one header for each FileStructure with specific information and one global header for the model file which contains information about the PRC File Writer that assembled all these individual FileStructures into the final PRC File. Header sections gather primarily information on file version, FileStructure ids and offsets for reading / skipping sections.

Each FileStructure contains one or more product occurrences (see 8.3.10.2). The product occurrences denote the assembly hierarchy of the FileStructure. A product occurrence can have child nodes, which are also product occurrences. There is exactly one root product occurrence in each FileStructure. The root product occurrence is the only entry point to the FileStructure. (refer to Figure 1 below)

In parallel to the FileStructures, the model file also contains an array of root product occurrences that comprise the starting point for the entire assembly description. A product occurrence may refer to a corresponding part definition (see 8.3.11) which in turn contains:

- Geometrical data stored in a tree of representation items (see 8.6.3)
- An optional tree of part markups, grouped into annotation entities and views (see 8.7.2)

A product occurrence can contain:

- A tree of product markups
- Filters used to redefine the loading and presentation of data defined in the child product occurrences or in the part definition (see 8.3.12)

The representation items are defined through a combination of tessellation or topology and geometry data, which may be highly compressed. The markups are defined by tessellation data.

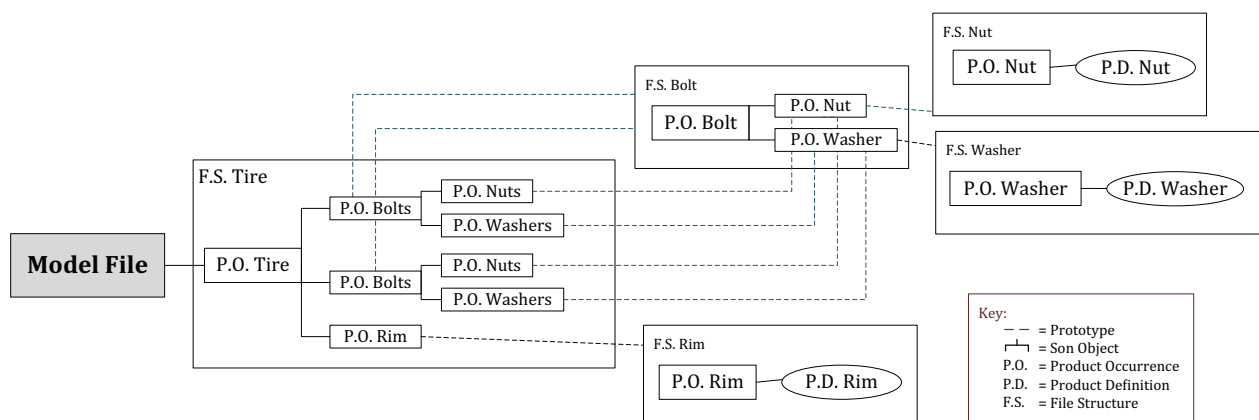


**Figure 1 —Tessellation data**

To optimize reading, a PRC File is arranged so that referenced entities are read before being referenced. Therefore, the FileStructures are ordered using parts, then subassemblies, and finally, the top (root) assembly.

A PRC File is composed of one header section, which starts with uncompressed data, one or more FileStructures, and one model file (*PRC\_TYPE\_ASM\_ModelFile*) data section at the end, each individually compressed. Refer to 8.3.10 for more information about the PRC file structure.

Table 2 — PRC file structure

| Section          | Sub Sections                  | Compression  | Description                                                                                                                        |
|------------------|-------------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------|
| FileHeader       |                               | Uncompressed | Defines originating author of the file; specifies start, and possibly end, location of other sections in file                      |
| File Structure 1 | <i>FileStructureHeader</i>    | Uncompressed | Identifiers of other File Structures referenced from within this FileStructure;                                                    |
|                  | <i>Schema</i>                 | Compressed   | Description of changes between minimal version and authoring version of the FileStructure within the PRC File Format Specification |
|                  | <i>Globals</i>                | Compressed   | Referenced FileStructures and colors, line styles, and coordinate systems for each tree entity of the FileStructure                |
|                  | <i>Tree</i>                   | Compressed   | a description of the tree of items (product occurrences, part definitions, representation items, and markup)                       |
|                  | <i>Tessellation</i>           | Compressed   | All tessellated (triangulated) data in the leaf entities of the tree (representation items and markups).                           |
|                  | <i>Geometry</i>               | Compressed   | All exact geometry and topology data of the leaf entities of the tree(representation items)                                        |
|                  | <i>Extra Geometry</i>         | Compressed   | Geometry summary data, which allow for partial loading of the FileStructure without loading the entire geometry                    |
|                  |                               |              | Additional FileStructure sections in the PRC File                                                                                  |
| File Structure N |                               | Compressed   | Last FileStructure section in the PRC File                                                                                         |
| Schema           | <i>ModelFile Schema</i>       | Compressed   | Description of changes between minimal version and authoring version of the ModelFile Section of the PRC File Format Specification |
| Model File Data  | <i>PRC_TYPE_ASM_ModelFile</i> | Compressed   |                                                                                                                                    |

## 5.2 Versioning

A file format version number is used to define the particular version of this international standard that a PRC File Reader or PRC File Writer conforms to.

Version numbers consist of the year modulo 2000 followed by three digits representing the day of the year. This international standard shall have the version 10001, corresponding to the 1st day of the year 2010.

The **current\_version** is the maximum version number that a PRC File Reader or PRC File Writer conforms to.

Each FileStructure and the FileHeader are independent and can be copied around or written by PRC File Writers of different versions. Each of them have their own **authoring\_version** and **minimal\_version\_for\_read** version numbers. The **authoring\_version** represents the version of the international standard that the PRC File Writer conformed to at the time the FileHeader or the FileStructure was written. The **minimal\_version\_for\_read** represents the version of the international standard that a PRC File Reader shall conform to to successfully read the FileHeader or the FileStructure. If the **current\_version** of the PRC File Reader is less than the **minimal\_version\_for\_read**, then the PRC File Reader shall not continue to process the file and report an error.

If **minimal\_version\_for\_read** is lower than **authoring\_version**, the PRC File Writer shall write a schema description in the PRC File providing the differences between the two versions of this international standard. This description will enable a PRC File Reader to read and skip new information, since these new data cannot be interpreted as they are from a newer version.

A PRC File schema contains a description of new fields or new entity types added between the **minimal\_version\_for\_read** and the **authoring\_version**. See 6.3 for a description of a schema.

A PRC File Reader uses the information in the schema to read and skip new information:

- After reading each entity type, the schema information is queried and new data are skipped accordingly, following the tokens.
- Each time an entity type is read, if the type is unknown, the schema is searched and its data is skipped.

## 5.3 Unique identifiers

### 5.3.1 General

A PRC File reader/writer shall generate (writer) or interpret (reader) unique identifiers for information within the PRC File. Each FileStructure within a PRC File has an identifier (UUID) which uniquely identifies this particular FileStructure among all of the FileStructures within the PRC File. Within each FileStructure, unique identifiers for referenceable entities are generated using the order that they are first encountered in the FileStructure. Thus, the first referenceable entity in the FileStructure has the number 1 as its identifier. Subsequent identifiers of referenceable entities are incremented by 1.

### 5.3.2 File structure

A PRC FileStructure is identified by an identifier (see *UncompressedUniqueId*) which is unique among all of the FileStructures within a single PRC File.

The method to calculate a unique identifier for a given FileStructure is not part of this international standard.

This approach offers an advantage, for example, such as when there is some intent to repurpose FileStructures inside other PRC files without entirely rewriting them.

### 5.3.3 Base entities

The PRC format provides support for referencing entities. See Entity Types in 8.2.1 for a list of entity types whose entities are referenceable.

The purpose of using references on entities is to enable an interpreter to handle the same entity several times without any duplication of data, either by any other program, or by another structure in the same or another PRC file.

A referenceable entity is retrieved using the following:

- The UUID of the FileStructure.
- The entity's unique identifier (a non-zero unsigned 32-bit integer) within the FileStructure.

Just as the FileStructure UUID is unique among all of the FileStructures in the PRC File, the entity identifier is unique inside a FileStructure for all referenceable entities.

A PRC File Writer shall ensure that two referenceable entities inside the same FileStructure do not have the same identifier. The next available index within a FileStructure is the maximum value that has been assigned for identifiers to date, and is stored in the PRC File (see 8.3.4) so it is possible to safely add new entities to a FileStructure and assign them a unique identifier greater than this maximum value.

Data within the FileStructure tessellation and geometry sections are not accessible from outside the FileStructure using the approach for referenceable entities. These data are referenced only by data within the tree section (see 5.1 above) of the same FileStructure. However, it is possible (see 8.4.8) to reference topological data from outside the particular FileStructure which the topological entity lies in. This is restricted to faces in the current version of PRC but may be extended to other data in future versions of the format.

#### 5.3.4 Other systems

In addition to the unique identifier mechanisms described above, this international standard provides for storage of identifiers from the originating system. The external identifiers and their persistence flag are stored as information in PRC strictly to support external workflows. These identifiers by themselves only exist to convey information and do not play any role in PRC.

#### 5.4 Current data values

A PRC File reader and writer shall implement the concept of **CURRENT** for various data.

**NOTE** This enables smaller file sizes since duplicate data need not be written to the file. It also enables faster readers because some of the data is not being read. Default initial values are in parentheses.

- Current name (NULL)
- Current graphics
  - Index of layer (-1)
  - Index of line style (-1)
  - Behavior bit field (-1)

Current values shall be updated as they are encountered in the file. Values shall be reset when serialization is flushed at the end of every flate-compressed section (see 5.8 Compressed File Sections).

#### 5.5 Userdata

The PRC File format provides a mechanism for a PRC File Writer to write private data within various sections of a PRC File. Such data shall consist of a bit stream of data containing the size of the bit stream followed by the specified number of bits.

**NOTE** Any PRC File Reader can read the bit stream and can even resave the private data, but it may not be able to interpret the data.

Each FileStructure within a PRC File may contain UserData. UserData are defined in conjunction with an application unique identifier (UUID) which allows for their interpretation. This application identifier

shall be stored in the FileStructurerHeading. By default, any user data within the FileStructure shall be assumed to be written by this one writer. However, UserData may contain streams from different writers. Therefore, different application UUIDs shall be stored accordingly with each of these other data. UserData are meant to be interpreted only by software which is aware of its meaning according to a given writers UUID. A conforming PRC File Reader should either ignore those UserData, or interpret them according to the writers UUID. PRC also allows applications to define special attributes which behave similarly to UserData, as discussed in 7.3.3.8.

Unique application identifiers are assigned through Adobe Systems, initially, and probably ISO in the long term.. Each company should have it's own unique application identifier, and if they want to share data with another company, they should share application identifiers.

## 5.6 Units

A conforming writer shall define the units used in a PRC File. This should be done at both the model file level (see 8.3.3) and at the product occurrence level (see 8.3.10). The unit may come from an actual CAD file and therefore be considered reliable to represent physical values in the data. If a unit is valid/reliable, the flag unit\_from\_CAD\_file shall be set to TRUE. However, some formats do not contain units. Regardless, it is mandatory to define one unit at both the model file level and at the product occurrence level .

The unit which shall be used is the first valid unit in the ModelFile / ProductOccurrence chain. If a valid unit is defined at the ModelFile level, it will apply to all product occurrences. Once a valid unit is found, the remainder of the data shall be interpreted with respect of that unit, even for occurrences higher in the product occurrence hierarchy.

**NOTE** In other words, if a product occurrence having no valid unit has a child with a valid unit, it is assumed that the entire hierarchy of model file and product occurrence are to be interpreted and used according to this unit.

If no valid unit is found at either the ModelFile level or any ProductOccurrence level (i.e. unit\_from\_CAD\_file is always set to FALSE), a conforming PRC File Reader shall clearly indicate that the unit defined is not valid for measurement purposes.

## 5.7 Tolerances

PRC distinguishes between several notions of tolerances:

- A first notion of tolerance represents the maximum deviation between compressed and original data, as introduced by the lossy compression of geometry or topology. When provided, this tolerance shall be a user-defined value which represents a physical length given with a unit.
- A second notion of tolerance is introduced by numerical uncertainty inherent in every 3D modelling system. This form of tolerance is non-dimensional. Its purpose is to perform consistent numerical operations. This tolerance value corresponds to coincidence (e.g. of two 3D vertices) and is generally defined in conjunction with a minimal value representing zero and a maximal value representing infinity. For instance, a system might define coincidence at  $1e-3$ , zero as any value less than  $1e-12$  and infinity as any value greater than  $1e6$ . Then, additional logic outside the modelling system should define the unit so that the numerical values can be interpreted by the computer as physical values (i.e. in a particular unit).

In PRC, the 3D modelling system corresponds to entities in tessellation section (8.8) and topology section (7.9). Hence the various tolerances stored within these sections are always numerical values with no unit. A conforming writer shall never store a tolerance which can be directly interpreted as a physical value.

**NOTE** For instance, brep\_data\_compressed\_tolerance in 8.9.19 which represents the deviation introduced between original data and compressed one is stored without unit, even if it might be derived from a user interface

which takes those units into account. Then, outside this 3D modelling system, unit is defined in ProductOccurrences and ModelFile as discussed in previous chapter. The physical interpretation of those tolerances should then be done from both indications. For instance, if a tolerance in a topology section is stored as 0,001 and if the unit of the ProductOccurrence it belongs to is 1000 meters, then the physical tolerance on data is actually 1 meter.

## 5.8 Compressed file sections

Within a PRC File, all sections, except header sections, are individually compressed with a Flate method.

**NOTE** This form of compression is considered to be "lossless". It occurs systematically whatever the actual content of the PRC file, and even if it contains compressed geometry or tessellation.

## 5.9 Compressed geometry

Compression of geometry results in very small files. This compression is "lossy" in that the geometry is an approximation to geometry to within a specified tolerance (typically 0,001mm). See 8.9.21 and 8.9.20 for entities representing compressed geometry. Note that the methodology to determine an approximation of the original geometry (e.g. analytic recognition) is not part of PRC standard. Only the resulting entities and the method to store them are described in PRC.

## 5.10 Compressed tessellation

Compression of tessellation data results in very small files. This compression is "lossy" in that the tessellation data is an approximation to tessellation data to within a specified tolerance (typically 0,01mm). See 8.8.9 for an entity representing compressed tessellation. Tessellation data is not necessarily generated from or considered as an "approximation" of geometry; it is an alternative way to convey data. Note that the methodology to determine an approximation of the original tessellation (e.g. polygon decimation) is not part of PRC standard. Only the resulting entities and the method to store them are described in PRC.

# 6 PRC file contents

## 6.1 Fileheader

### 6.1.1 General

The Header contains the file version for the authoring version (PRC Format Specification) that the PRC File Writer is based on and the minimal version for read (PRC Format Specification) that a conforming PRC File Reader is based on that write/read the data outside of the individual FileStructures in the PRC File.

Each FileStructure has a identifier which is unique among all of the FileStructures within this PRC File.

Each application has a unique identifier which enables the interpretation of UserData. This identifier shall be set to 0000 or to a valid application identifier. 0000 indicates that the authoring application is not "registered" but (as discussed above) there can still be user data from another application in the file. Valid identifiers are distributed by Adobe Systems, Inc. upon request.

A valid PRC File shall contain at least one FileStructure.

Table 3 — Fileheader

| Name                            | Data Type                                                                  | Data Description                                                                                                                                                                                 |
|---------------------------------|----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                 | 3 bytes: P, R, C                                                           | (Required) Characters "PRC"                                                                                                                                                                      |
| <b>minimal_version_for_read</b> | <i>UncompressedUnsignedInteger</i>                                         | (Required) Minimal version for read (see 5.2)                                                                                                                                                    |
| <b>authoring_version</b>        | <i>UncompressedUnsignedInteger</i>                                         | (Required) Authoring version (see 5.2)                                                                                                                                                           |
| <b>unique_id_file</b>           | <i>UncompressedUniqueId</i>                                                | (Required) Unique ID for file structure; The first and last sections of PRC File (before and after the FileStructures themselves) are a kind of special file structure which bear an ID as well. |
| <b>unique_id_application</b>    | <i>UncompressedUniqueId</i>                                                | (Required) Unique ID for application                                                                                                                                                             |
| <b>filestructure_count</b>      | <i>UncompressedUnsignedInteger</i>                                         | (Required) Number of FileStructures in PRC file                                                                                                                                                  |
| <b>file_info</b>                | Array<br>< <i>FileStructureDescription</i> >[ <b>filestructure_count</b> ] | (Required) Information describing each FileStructure in PRC File; the ordering of this array reflects the ordering of the FileStructure within the file.                                         |
| <b>start_offset</b>             | <i>UncompressedUnsignedInteger</i>                                         | (Required) Start offset of ModelFileData section from beginning of PRC File (in bytes)                                                                                                           |
| <b>end_offset</b>               | <i>UncompressedUnsignedInteger</i>                                         | (Required) End offset of ModelFileData section from beginning of PRC File (in bytes)                                                                                                             |
| <b>file_count</b>               | <i>UncompressedUnsignedInteger</i>                                         | (Required) Number of uncompressed files that are saved in the PRC File                                                                                                                           |
| <b>files:</b>                   | Array < <i>UncompressedFiles</i> >[ <b>file_count</b> ]                    | (Optional; if <b>file_count</b> is greater than 0) Array of uncompressed files stored in the PRC File                                                                                            |

### 6.1.2 FileStructureDescription

The *FileStructureDescription* contains information defining a particular FileStructure within the PRC File:

- Each FileStructure has an identifier which is unique among the FileStructures within a PRC File.
- The starting offset (in bytes from the beginning of the PRC File) of each of the following sections within a FileStructure: header, globals, tree, tessellation, geometry, and extra geometry.

Table 4 — *FileStructureDescription*

| Name                   | Data Type                                                               | Data Description                                                                                                              |
|------------------------|-------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>unique_id</b>       | <i>UncompressedUniqueId</i>                                             | (Required) Unique id for this file structure                                                                                  |
|                        | <i>UncompressedUnsignedInteger</i>                                      | (Required) Reserved; shall be 0                                                                                               |
| <b>section_count</b>   | <i>UncompressedUnsignedInteger</i>                                      | (Required) Number of sections in this FileStructure (6 for header, globals, tree, tessellation, geometry, and extra geometry) |
| <b>section_offsets</b> | Array<br>< <i>UncompressedUnsignedInteger</i> >[ <b>section_count</b> ] | (Required) Start offset (in bytes) of each section from the beginning of PRC File                                             |

### 6.1.3 UncompressedFiles

Directly embeds private data inside a PRC File. This may be referenced by objects in the same PRC File using the index of the uncompressed file within this array, and interpreted accordingly. Up to this version of the PRC File Format Specification, only picture objects make reference to these files (see 8.5.5)

Table 5 — *UncompressedFiles*

| Name                  | Data Type                                | Data Description                                     |
|-----------------------|------------------------------------------|------------------------------------------------------|
| <b>count</b>          | <i>UncompressedUnsignedInteger</i>       | (Required) Number of uncompressed files              |
| <b>array_of_files</b> | Array< <i>UncompressedBlock</i> >[count] | (Required) Arbitrary data to embed within a PRC File |

## 6.2 Filestructure

### 6.2.1 General

Table 6 — *Filestructure*

| Name                | Data Type                                     | Data Description                                                                                                                                                           |
|---------------------|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>header</b>       | <i>FileStructureHeader</i>                    | (Required)                                                                                                                                                                 |
| <b>schema</b>       | <i>FileStructureSchema</i>                    | (Required) Define the <b>schema</b> for the entities in this FileStructure which have changed between the <b>minimal_version_for_read</b> and the <b>authoring_version</b> |
| <b>globals</b>      | <i>PRC_TYPE_ASM_FileStructureGlobals</i>      | (Required) Referenced FileStructures and colors, line styles, and coordinate systems for each tree entity of the file structure                                            |
| <b>tree</b>         | <i>PRC_TYPE_ASM_FileStructureTree</i>         | (Required) a description of the tree of items (product occurrences, part definitions, representation items, and markup)                                                    |
| <b>tessellation</b> | <i>PRC_TYPE_ASM_FileStructureTessellation</i> | (Required) All tessellated (triangulated) data in the leaf entities of the tree (representation items and markups).                                                        |

**Table 6 (continued)**

| Name                  | Data Type                                      | Data Description                                                                                                            |
|-----------------------|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Geometry</b>       | <i>PRC_TYPE_ASM_FileStructureGeometry</i>      | (Required) All exact geometry and topology data of the leaf entities of the tree (representation items)                     |
| <b>extra_geometry</b> | <i>PRC_TYPE_ASM_FileStructureExtraGeometry</i> | (Required) Geometry summary data, which allow for partial loading of the file structure without loading the entire geometry |

### 6.2.2 FileStructureHeader

A *FileStructureHeader* defines various properties of a FileStructure:

- The minimal file version of the a conforming PRC File Reader;
- The authoring version of a conforming PRC File Writer that wrote this FileStructure;
- The unique ID of this FileStructure; this ID shall be the same as the identifier in the PRC File Header section;
- The unique ID of the conforming PRC File Writer creating this file structure;
- A variable number of private uncompressed data files.

**Table 7 — FileStructureHeader**

| Name                            | Data Type                                         | Data Description                              |
|---------------------------------|---------------------------------------------------|-----------------------------------------------|
|                                 | <i>3 bytes: P, R, C</i>                           | (Required) Characters "PRC"                   |
| <b>minimal_version_for_read</b> | <i>UncompressedUnsignedInteger</i>                | (Required) Minimal version for read (see 5.2) |
| <b>authoring_version</b>        | <i>UncompressedUnsignedInteger</i>                | (Required) Authoring version (see 5.2)        |
| <b>unique_id_file</b>           | <i>UncompressedUniqueId</i>                       | (Required) Unique ID of this FileStructure    |
| <b>unique_id_application</b>    | <i>UncompressedUniqueId</i>                       | (Required) Unique ID for application          |
| <b>file_count</b>               | <i>UncompressedUnsignedInteger</i>                | (Required) Number of uncompressed files       |
| <b>files</b>                    | <i>Array &lt;UncompressedFile&gt;[file_count]</i> | (Required) Array of uncompressed data         |

### 6.2.3 FileStructureSchema

Each FileStructure in a PRC File may represent a different version of the PRC Format Specification written by a different application (PRC File Writer). Each FileStructure shall define the schema for changes (new entities and new data fields to existing entities) for the entities that are stored within it. See 6.3 for a description of the facilities for describing a schema.

## 6.3 PRC Schema

### 6.3.1 General

A schema describes changes between versions of PRC File Format Specification. See 5.2 for a basic description of versioning in PRC. This mechanism allows information to be added for a given entity type (**schema\_type**) and still be readable by previous versions of the software.

Only those types which have changed between the **minimal\_version\_for\_read** and **authoring\_version** require a schema description, and only if they are present in the file.

**Table 8 — PRC Schema**

| Name                | Data Type                                                  | Data Description                                                                                           |
|---------------------|------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| <b>schema_count</b> | <i>UnsignedInteger</i>                                     | (Required) Number of entity types that have changed between minimal version for read and authoring version |
| <b>schemas</b>      | Array<br>< <i>Entity_schema_definition</i> >[schema_count] | (Required) Schema definition of each type that has changed                                                 |

### 6.3.2 *Entity\_schema\_definition*

This provides the schema definition for an entity type in a PRC File. The entity is described by an array of schema tokens which in turn should be viewed as a list of versioned blocks which describe versions of the entity. See 9.3.20 for a description.

**Table 9 — *Entity\_schema\_definition***

| Name                 | Data Type                                        | Data Description                                                                                                          |
|----------------------|--------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>entity_type</b>   | <i>UnsignedInteger</i>                           | (Required) entity_type represents the entity type, such as 8.10.1, that is being described by the array of schema tokens. |
| <b>token_count</b>   | <i>UnsignedInteger</i>                           | (Required) Number of tokens describing this entity type                                                                   |
| <b>schema_tokens</b> | Array<br>< <i>UnsignedInteger</i> >[token_count] | (Required) Array of schema tokens describing this entity type                                                             |

## 7 PRC basic types

### 7.1 General

All data within a physical PRC File exists in a section which is a contiguous stream of bytes which starts and ends on a byte boundary. A section may be either uncompressed (header sections) or compressed (all other sections).

Data within an uncompressed section is written so that individual variables occupy a specific number of bytes.

Data within a compressed section has been written in a bit-by-bit manner with no restrictions on individual variables crossing byte boundaries. At the end of the section, the last byte is padded with zero bits, the entire section is compressed using flate (see Bibliography) and the compressed section is written to the file. At this point, the current name and current graphics are reset.

Reading one section is independent from reading other sections and one can skip directly to a section since the offset (in bytes) of the section from the beginning of the file is known as it is also stored in the PRC File.

## 7.2 Uncompressed types

### 7.2.1 *UncompressedUnsignedInteger*

This type represents writing an unsigned integer. An unsigned integer is converted into a 4 byte array of unsigned characters which is independent of machine byte ordering using the algorithm `MakePortable32BitsUnsigned`.

**Table 10 — *UncompressedUnsignedInteger***

| Name | Data Type | Data Description                                 |
|------|-----------|--------------------------------------------------|
|      | 4 bytes   | (Required) 4 bytes representing unsigned integer |

### 7.2.2 *UncompressedBlock*

The purpose of this function is to write the number of bytes being written followed by the specified number of bytes without the need to further interpret the content of the byte stream.

**Table 11 — *UncompressedBlock***

| Name              | Data Type                          | Data Description                              |
|-------------------|------------------------------------|-----------------------------------------------|
| <b>block_size</b> | <i>UncompressedUnsignedInteger</i> | (Required) size (bytes) of uncompressed block |
| <b>block</b>      | Byte stream of the specified size  | (Required) Block of specified size            |

### 7.2.3 *UncompressedFiles*

This type represents writing the data for multiple uncompressed file data.

**Table 12 — *UncompressedFiles***

| Name              | Data Type                                         | Data Description                           |
|-------------------|---------------------------------------------------|--------------------------------------------|
| <b>file_count</b> | <i>UncompressedUnsignedInteger</i>                | (Required) Number of uncompressed Files    |
| <b>files</b>      | Array<br>< <i>UncompressedBlock</i> >[file_count] | (Required) Array of uncompressed file data |

### 7.2.4 *UncompressedUniqueId*

This saves information on uniqueid in uncompressed mode.

**Table 13 — *UncompressedUniqueId***

| Name              | Data Type                          | Data Description |
|-------------------|------------------------------------|------------------|
| <b>unique_id0</b> | <i>UncompressedUnsignedInteger</i> | (Required)       |
| <b>unique_id1</b> | <i>UncompressedUnsignedInteger</i> | (Required)       |
| <b>unique_id2</b> | <i>UncompressedUnsignedInteger</i> | (Required)       |
| <b>unique_id3</b> | <i>UncompressedUnsignedInteger</i> | (Required)       |

## 7.3 Compressed types

### 7.3.1 General

The following types are used to define data contained within compressed sections of the PRC File. Each type may start or end at any bit position within a byte in the compressed section.

### 7.3.2 Atomic Types

#### 7.3.2.1 *Bits(n)*

Writes n-number of bits from left to right.

#### 7.3.2.2 *Boolean*

Boolean values shall be written as a single bit

**Table 14 — *Boolean***

| Name | Data Type      | Data Description                      |
|------|----------------|---------------------------------------|
|      | <i>Bits(1)</i> | (Required) Single bit in a bit stream |

#### 7.3.2.3 *Character*

Characters will be written as a single 8 bits.

**Table 15 — *Character***

| Name | Data Type      | Data Description                            |
|------|----------------|---------------------------------------------|
|      | <i>Bits(8)</i> | (Required) Single character in a bit stream |

#### 7.3.2.4 *Integer*

This requires a special algorithm. See 10.11.

#### 7.3.2.5 *IntegerWithVariableBitNumber*

This requires a special algorithm. See 10.12.

#### 7.3.2.6 *UnsignedInteger*

This requires a special algorithm. See 10.10.

#### 7.3.2.7 *UnsignedIntegerWithVariableBitNumber*

This requires a special algorithm. See 10.13.

#### 7.3.2.8 *NumberOfBitsThenUnsignedInteger*

This requires a special algorithm. See 10.15.

#### 7.3.2.9 *Float*

This requires a special algorithm. See 10.5.

#### 7.3.2.10 *Double*

This requires a special algorithm. See 10.17.

### 7.3.2.11 *DoubleWithVariableBitNumber*

This requires a special algorithm. See 10.14.

## 7.3.3 Compound Types

### 7.3.3.1 *CompressedEntityType*

This requires a special algorithm. See 10.16.

### 7.3.3.2 *CompressedUniqueId*

This saves information on uniqueid in compressed mode. See 5.3

**Table 16 — *CompressedUniqueId***

| Name              | Data Type              | Data Description |
|-------------------|------------------------|------------------|
| <b>unique_id0</b> | <i>UnsignedInteger</i> | (Required)       |
| <b>unique_id1</b> | <i>UnsignedInteger</i> | (Required)       |
| <b>unique_id2</b> | <i>UnsignedInteger</i> | (Required)       |
| <b>unique_id3</b> | <i>UnsignedInteger</i> | (Required)       |

### 7.3.3.3 *String*

This is a UTF8-encoded string without a terminating NULL character. The number of characters does not include a terminating null character.

**Table 17 — *String***

| Name             | Data Type                        | Data Description                                                      |
|------------------|----------------------------------|-----------------------------------------------------------------------|
| <b>null_flag</b> | <i>Boolean</i>                   | (Required) TRUE if the string is not NULL; else FALSE                 |
| <b>size</b>      | <i>UnsignedInteger</i>           | (Optional; if <b>null_flag</b> is TRUE) Size of character array       |
| <b>string</b>    | Array <Character>[ <b>size</b> ] | (Optional; if <b>null_flag</b> is TRUE) Array of characters in String |

### 7.3.3.4 *CharacterArray*

This requires a special algorithm. See 10.6.

### 7.3.3.5 *ShortArray*

This requires a special algorithm. See 10.7.

### 7.3.3.6 *CompressedIntegerArray*

This requires a special algorithm. See 10.8.

### 7.3.3.7 *CompressedIndiceArray*

This requires a special algorithm. See 10.9.

### 7.3.3.8 UserData

Applications that write a PRC File may store an arbitrary bit stream of private data for the following data types :

- Subtypes of PRC\_TYPE\_ASM
- Subtypes of PRC\_TYPE\_RI
- Subtypes of PRC\_TYPE\_MKP
- Subtypes of PRC\_TYPE\_MISC

All those entities have UserData at the end of their data definition.

**Table 18 — UserData**

| Name               | Data Type              | Data Description                                                                        |
|--------------------|------------------------|-----------------------------------------------------------------------------------------|
| <b>stream_size</b> | <i>UnsignedInteger</i> | (Required) Number of bits in the <i>UserDataStream</i>                                  |
| <b>stream</b>      | <i>UserDataStream</i>  | (Optional; if <b>stream_size</b> > 0) Arbitrary bit stream of <b>stream_size</b> length |

#### 7.3.3.8.1 UserDataStream

**Table 19 — UserDataStream**

| Name                 | Data Type                                    | Data Description                                             |
|----------------------|----------------------------------------------|--------------------------------------------------------------|
| <b>section_count</b> | <i>UnsignedInteger</i>                       | (Required) Number of UserData sub sections in the bit stream |
| <b>sub_sections</b>  | Array<br><UserDataSubSection>[section_count] | (Required) Array of sub-sections                             |

#### 7.3.3.8.2 UserDataSubSection

| Name                         | Data Type                 | Data Description                                                                 |
|------------------------------|---------------------------|----------------------------------------------------------------------------------|
| <b>same_flag</b>             | <i>Boolean</i>            | (Required) Same Application UUID as FileStructure                                |
| <b>unique_id_application</b> | <i>CompressedUniqueId</i> | (Optional; if <b>same_flag</b> is FALSE) Application UUID                        |
| <b>stream_size</b>           | <i>UnsignedInteger</i>    | (Required) Number of bits in the bit stream for this user data subsection        |
| <b>stream</b>                | <i>Bits(n)</i>            | (Optional; if stream_size is > 0) Arbitrary bit stream of the stream_size length |

### 7.3.4 Parameter Range Types

#### 7.3.4.1 Infinite\_param

The constant **infinite\_param** shall have the double precision floating point value 12345. It is used to represent infinity, and as **-infinite\_param** also minus infinity, for parameter values in **interval**.

### 7.3.4.2 Interval

An Interval is a subset of  $R^1$ . It is represented by two double precision numbers defining the minimum and maximum values of the interval. An interval is defined to be all values between the minimum and maximum

$$\text{minimum} \leq t \leq \text{maximum}$$

The minimum shall not be smaller than **-infinite\_param** and the maximum shall not be larger than **infinite\_param**. Infinite or semi-infinite intervals may be specified by using **infinite\_param** to represent infinity. The following are examples of intervals:

- [0.0, 1.0]
- [-infinite\_param, infinite\_param]
- [0.0, infinite\_param]

The primary use of an interval is to define the domain of a curve. For a non-periodic curve, whether an open or closed curve, the interval defines the domain of legal parameter values for the curve. For a periodic closed curve, such as a circle or periodic closed NURBS curve, the interval defines the period (i.e. the length of the interval) as well as the primary domain of the curve. Any value is a legal parameter when evaluated modulo the period. For a periodic open curve, such as a proper subset of a circle, the interval defines the valid subset of  $R^1$  that is the domain of legal parameter values.

**Table 20 — Interval**

| Name             | Data Type     | Data Description         |
|------------------|---------------|--------------------------|
| <b>min_value</b> | <i>Double</i> | (Required) Minimum value |
| <b>max_value</b> | <i>Double</i> | (Required) MaximumValue  |

### 7.3.4.3 Parameterization

Parameterization data provide a way to reparameterize a curve and to define the domain of the curve. The domain of the curve define legal parameter values.

Two doubles, **coeff\_a** and **coeff\_b**, are used to reparameterize a curve towards its implicit parameterization.

The evaluation formula to calculate the implicit parameter from a parameter is:

$$\text{implicit\_parameter} \leftarrow . \text{coeff\_a} * \text{parameter} + \text{coeff\_b}$$

If there is no reparameterization of the curve, **coeff\_a** shall be set to 1.0 and **coeff\_b** shall be set to 0.0. In this case **implicit\_parameter** equals the specified parameter.

Parameterization data also contain an interval used to define the domain of the curve. This interval restricts the curve before applying the reparameterization formula. The reparameterization formula can be used to calculate the **implicit\_interval** from the given interval in the same way that it is used to calculate the **implicit\_parameter** from a given parameter.

All curves shall have an interval to define the legal parameter values of the curve. In the case of base curves (curves not defined by reference to other curves), this interval defines the domain of definition for the curve. For curves which are defined in terms of other curves, the interval represents a subset of the curve which may be the entire curve or a portion of it. The interpretation of the interval depends upon the curve being periodic or non-periodic (see the definition of Interval).

For instance, a circle has the implicit domain of  $[0.0, 2\pi]$  and a line has the implicit domain of  $[-\infty, \infty]$ . A portion of the circle might be limited to  $[0.0, \pi]$ .

**Table 21 — Parameterization**

| Name                 | Data Type       | Data Description         |
|----------------------|-----------------|--------------------------|
| <b>trim_interval</b> | <i>Interval</i> | (Required) trim interval |
| <b>coeff_a</b>       | <i>Double</i>   | (Required) Coeff_a       |
| <b>coeff_b</b>       | <i>Double</i>   | (Required) Coeff_b       |

As an example, consider a circle. It has an implicit parameterization on the interval  $[0.0, 2\pi]$ . For this case:

- Coeff\_a = 1.0
- Coeff\_b = 0.0
- Interval =  $[0.0, 2\pi]$

To reparameterize the circle so the parameter values are in the interval  $[0.0, 1.0]$  would give

- Coeff\_a =  $2\pi$
- Coeff\_b = 0.0
- Interval =  $[0.0, 1.0]$

To define a semi circle

- Coeff\_a = 1.0
- Coeff\_b = 0.0
- Interval =  $[-\pi/2.0, \pi/2.0]$

#### 7.3.4.4 Domain

A Domain is a subset of  $R^2$  that is used to define the domain of definition of a surface. It is represented by a 2D vector defining the minimum UV parameter values and a 2D vector defining the maximum UV parameters.

The domain of the surface is

$$\text{minimum } U \leq u \leq \text{maximum } U$$

$$\text{minimum } V \leq v \leq \text{maximum } V$$

The interval in U (or V) when used with a specific surface definition will define a surface that is open, closed, or periodic in that parameter. For a non-periodic surface, whether an open or closed surface, the interval defines the domain of legal parameter values for the surface. For a periodic closed surface, such as a cone or periodic closed NURBS surface, the interval defines the period (i.e. the length of the interval) as well as the primary domain of the surface. Any value is a legal parameter when evaluated modulo the period. For a periodic open surface, such as a proper subset of a cone, the interval defines the valid subset of  $R^2$  that is the domain of legal parameter values.

Table 22 — Domain

| Name          | Data Type       | Data Description           |
|---------------|-----------------|----------------------------|
| <b>min_uv</b> | <i>Vector2d</i> | (Required) Minimum U and V |
| <b>max_uv</b> | <i>Vector2d</i> | (Required) Maximum U and V |

#### 7.3.4.5 UVParameterization

This describes a reparameterization of a surface towards implicit parameterization. Coeff\_a shall be set to 1.0 and Coeff\_b shall be set to 0.0 if there is no reparameterization.

The evaluation formula is:

If (swap\_uv is FALSE) {

implicit\_param.u  $\leftarrow$  **u\_param\_coeff\_a** \* param.u + **u\_param\_coeff\_b**

implicit\_param.v  $\leftarrow$  **v\_param\_coeff\_a** \* param.v + **v\_param\_coeff\_b**

} else {

implicit\_param.u  $\leftarrow$  **v\_param\_coeff\_a** \* param.v + **v\_param\_coeff\_b**

implicit\_param.v  $\leftarrow$  **u\_param\_coeff\_a** \* param.u + **u\_param\_coeff\_b**

}

Table 23 — UVParameterization

| Name                   | Data Type      | Data Description                                                                                                        |
|------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>swap_uv</b>         | <i>Boolean</i> | (Required) <b>swap_uv</b> TRUE implies swap the uv param; FALSE implies do not swap                                     |
| <b>surface_domain</b>  | <i>Domain</i>  | (Required) The domain of the surface is specified in numbers before the previous reparameterization formula is applied. |
| <b>u_param_coeff_a</b> | <i>Double</i>  | (Required)                                                                                                              |
| <b>v_param_coeff_a</b> | <i>Double</i>  | (Required)                                                                                                              |
| <b>u_param_coeff_b</b> | <i>Double</i>  | (Required)                                                                                                              |
| <b>v_param_coeff_b</b> | <i>Double</i>  | (Required)                                                                                                              |

### 7.3.5 Basic Geometry Types

#### 7.3.5.1 Vector2d

Representation of a 2D vector. This type may be used to represent either 2D positions or vectors. The context shall be used to determine if a position or vector is meant.

Table 24 — Vector2d

| Name           | Data Type     | Data Description |
|----------------|---------------|------------------|
| <b>x_value</b> | <i>Double</i> | (Required)       |
| <b>y_value</b> | <i>Double</i> | (Required)       |

### 7.3.5.2 Vector3d

Representation of a 3D vector. This type may be used to represent either 3D positions or vectors. The context shall be used to determine if a position or vector is meant.

**Table 25 — Vector3d**

| Name           | Data Type     | Data Description |
|----------------|---------------|------------------|
| <b>x_value</b> | <i>Double</i> | (Required)       |
| <b>y_value</b> | <i>Double</i> | (Required)       |
| <b>z_value</b> | <i>Double</i> | (Required)       |

### 7.3.5.3 BoundingBox

Define a bounding box with sides parallel to the XYZ coordinate planes using two diagonal corners of the box. The minimum and maximum shall satisfy

- $X_{min} < X_{max}$
- $Y_{min} < Y_{max}$
- $Z_{min} < Z_{max}$

If the above is not satisfied the bounding box is considered invalid.

**Table 26 — BoundingBox**

| Name                  | Data Type       | Data Description |
|-----------------------|-----------------|------------------|
| <b>minimum_corner</b> | <i>Vector3d</i> | (Required)       |
| <b>maximum_corner</b> | <i>Vector3d</i> | (Required)       |

## 8 Base entities

### 8.1 General

**BASE ENTITIES** represent high level concepts such as curves, surfaces, topology, parts, assemblies, markups, or tessellation data which are stored in a PRC File. Entities are defined by a type name used for descriptive purposes, a type value which is stored in the PRC File to indicate that the data defining the entity follows, and an indication if the entity is referenceable. An entity is referenceable if it may be referenced using a unique identifier.

### 8.2 Abstract root types

#### 8.2.1 Entity types

**Table 27 — Abstract root types entity types**

| Type Name                                | Type Value               |
|------------------------------------------|--------------------------|
| <i>PRC_TYPE_ROOT</i>                     | 0                        |
| <i>PRC_TYPE_ROOTBase</i>                 | <i>PRC_TYPE_ROOT</i> + 1 |
| <i>PRC_TYPE_ROOT_PRCBaseWithGraphics</i> | <i>PRC_TYPE_ROOT</i> + 2 |
| <i>PRC_TYPE_ROOT_PRCBaseNoReference</i>  | <i>PRC_TYPE_ROOT</i> + 3 |

### 8.2.2 *PRC\_TYPE\_ROOT*

An entity of this type in a PRC File is to be interpreted as a NULL pointer or entity depending on the specific circumstances.

### 8.2.3 *PRC\_TYPE\_ROOTBase*

#### 8.2.3.1 General

This is the abstract root type for any PRC entity.

#### 8.2.3.2 *ContentPRCBase*

This represents common data for all PRC base entities that are not eligible to be referenced.. All PRC base entities have attribute and name information.

**Table 28 — *ContentPRCBase***

| Name                  | Data Type            | Data Description                                     |
|-----------------------|----------------------|------------------------------------------------------|
| <b>attribute_data</b> | <i>AttributeData</i> | (Required) Attribute data associated with the entity |
| <b>entity_name</b>    | <i>Name</i>          | (Required) Entity Name                               |

#### 8.2.3.3 *ContentPRCRefBase*

This represents common data for all PRC base entities that are eligible to be referenced. All PRC base entities have attribute, name information, a PRC FileStructure unique identifier and a persistent and non-persistent identifier from the originating CAD file.

**Table 29 — ContentPRCRefBase**

| Name                  | Data Type              | Data Description                                               |
|-----------------------|------------------------|----------------------------------------------------------------|
| <b>attribute_data</b> | <i>AttributeData</i>   | (Required) Attribute data associated with the entity           |
| <b>entity_name</b>    | <i>Name</i>            | (Required) Entity Name                                         |
| <b>unique_id_cad</b>  | <i>UnsignedInteger</i> | (Required) persistent CAD identifier from originating CAD file |
| <b>unique_id</b>      | <i>UnsignedInteger</i> | (Required) PRC FileStructure unique identifier                 |

**8.2.3.3.1 AttributeData**

A base entity may have zero or more associated attributes.

**Table 30 — AttributeData**

| Name                   | Data Type                                           | Data Description                  |
|------------------------|-----------------------------------------------------|-----------------------------------|
| <b>attribute_count</b> | <i>UnsignedInteger</i>                              | (Required) Number of attributes   |
| <b>attributes</b>      | Array<br><PRC_TYPE_MISC_Attribute>[attribute_count] | (Required) An array of attributes |

**8.2.3.3.2 Name**

PRC employs the concept of a *current\_name* which retains the name of the last entity being read or written. If the name of the subsequent entity being read or written is the same as the current name, no name will be in the file. Otherwise, a name for the entity is read from the file and the current name is updated to this new name. This is done to optimize on space within the file.

**Table 31 — Name**

| Name             | Data Type      | Data Description                                                                                                              |
|------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>same_name</b> | <i>Boolean</i> | (Required) TRUE implies the name of this entity is the same as the current name; FALSE implies that a new name is in the file |
| <b>name</b>      | <i>String</i>  | (Optional; if <b>same_name</b> is FALSE) Name of the entity and the current name is set to this name                          |

**8.2.4 PRC\_TYPE\_ROOT\_PRCBaseWithGraphics****8.2.4.1 General**

Information for any base PRC entity which can be referenced and which contains graphics.

PRC employs the concept of current graphics content which retains the graphics content of the last entity being read or written. If the graphics content of the subsequent entity being read or written is the same as that of the current graphics content, no graphics content will be in the file. Otherwise, a graphics content for the entity is read from the file and the current graphics content is updated to this new graphics content. This is done to optimize on space within the file.

Table 32 — *PRC\_TYPE\_ROOT\_PRCBaseWithGraphics*

| Name                   | Data Type                | Data Description                                                                                                                                       |
|------------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>base</b>            | <i>ContentPRCRefBase</i> | (Required) Base information associated with the entity                                                                                                 |
| <b>same_graphics</b>   | <i>Boolean</i>           | (Required) SameGraphicsAsCurrent                                                                                                                       |
| <b>graphic_content</b> | <i>GraphicsContent</i>   | (Optional; if <b>same_graphics</b> is TRUE) Graphical data associated with the entity and the current graphics content is set to this graphics content |

#### 8.2.4.2 GraphicsContent

**index\_of\_line\_style** represents the index into the array of styles, which is stored in FileStructureInternalGlobalData (See section 8.3.5.2 below). The array of styles is a set of PRC\_TYPE\_GRAPH\_Style entities (See 8.5.3) and should have a value less than 65535.

**behaviour\_bit\_field** is an unsigned short integer (2 bytes). This bit field controls visibility and removal of an entity as well as how values of entities (visibility, color, transparency, layer, line pattern, and line width) are inherited in the tree of entities.

Each value that is subject to inheritance has two bit flags in the **behavior\_bit\_field**: one flag is called ChildHerit. If ChildHerit is set, the actual value of ParentHerit is ignored and assumed to be FALSE.

The actual value that the entity uses depends on the path of entities from the root of the tree of entities to this entity and the **behavior\_bit\_field** settings along this path.

If there are ChildHerit flags in the tree, it is the lowest node in the tree which has this flag which defines the value. Else if there are ParentHerit flags in the tree, it is the highest node in the tree which has this flag which defines the value. Else if there is no flag, the current value is set, if any.

Potential values and meanings of this bit field are:

Table 33 — **behavior\_bit\_field**

| Name                                        | Value  | Meaning                                                       |
|---------------------------------------------|--------|---------------------------------------------------------------|
| <b>PRC_GRAPHICS_Show</b>                    | 0x0001 | The entity is shown.                                          |
| <b>PRC_GRAPHICS_ChildHeritShow</b>          | 0x0002 | Shown entity child inheritance.                               |
| <b>PRC_GRAPHICS_FatherHeritShow</b>         | 0x0004 | Shown entity parent inheritance.                              |
| <b>PRC_GRAPHICS_ChildHeritColor</b>         | 0x0008 | Color/material child inheritance.                             |
| <b>PRC_GRAPHICS_ParentHeritColor</b>        | 0x0010 | Color/material parent inheritance                             |
| <b>PRC_GRAPHICS_ChildHeritLayer</b>         | 0x0020 | Layer child inheritance.                                      |
| <b>PRC_GRAPHICS_ParentHeritLayer</b>        | 0x0040 | Layer parent inheritance.                                     |
| <b>PRC_GRAPHICS_ChildHeritTransparency</b>  | 0x0080 | Transparency child inheritance.                               |
| <b>PRC_GRAPHICS_ParentHeritTransparency</b> | 0x0100 | Transparency parent inheritance                               |
| <b>PRC_GRAPHICS_ChildHeritLinePattern</b>   | 0x0200 | Line pattern child inheritance.                               |
| <b>PRC_GRAPHICS_ParentHeritLinePattern</b>  | 0x0400 | Line pattern parent inheritance.                              |
| <b>PRC_GRAPHICS_ChildHeritLineWidth</b>     | 0x0800 | Line width child inheritance                                  |
| <b>PRC_GRAPHICS_ParentHeritLineWidth</b>    | 0x1000 | Line width parent inheritance                                 |
| <b>PRC_GRAPHICS_Removed</b>                 | 0x2000 | The entity has been removed and no longer appears in the tree |

Table 34 — *GraphicsContent*

| Name                              | Data Type                | Data Description                                                                                                                                                                                                                                                                                                                                  |
|-----------------------------------|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>biased_layer_index</b>         | <i>UnsignedInteger</i>   | <b>(Required)</b> <b>layer_index</b> + 1 where <b>layer_index</b> represents the layer the entity lies on. It should have a value less than 65535.                                                                                                                                                                                                |
| <b>biased_index_of_line_style</b> | <i>UnsignedInteger</i>   | <b>(Required)</b> <b>index_of_line_style</b> + 1 where <b>index_of_line_style</b> represents the index into the array of styles, which is stored in <i>FileStructureInternalGlobalData</i> (See section 8.3.5.2 below). The array of styles is a set of <i>PRC_TYPE_GRAPH_Style</i> entities (See 8.5.3) and should have a value less than 65535. |
| <b>behavior_bit_field1</b>        | <i>UnsignedCharacter</i> | <b>(Required)</b> <b>behavior_bit_field1</b> is <b>behavior_bit_field &amp; 0xff</b>                                                                                                                                                                                                                                                              |
| <b>behavior_bit_field2</b>        | <i>UnsignedCharacter</i> | <b>(Required)</b> <b>behavior_bit_field2</b> is <b>behavior_bit_field &gt;&gt; 8</b>                                                                                                                                                                                                                                                              |

### 8.2.5 *PRC\_TYPE\_ROOT\_PRCBaseNoReference*

This is the abstract root type for any PRC entity that can not be referenced.

This type is useful for schema descriptions. For example, *EPRCSchema\_Parent\_Type* can be used to define a new type which has one of three different ancestor types. It can be a child of either *PRC\_TYPE\_ROOTBase*, *PRC\_TYPE\_ROOT\_PRCBaseWithGraphics*, or *PRC\_TYPE\_ROOT\_PRCBaseNoReference*.

*PRC\_TYPE\_ROOTBase*: ancestor type would indicate that the new type is a referencable type

*PRC\_TYPE\_ROOT\_PRCBaseNoReference*: ancestor type would indicate that the new type is not referencable.

*PRC\_TYPE\_ROOT\_PRCBaseWithGraphics*.: ancestor type would indicate that the new type is a referencable type which bear graphics

Examples :

*PRC\_TYPE\_GRAPH\_LinePattern* has *PRC\_TYPE\_ROOTBase* in its ancestor chain

*PRC\_TYPE\_ASM\_FileStructure* has *PRC\_TYPE\_ROOT\_PRCBaseNoReference* in its ancestor chain

*PRC\_TYPE\_RI\_Curve* has *PRC\_TYPE\_ROOT\_PRCBaseWithGraphics* in its ancestors chain

## 8.3 Structure and assembly

### 8.3.1 Entity types

Table 35 — Structure and assembly entity types

| Type Name                                      | Type Value                 | Referenceable |
|------------------------------------------------|----------------------------|---------------|
| <i>PRC_TYPE_ASM</i>                            | <i>PRC_TYPE_ROOT</i> + 300 | no            |
| <i>PRC_TYPE_ASM_ModelFile</i>                  | <i>PRC_TYPE_ASM</i> + 1    | no            |
| <i>PRC_TYPE_ASM_FileStructure</i>              | <i>PRC_TYPE_ASM</i> + 2    | no            |
| <i>PRC_TYPE_ASM_FileStructureGlobals</i>       | <i>PRC_TYPE_ASM</i> + 3    | no            |
| <i>PRC_TYPE_ASM_FileStructureTree</i>          | <i>PRC_TYPE_ASM</i> + 4    | no            |
| <i>PRC_TYPE_ASM_FileStructureTessellation</i>  | <i>PRC_TYPE_ASM</i> + 5    | no            |
| <i>PRC_TYPE_ASM_FileStructureGeometry</i>      | <i>PRC_TYPE_ASM</i> + 6    | no            |
| <i>PRC_TYPE_ASM_FileStructureExtraGeometry</i> | <i>PRC_TYPE_ASM</i> + 7    | no            |
| <i>PRC_TYPE_ASM_ProductOccurrence</i>          | <i>PRC_TYPE_ASM</i> + 8    | yes           |
| <i>PRC_TYPE_ASM_PartDefinition</i>             | <i>PRC_TYPE_ASM</i> + 9    | yes           |
| <i>PRC_TYPE_ASM_Filter</i>                     | <i>PRC_TYPE_ASM</i> + 10   | yes           |

### 8.3.2 *PRC\_TYPE\_ASM*

This is the abstract type for the top level PRC structure.

### 8.3.3 *PRC\_TYPE\_ASM\_ModelFile*

#### 8.3.3.1 General

A model file (*PRC\_TYPE\_ASM\_ModelFile*) is typically created by importing data from a CAD file. There is only a single *PRC\_TYPE\_ASM\_ModelFile* entity in a PRC File. The model file contains product occurrences, which are split into different FileStructures.

**units\_from\_CAD\_file** is to be interpreted as discussed in section *Units* (See 5.6)

**number\_of\_root\_product\_occurrences** is the number of root product occurrences in the model file.

**product\_occurrences** represents the root product occurrences in the model file.

**file\_structure\_index\_in\_model\_file** indicates the index at which the FileStructure should be stored in memory within a ModelFile (*PRC\_TYPE\_ASM\_ModelFile*). A conforming PRC File Reader should reconstitute / maintain FileStructures in memory in the order that they appear in the ModelFile, regardless of the order in the physical PRC file. In the physical PRC file, the order of file structures shall be in accordance with their dependencies with other file structures as follows: if a given file structure A depends on another file structure B (in the sense that A references elements of B), B shall appear first in the PRC physical file. This restriction does not exist for the memory storage of the file structures.

**number\_of\_file\_structures** is obtained directly from the header of the PRC File.

Table 36 — *PRC\_TYPE\_ASM\_ModelFile*

| Name                                      | Data Type                                                                                 | Data Description                                                                                                                                                                               |
|-------------------------------------------|-------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                           | <i>UnsignedInteger</i>                                                                    | (Required)<br><i>PRC_TYPE_ASM_ModelFile</i>                                                                                                                                                    |
| <b>Base</b>                               | <i>ContentPRCBase</i>                                                                     | (Required) Base information associated with the entity                                                                                                                                         |
| <b>units_from_CAD_file</b>                | <i>Boolean</i>                                                                            | (Required) <i>units_from_CAD_file</i> is TRUE if the units come from a CAD system and are thus suitable for measurement; else FALSE                                                            |
|                                           | <i>Double</i>                                                                             | (Optional; if <b>units_from_CAD_file</b> is TRUE) Units in multiple of mm                                                                                                                      |
| <b>number_of_root_product_occurrences</b> | <i>UnsignedInteger</i>                                                                    | (Required)<br><b>number_of_root_product_occurrences</b>                                                                                                                                        |
| <b>product_occurrences</b>                | Array < <i>ProductOccurrenceReference</i> > [ <b>number_of_root_product_occurrences</b> ] | (Required) References to the root <b>product occurrences</b> in the model file                                                                                                                 |
| <b>file_structure_index_in_model_file</b> | Array < <i>UnsignedInteger</i> > [ <b>number_of_root_product_occurrences</b> ]            | (Required)<br><b>file_structure_index_in_model_file</b> : Indices to to <i>FileStructure</i> within the model file; the size of the array is obtained directly from the header of the PRC file |
| <b>user_data</b>                          | <i>UserData</i>                                                                           | (Required) User defined data                                                                                                                                                                   |

### 8.3.3.2 *ProductOccurrenceReference*

This defines the unique identifier of the *PRC\_TYPE\_ASM\_FileStructure* and the index of the root product occurrence in the array of product occurrences within the *PRC\_TYPE\_ASM\_FileStructureTree*, as contained within the *PRC\_TYPE\_ASM\_FileStructure*.

**product\_occurrence\_is\_active** is reserved for future use, and is currently unused. Its default value should be TRUE. Its use will be to control storing different versions/configurations of a product in the same PRC File. Currently, only one can be active at a time but you can switch from one to the other.

Table 37 — *ProductOccurrenceReference*

| Name                                | Data Type                 | Data Description                                                                                     |
|-------------------------------------|---------------------------|------------------------------------------------------------------------------------------------------|
| <b>unique_id</b>                    | <i>CompressedUniqueId</i> | (Required) Unique identifier of the <i>FileStructure</i> that contains this root product occurrence. |
| <b>root_index</b>                   | <i>UnsignedInteger</i>    | (Required) Index of the root product occurrence within the <i>FileStructure</i>                      |
| <b>product_occurrence_is_active</b> | <i>Boolean</i>            | (Required) TRUE if the product occurrence is active; else FALSE                                      |

### 8.3.4 *PRC\_TYPE\_ASM\_FileStructure*

This type gathers internal data of a file structure as described in *PRC\_TYPE\_ASM\_FileStructureTree*

**Table 38 — *PRC\_TYPE\_ASM\_FileStructure***

| Name                            | Data Type              | Data Description                                                                                                                                                               |
|---------------------------------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                 | <i>UnsignedInteger</i> | (Required) <i>PRC_TYPE_ASM_FileStructure</i>                                                                                                                                   |
| <b>Base</b>                     | <i>ContentPRCBase</i>  | (Required) Base information associated with the entity                                                                                                                         |
| <b>next_available_index</b>     | <i>UnsignedInteger</i> | (Required) <b>next_available_index</b> is used when re-opening the FileStructure, to be able to safely add entities with a unique id without overlap to pre-existing entities. |
| <b>index_product_occurrence</b> | <i>UnsignedInteger</i> | (Required) <b>index_product_occurrence</b> is used to denote which product occurrence inside the FileStructure is the unique root.                                             |

### 8.3.5 *PRC\_TYPE\_ASM\_FileStructureGlobals*

#### 8.3.5.1 General

This type gathers global data of a file structure as described in Section 6.2.

**Table 39 — *PRC\_TYPE\_ASM\_FileStructureGlobals***

| Name               | Data Type                                                        | Data Description                                                                                                    |
|--------------------|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
|                    | <i>UnsignedInteger</i>                                           | (Required)<br><i>PRC_TYPE_ASM_FileStructureGlobals</i>                                                              |
| <b>Base</b>        | <i>ContentPRCBase</i>                                            | (Required) Base information associated with the entity                                                              |
| <b>file_count</b>  | <i>UnsignedInteger</i>                                           | (Required) Number of referenced FileStructures                                                                      |
| <b>unique_ids</b>  | Array <i>&lt;CompressedUniqueId&gt;</i><br>[ <b>file_count</b> ] | (Required) Unique ids for FileStructures within the PRC File which are referenced by entities in this FileStructure |
| <b>global_data</b> | <i>FileStructureInternalGlobalData</i>                           | (Required)                                                                                                          |
| <b>user_data</b>   | <i>UserData</i>                                                  | (Required) User defined data                                                                                        |

#### 8.3.5.2 *FileStructureInternalGlobalData*

##### 8.3.5.2.1 General

This internal structure is used in *PRC\_TYPE\_Asm\_FileStructureGlobals*.

Table 40 — *FileStructureInternalGlobalData*

| Name                      | Data Type                                                                      | Data Description                                  |
|---------------------------|--------------------------------------------------------------------------------|---------------------------------------------------|
| <b>tess_chord</b>         | <i>Double</i>                                                                  | (Required) Tessellation chord height              |
| <b>tess_angle</b>         | <i>Double</i>                                                                  | (Required) Tessellation angle (degrees)           |
| <b>serialize_help</b>     | <i>MarkupSerializationHelper</i>                                               | (Required) See below                              |
| <b>color_count</b>        | <i>UnsignedInteger</i>                                                         | (Required) Number of colors                       |
| <b>Colors</b>             | Array <i>&lt;RgbColor&gt;</i> [ <b>color_count</b> ]                           | (Required) Array of color definitions             |
| <b>picture_count</b>      | <i>UnsignedInteger</i>                                                         | (Required) Number of pictures                     |
| <b>Pictures</b>           | Array <i>&lt;PRC_TYPE_GRAPH_Picture&gt;</i> [ <b>picture_count</b> ]           | (Required) Array of pictures                      |
| <b>texture_count</b>      | <i>UnsignedInteger</i>                                                         | (Required) Number of textures definitions         |
| <b>Textures</b>           | Array <i>&lt;PRC_TYPE_GRAPH_TextureDefinition&gt;</i> [ <b>texture_count</b> ] | (Required) Array of texture definitions           |
| <b>material_count</b>     | <i>UnsignedInteger</i>                                                         | (Required) Number of materials                    |
| <b>Materials</b>          | Array <i>&lt;PRC_TYPE_GRAPH_Material&gt;</i> [ <b>material_count</b> ]         | (Required) Array of materials                     |
| <b>line_pattern_count</b> | <i>UnsignedInteger</i>                                                         | (Required) Number of line patterns                |
| <b>line_patterns</b>      | Array <i>&lt;PRC_TYPE_GRAPH_LinePattern&gt;</i> [ <b>line_pattern_count</b> ]  | (Required) Array of line patterns                 |
| <b>style_count</b>        | <i>UnsignedInteger</i>                                                         | (Required) Number of styles                       |
| <b>Styles</b>             | Array <i>&lt;PRC_TYPE_GRAPH_Style&gt;</i> [ <b>style_count</b> ]               | (Required) Array of category 1 line styles        |
| <b>fill_count</b>         | <i>UnsignedInteger</i>                                                         | (Required) Number of fill patterns                |
| <b>Fills</b>              | Array <i>&lt;PRC_TYPE_GRAPH_FillPattern&gt;</i> [ <b>fill_count</b> ]          | (Required) Array of fill patterns                 |
| <b>ref_coord_count</b>    | <i>UnsignedInteger</i>                                                         | (Required) Number of reference coordinate systems |
| <b>ref_coords</b>         | Array <i>&lt;PRC_TYPE_RI_CoordinateSystem&gt;</i> [ <b>ref_coord_count</b> ]   | (Required) Array of reference coordinate systems  |

### 8.3.5.2.2 MarkupSerializationHelper

#### 8.3.5.2.2.1 General

This Global data is composed of font information for markup. The following example shows how the global data is serialized for markup font information.

**Table 41 — MarkupSerializationHelper**

| Name                            | Data Type                                        | Data Description                                                                                                                                                                                                                                                                                                                 |
|---------------------------------|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>default_font_family_name</b> | <i>String</i>                                    | (Required) <b>default_font_family_name</b> defines the case-sensitive default font family name used for text if a given font family is not available on the computer. If this default font family itself is not available on the computer, the font will be MyriadPro from Adobe Systems, Inc.                                   |
| <b>font_keys_count</b>          | <i>UnsignedInteger</i>                           | (Required) Number of fonts                                                                                                                                                                                                                                                                                                       |
| <b>font_keys_of_font</b>        | Array<br><FontKeysSameFont><br>[font_keys_count] | (Required) <b>font_keys_of_font</b> represents several font keys sharing the same base font name. Indices for font keys used in <i>ContentMarkupTess</i> are calculated from this array. For example, if there are two font names representing 4 and 5 font keys respectively, index #7 would be represented by font_keys[1][3]. |

**8.3.5.2.3 FontKeysSameFont**

This type describes a list of usages of the same font (referred to by its name) with different metrics and attributes.

— **attributes** represents the font attributes, and is a combination of the values in the table below.

**Table 42 — FontKeySameFont**

| Name                 | Data Type                  | Data Description               |
|----------------------|----------------------------|--------------------------------|
| <b>font_name</b>     | <i>String</i>              | (Required) Font Name           |
| <b>character_set</b> | <i>UnsignedInteger</i>     | (Required) Character Set       |
| <b>key_count</b>     | <i>UnsignedInteger</i>     | (Required) Number-of-font-keys |
| <b>font_key_list</b> | Array<FontKey> [key_count] | (Required)                     |

**Table 43 — FontKey**

| Name                   | Data Type              | Data Description           |
|------------------------|------------------------|----------------------------|
| <b>font_size</b>       | <i>UnsignedInteger</i> | (Required) font size + 1   |
| <b>font_attributes</b> | <i>Character</i>       | (Required) Font Attributes |

The following table contains the possible values for the **Character Set**.

**Table 44 — Character Set**

| <b>Value</b> | <b>Description</b>  |
|--------------|---------------------|
| 0            | Roman               |
| 1            | Japanese            |
| 2            | Traditional Chinese |
| 3            | Korean              |
| 4            | Arabic              |
| 5            | Hebrew              |
| 6            | Greek               |
| 7            | Cyrillic            |
| 8            | RightLeft           |
| 9            | Devanagari          |
| 10           | Gurmukhi            |
| 11           | Gujarati            |
| 12           | Oriya               |
| 13           | Bengali             |
| 14           | Tamil               |
| 15           | Telugu              |
| 16           | Kannada             |
| 17           | Malayalam           |
| 18           | Sinhalese           |
| 19           | Burmese             |
| 20           | Khmer               |
| 21           | Thai                |
| 22           | Laotian             |
| 23           | Georgian            |
| 24           | Armenian            |
| 25           | Simplified Chinese  |
| 26           | Tibetan             |
| 27           | Mongolian           |
| 28           | Geez                |
| 29           | EastEuropeanRoman   |
| 30           | Vietnamese          |
| 31           | ExtendedArabic      |

**font\_attributes** represents the font attributes, and is a combination of the following values.

**Table 45 — Font attributes**

| Font attribute value | Description                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2                    | Bold                                                                                                                                                          |
| 4                    | Italic                                                                                                                                                        |
| 8                    | Underlined                                                                                                                                                    |
| 16                   | Strike-Out                                                                                                                                                    |
| 32                   | Overlined                                                                                                                                                     |
| 64                   | Stretch. In case the font to be used is not the original font, this attribute value indicates that the text shall be stretched to fit within its bounding box |
| 128                  | Wire. This attribute value indicates that the original font is a wireframe font.                                                                              |

**8.3.5.2.4 RgbColor**

Color definition with 3 components.

**Table 46 — RgbColor**

| Name  | Data Type | Data Description |
|-------|-----------|------------------|
| Red   | Double    | (Required) Red   |
| Green | Double    | (Required) Green |
| Blue  | Double    | (Required) Blue  |

**8.3.6 PRC\_TYPE\_ASM\_FileStructureTree**

Each FileStructure within a PRC file has a FileStructureTree which defines

- the number of parts and part data
- the number of product occurrences and product occurrence data

within this FileStructure.

By convention, the data within PRC Files are ordered so that data is defined before it is referenced. In the case of a FileStructureTree, part data is defined before product occurrence data which may refer to it, but within the array of part data, the order of the parts is immaterial. This is not the case for product occurrence data.

Product occurrence data represent an assembly, with a single product occurrence being the root. Each product occurrence (except the root) may be referenced by only one product occurrence. However, each product occurrence may refer to multiple product occurrences. The array of product occurrence data should be ordered so that any product occurrence is defined before the product occurrence referring to it.

The FileStructureInternalData defines the index of the root product occurrence and the next available index available to use when assigning unique index (identifiers) to referenceable entities within the FileStructure.

**Table 47 — PRC\_TYPE\_ASM\_FileStructureTree**

| Name                 | Data Type                                                                  | Data Description                                       |
|----------------------|----------------------------------------------------------------------------|--------------------------------------------------------|
|                      | <i>UnsignedInteger</i>                                                     | (Required)<br><i>PRC_TYPE_ASM_FileStructureTree</i>    |
| <b>Base</b>          | <i>ContentPRCBase</i>                                                      | (Required) Base information associated with the entity |
| <b>part_count</b>    | <i>UnsignedInteger</i>                                                     | (Required) Number of part definitions                  |
| <b>Parts</b>         | Array < <i>PRC_TYPE_ASM_PartDefinition</i> >[ <b>part_count</b> ]          | (Required) An array of part definitions                |
| <b>product_count</b> | <i>UnsignedInteger</i>                                                     | (Required) Number of product occurrences               |
| <b>Products</b>      | Array<br>< <i>PRC_TYPE_ASM_ProductOccurrence</i> >[ <b>product_count</b> ] | (Required) An array of product occurrences             |
| <b>internal_data</b> | <i>PRC_TYPE_ASM_FileStructure</i>                                          | (Required)<br><i>FileStructureInternalData</i>         |
| <b>user_data</b>     | <i>UserData</i>                                                            | (Required) User defined data                           |

### 8.3.7 PRC\_TYPE\_ASM\_FileStructureTessellation

This type gathers tessellation data of a file structure (See 6.2).

**Table 48 — PRC\_TYPE\_ASM\_FileStructureTessellation**

| Name              | Data Type                                           | Data Description                                         |
|-------------------|-----------------------------------------------------|----------------------------------------------------------|
|                   | <i>UnsignedInteger</i>                              | (Required) <i>PRC_TYPE_ASM_FileStructureTessellation</i> |
| <b>base</b>       | <i>ContentPRCBase</i>                               | (Required) Base information associated with the entity   |
| <b>tess_count</b> | <i>UnsignedInteger</i>                              | (Required) Number_of_tessellations                       |
| <b>tess</b>       | Array < <i>PRC_TYPE_TESS</i> >[ <b>tess_count</b> ] | (Required) Content of all tessellations                  |
| <b>user_data</b>  | <i>UserData</i>                                     | (Required) User defined data                             |

### 8.3.8 PRC\_TYPE\_ASM\_FileStructureGeometry

#### 8.3.8.1 General

This type gathers geometry data of a file structure (See 6.2).

**Table 49 — *PRC\_TYPE\_ASM\_FileStructureGeometry***

| Name                  | Data Type                         | Data Description                                                         |
|-----------------------|-----------------------------------|--------------------------------------------------------------------------|
|                       | <i>UnsignedInteger</i>            | (Required)<br><i>PRC_TYPE_ASM_FileStructureGeometry</i>                  |
| <b>Base</b>           | <i>ContentPRCBase</i>             | (Required) Base information associated with the entity                   |
| <b>exact_geometry</b> | <i>FileStructureExactGeometry</i> | (Required) Topological context entities and their associated brep bodies |
| <b>user_data</b>      | <i>UserData</i>                   | (Required) User defined data                                             |

**8.3.8.2 *FileStructureExactGeometry***

The *FileStructureExactGeometry* section consists of an array of "topological contexts". Each "topological context" contains an array of brep bodies contained within that topological context. Every geometrical and topological entity within a *FileStructure* may only belong to a single topological context.

A pair of indices (topological context, brep body) uniquely identifies a brep body within the topological entities of a *FileStructure*.

**Table 50 — *FileStructureExactGeometry***

| Name                      | Data Type                                                           | Data Description                                                                  |
|---------------------------|---------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| <b>topo_context_count</b> | <i>UnsignedInteger</i>                                              | (Required) Number of topological contexts                                         |
| <b>topo_contexts</b>      | Array<br>< <i>TopologicalContext</i> >[ <b>topo_context_count</b> ] | (Required) Array of topological contexts together with its associated brep bodies |

**8.3.9 *PRC\_TYPE\_ASM\_FileStructureExtraGeometry*****8.3.9.1 General**

The extra geometry data is summary data pertaining to the geometry which can be used to enable partial loading of the file structure without loading the entire geometry. This type gathers summary information of the exact geometry section, by topological context.

**Table 51 — *PRC\_TYPE\_ASM\_FileStructureExtraGeometry***

| Name                    | Data Type                                                    | Data Description                                             |
|-------------------------|--------------------------------------------------------------|--------------------------------------------------------------|
|                         | <i>UnsignedInteger</i>                                       | (Required)<br><i>PRC_TYPE_ASM_FileStructureExtraGeometry</i> |
| <b>Base</b>             | <i>ContentPRCBase</i>                                        | (Required) Base information associated with the entity       |
| <b>extra_geom_count</b> | <i>UnsignedInteger</i>                                       | (Required) Number of extra geometry contexts                 |
| <b>extra_geom</b>       | Array<br>< <i>ExtraGeometry</i> >[ <b>extra_geom_count</b> ] | (Required)                                                   |
| <b>user_data</b>        | <i>UserData</i>                                              | (Required) User defined data                                 |

### 8.3.9.2 *ExactGeometry*

This is the summary of a topological context.

**Table 52 — *ExactGeometry***

| Name                | Data Type              | Data Description                 |
|---------------------|------------------------|----------------------------------|
| <b>Summary</b>      | <i>GeometrySummary</i> | (Required) Summary Data          |
| <b>context_data</b> | <i>ContextGraphics</i> | (Required) Graphics Context Data |

#### 8.3.9.2.1 *GeometrySummary*

##### 8.3.9.2.1.1 General

This describes the summary list of the bodies of the topological context.

The **number\_of\_bodies** shall be the same as the number of bodies in the context.

**Table 53 — *GeometrySummary***

| Name                    | Data Type                                                      | Data Description                                                                                 |
|-------------------------|----------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <b>number_of_bodies</b> | <i>UnsignedInteger</i>                                         | (Required) The <b>number_of_bodies</b> shall be the same as the number of bodies in the context. |
| <b>Bodies</b>           | Array<br>< <i>BodyInformation</i> >[ <b>number_of_bodies</b> ] | (Required) Graphics information specific to each body                                            |

##### 8.3.9.2.1.2 *BodyInformation*

This describes the summary information of a body.

The **body\_serial\_type** shall be the type of the body that is in the topological context or it shall be set to *PRC\_ROOT\_TYPE* if the body has no geometry. See the section 8.9.15 for details.

**Table 54 — *BodyInformation***

| Name                    | Data Type              | Data Description                                                                                                                                                                                                                                                                                                           |
|-------------------------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>body_serial_type</b> | <i>UnsignedInteger</i> | (Required)                                                                                                                                                                                                                                                                                                                 |
| <b>tolerance</b>        | <i>Double</i>          | (Optional; if <b>body_serial_type</b> is <i>PRC_TYPE_TOPO_BrepDataCompress</i> or <b>body_serial_type</b> is <i>PRC_TYPE_TOPO_SingleWireBodyCompress</i> or <b>body_serial_type</b> is <i>PRC_TYPE_TESS_3D_Compress</i> ) The <b>tolerance</b> which corresponds to the compression tolerance for the corresponding entity |

#### 8.3.9.2.2 *ContextGraphics*

##### 8.3.9.2.2.1 General

This describes the summary list of the graphical attributes of entities within the topological context.

The following loop shows how to traverse a topological context to gather *GraphicInformation*. A *GraphicInformation* is gathered as soon as there is a graphic content provided for a particular entity :

```

For (i=0; i<number_of_body; i++) {
    If (body[i] is PRC_TYPE_TOPO_BrepData) {
        For (j=0; j<body[i].number_of_connex; j++) {
            For (k=0; k<body[i].connex[j].number_of_shell; k++) {
                For (l=0; l<body[i].connex[j].shell[k].number_of_face; l++) {
                    Add_to_output(body[i].connex[j].shell[k].face[l])
                }
            }
        }
    }
}

```

**number\_of\_treat\_type** corresponds to the number of entity types for which *GraphicsInformation* is stored (currently, only *PRC\_TYPE\_TOPO\_Face* is supported, so if there are graphics on some faces, **number\_of\_treat\_type** is 1, else it is 0).

The current graphics as explained in 8.5 is reset prior to writing/reading the context graphics.

**Table 55 — ContextGraphics**

| Name                        | Data Type                                                                | Data Description                             |
|-----------------------------|--------------------------------------------------------------------------|----------------------------------------------|
| <b>number_of_treat_type</b> | <i>UnsignedInteger</i>                                                   | (Required) number_of_treat_type              |
| <b>treat_types</b>          | Array <i>&lt;GraphicsInformation&gt;</i> [ <b>number_of_treat_type</b> ] | (Required) Graphic information for each type |

#### 8.3.9.2.2.2 GraphicsInformation

This describes the particular graphics for one particular type of the topological context.

**Table 56 — GraphicsInformation**

| Name                       | Data Type                                                            | Data Description                                                                                                                                                                   |
|----------------------------|----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>element_type</b>        | <i>UnsignedInteger</i>                                               | (Required)                                                                                                                                                                         |
| <b>number_of_element</b>   | <i>UnsignedInteger</i>                                               | (Required) The <b>number_of_element</b> represents the number of elements collected during a recursive search on the Context data (including duplicated elements) as shown in 8.5. |
| <b>element_information</b> | Array <i>&lt;ElementInformation&gt;</i> [ <b>number_of_element</b> ] | (Required) Graphics Information for each Element                                                                                                                                   |

#### 8.3.9.2.2.3 ElementInformation

This describes the particular graphics for one particular element of the topological context.

**Table 57 — ElementInformation**

| Name                    | Data Type                      | Data Description                                                      |
|-------------------------|--------------------------------|-----------------------------------------------------------------------|
| <b>has_graphics</b>     | <i>Boolean</i>                 | (Required) <b>has_graphics</b> is TRUE if element[i] has graphics     |
| <b>graphic_behavior</b> | <i>ElementGraphicsBehavior</i> | (Optional; if <b>has_graphics</b> is TRUE) If element[i] has Graphics |

**8.3.9.2.2.4 ElementGraphicsBehavior**

This describes graphics for one particular element of the topological context.

**Table 58 — ElementGraphicsBehavior**

| Name                              | Data Type                             | Data Description                                                                                                                                                                                          |
|-----------------------------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>use_context</b>                | <i>Boolean</i>                        | (Required) <b>use_context</b> is TRUE if use current graphic context                                                                                                                                      |
| <b>biased_layer_index</b>         | <i>UnsignedInteger</i>                | (Optional; if <b>use_context</b> is FALSE) <b>layer_index</b> + 1 where <b>layer_index</b> represents the index in the array of layers stored in <i>FileStructureInternalGlobalData</i> .                 |
| <b>biased_index_of_line_style</b> | <i>UnsignedInteger</i>                | (Optional; if <b>use_context</b> is FALSE) <b>index_of_line_style</b> + 1 where <b>index_of_line_style</b> represents the index in the array of styles stored in <i>FileStructureInternalGlobalData</i> . |
| <b>behavior_bit_field</b>         | Array < <i>UnsignedCharacter</i> >[2] | (Optional; if <b>use_context</b> is FALSE) <b>behavior_bit_field</b><br><b>behaviour_bit_field</b> is an unsigned short integer (2 bytes) . See Section <i>ContextGraphics</i> for details.               |

**8.3.10 PRC\_TYPE\_ASM\_ProductOccurrence****8.3.10.1 General**

A product occurrence defines an assembly tree. In the case of a single part, the product occurrence points directly to a part definition (*PRC\_TYPE\_ASM\_PartDefinition*). In the case of a more complex assembly, a product occurrence is comprised of a list of product occurrences.

A product occurrence is comprised of the following data:

- Part definition: A pointer to the corresponding part definition. It can be null.
- Product prototype: A pointer to the corresponding product occurrence prototype. It can be null.
- External data: A pointer to the corresponding external product occurrence. It can be null.
- Children: An array of pointers to the child product occurrences.

The product prototype is the product occurrence of a subassembly or part to be used in the parent assembly. This prototype acts as a template for a given product occurrence, and lets you link to information inside the subpart or assembly, such as geometry. When building assemblies from

subassemblies, the tree of children of the product occurrence is duplicated from the prototype description. For external data, the tree is only described inside the external data product occurrence. (refer to Figure 1 in Section 5.1)

When the assembly is heterogeneous (originating from different CAD systems), the link is specified through the external data rather than the prototype.

A product occurrence has at most one parent.

**Table 59 — PRC\_TYPE\_ASM\_ProductOccurrence**

| Name                                 | Data Type                                                                    | Data Description                                                                                                                                                                                              |
|--------------------------------------|------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                      | <i>UnsignedInteger</i>                                                       | (Required)<br><i>PRC_TYPE_ASM_ProductOccurrence</i>                                                                                                                                                           |
| <b>Base</b>                          | <i>PRCBaseWithGraphics</i>                                                   | (Required)                                                                                                                                                                                                    |
| <b>references_product_occurrence</b> | <i>ReferencesOfProductOccurrence</i>                                         | (Required)                                                                                                                                                                                                    |
| <b>product_behavior</b>              | <i>Character</i>                                                             | (Required)<br><b>product_behavior</b> represents the various flags for the product. In this version of PRC, only <i>PRC_PRODUCT_BEHAVIOUR_SUPPRESSED</i> == 0x01 is used. The other flags should be set to 0. |
| <b>product_information</b>           | <i>ProductInformation</i>                                                    | (Required)                                                                                                                                                                                                    |
| <b>has_transform</b>                 | <i>Boolean</i>                                                               | (Required) TRUE if there is a transformation; else FALSE                                                                                                                                                      |
| <b>Location</b>                      | <i>Transformation</i>                                                        | (Optional; if has_transform is TRUE) <b>location</b> transforms entities in the product occurrence to its parent space.                                                                                       |
| <b>entity_ref_count</b>              | <i>UnsignedInteger</i>                                                       | (Required) Number of references                                                                                                                                                                               |
| <b>entity_reference</b>              | Array<br>< <i>PRC_TYPE_MISC_EntityReference</i> >[ <b>entity_ref_count</b> ] | (Required)<br>entity_reference is the referenced entities with possible modifiers towards their nominal definition, which may include location, color, and visibility.                                        |

Table 59 (continued)

| Name                              | Data Type                                                              | Data Description                                                                                                                                              |
|-----------------------------------|------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Markups</b>                    | <i>MarkupData</i>                                                      | (Required) <b>markups</b> represents the markups of this product occurrence (as opposed to part definition markups); these are grouped into annotations.      |
| <b>number_of_views</b>            | <i>UnsignedInteger</i>                                                 | (Required) Number of <b>views</b>                                                                                                                             |
| <b>Views</b>                      | Array <PRC_TYPE_MKP_View>[number_of_views]                             | (Required) <b>views</b> represents the views which can contain annotations or specific display parameters.                                                    |
| <b>has_filter</b>                 | <i>Boolean</i>                                                         | (Required) TRUE if product occurrence has <b>entity_filter</b> ; else FALSE                                                                                   |
| <b>entity_filter</b>              | <i>PRC_TYPE_ASM_Filter</i>                                             | (Optional; if <b>has_filter</b> is TRUE) <b>entity_filter</b> is a specific filter applied when loading data from product prototypes denoting sub-assemblies. |
| <b>number_of_display_filters</b>  | <i>UnsignedInteger</i>                                                 | (Required) Number of <b>display_filters</b>                                                                                                                   |
| <b>display_filters</b>            | Array<br><PRC_TYPE_ASM_Filter>[number_of_display_filters]              | (Required) <b>display_filters</b> represents the filters to use for display. Several filters can be specified, but only one can be active (see 8.3.12).       |
| <b>number_of_scene_parameters</b> | <i>UnsignedInteger</i>                                                 | (Required) Number of <b>scene_display_parameters</b>                                                                                                          |
| <b>scene_display_parameters</b>   | Array<br><PRC_TYPE_GRAPH_SceneDisplayParameter>[number_of_scene_param] | (Required) <b>scene_display_parameters</b> is reserved for future use.                                                                                        |
| <b>user_data</b>                  | <i>UserData</i>                                                        | (Required) User defined data                                                                                                                                  |

### 8.3.10.2 ReferencesOfProductOccurrence

#### 8.3.10.2.1 General

For a given file structure, the product occurrences should be ordered based on the following criteria:

- A product prototype in the same file structure should be stored before any occurrences that use it.

- External data in the same file structure should be stored before any occurrences that use it.
- A child occurrence should be stored before its parent.
- A part definition can be referenced several times in the same file structure.

(See to Figure 1 in Section 5.1.)

**Table 60 — ReferencesOfProductOccurrence**

| Name                                        | Data Type                                                                         | Data Description                                                                                                                                                          |
|---------------------------------------------|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>biased_index_part</b>                    | <i>UnsignedInteger</i>                                                            | (Required) <b>index_part</b> + 1 where <b>index_part</b> represents the index of the part definition in the array of part definitions of the same <i>FileStructure</i> .  |
| <b>biased_index_prototype</b>               | <i>UnsignedInteger</i>                                                            | (Required) <b>index_prototype</b> + 1 where <b>index_prototype</b> represents the index of the product prototype in the <i>FileStructure</i> product occurrences array.   |
| <b>prototype_in_same_file_structure</b>     | <i>FileIdentifier</i>                                                             | (Optional; if <b>index_prototype</b> is NOT -1) <b>prototype_in_same_file_structure</b> indicates whether the prototype is in the same <i>FileStructure</i> .             |
| <b>biased_index_external_data</b>           | <i>UnsignedInteger</i>                                                            | (Required) <b>index_external_data</b> + 1 where <b>index_external_data</b> represents the index of the external data.                                                     |
| <b>external_data_in_same_file_structure</b> | <i>FileIdentifier</i>                                                             | (Optional; if <b>index_external_data</b> is NOT -1) <b>external_data_in_same_file_structure</b> indicates whether the external data is in the same <i>FileStructure</i> . |
| <b>number_of_child_product_occurrences</b>  | <i>UnsignedInteger</i>                                                            | (Required) Number of child product occurrences                                                                                                                            |
| <b>index_child_occurrence</b>               | Array<br>< <i>UnsignedInteger</i> >[ <b>number_of_child_product_occurrences</b> ] | (Required) <b>index_child_occurrence</b> , which is mandatory in the same file structure, represents the index of the child product occurrence.                           |

### 8.3.10.2.2 *FileIdentifier*

This is the identifier of the *FileStructure* that the prototype or external data if it is different from the *FileStructure* that the product occurrence lies in.

**Table 61 — FileIdentifier**

| Name             | Data Type                 | Data Description                                                                                |
|------------------|---------------------------|-------------------------------------------------------------------------------------------------|
| <b>Flag</b>      | <i>Boolean</i>            | (Required) <b>flag</b> is TRUE if the entity exists in the same <i>FileStructure</i> ;          |
| <b>unique_id</b> | <i>CompressedUniqueld</i> | (Optional; if flag is FALSE) the identifier of the <i>FileStructure</i> that the entity lies in |

### 8.3.10.3 ProductInformation

#### 8.3.10.3.1 General

This is used to save general information associated with a product occurrence.

**Table 62 — ProductInformation**

| Name                             | Data Type        | Data Description                                                                                  |
|----------------------------------|------------------|---------------------------------------------------------------------------------------------------|
| <b>unit_from_CAD_file</b>        | <i>Boolean</i>   | (Required) <b>unit_from_CAD_file</b> indicates whether the unit is read from the native CAD file. |
| <b>Unit</b>                      | <i>Double</i>    | (Required) <b>unit</b> represents the units in mm.                                                |
| <b>product_information_flags</b> | <i>Character</i> | (Required) <b>product_information_flags</b> is described in <i>PRCProductFlag</i> .               |
| <b>product_load_status</b>       | <i>Integer</i>   | (Required) <b>product_load_status</b> is described in <i>EPRCProductLoadStatus</i> .              |

#### 8.3.10.3.2 PRCProductFlag

These flags represent characteristics of product occurrences.

A product occurrence can be:

- Default: The product occurrence is the default container, configuration, or view. This means that it is loaded by default in the originating CAD system.
- Internal: when used as a prototype of another product occurrence, this product occurrence does not come from a different physical file. Hence it should belong to the same file structure.
- Container: The product occurrence acts as a repository of child occurrences that do not necessarily have relationships between them. This is useful for situations where a single CAD file can correspond to a whole database of parts and assemblies.
- Configuration: This is a specific arrangement of a product with respect to its whole hierarchy. Some parts may differ or are in a different position. For example, consider the case of an automobile where the steering wheel may be either on the left or right side.
- View: A product occurrence which is a view refers to another product occurrence (its prototype) to denote a particular setting of visibilities and position within the same hierarchy.

If none of these flags is specified, a product occurrence is referred to as regular. If the product occurrence has no parent, it is similar to a configuration. A product occurrence with no parent leads to a different *FileStructure*, unless it is internal, meaning that it represents a part, subassembly, or assembly

hierarchy inside the same FileStructure. If the flag is PRC\_PRODUCT\_FLAG\_REGULAR or PRC\_PRODUCT\_FLAG\_CONTAINER all other flags are ignored. Otherwise, the flags are legal in any combination.

**Table 63 — PRCProductFlag**

| Value | Type Name                  |
|-------|----------------------------|
| 0x00  | PRC_PRODUCT_FLAG_REGULAR   |
| 0x01  | PRC_PRODUCT_FLAG_DEFAULT   |
| 0x02  | PRC_PRODUCT_FLAG_INTERNAL  |
| 0x04  | PRC_PRODUCT_FLAG_CONTAINER |
| 0x08  | PRC_PRODUCT_FLAG_CONFIG    |
| 0x10  | PRC_PRODUCT_FLAG_VIEW      |

### 8.3.10.3 EPRCProductLoadStatus

This represents the status of a loading of a product occurrence that may exist in a different file. This status represents the availability of that external resource at the time the PRC file was created.

**Table 64 — EPRCProductLoadStatus**

| Value | Type Name                          | Description                                                               |
|-------|------------------------------------|---------------------------------------------------------------------------|
| 0     | KEPRCProductLoadStatus_Error       | Unknown status                                                            |
| 1     | KEPRCProductLoadStatus_NotLoaded   | Loading error. For example, there is a missing file                       |
| 2     | KEPRCProductLoadStatus_NotLoadable | Not loadable. For example, something prevents the file from being loaded. |
| 3     | KEPRCProductLoadStatus_Loaded      | The product was successfully loaded                                       |

### 8.3.10.4 MarkupData

This is all the data that are related to Markup for a part or a product.

**Table 65 — MarkupData**

| Name                          | Data Type                                                                  | Data Description                 |
|-------------------------------|----------------------------------------------------------------------------|----------------------------------|
| <b>number_of_linked_items</b> | <i>UnsignedInteger</i>                                                     | (Required)                       |
| <b>linked_items</b>           | Array<br><PRC_TYPE_MISC_MarkupLinkedItem>[ <b>number_of_linked_items</b> ] | (Required) array of linked items |
| <b>number_of_leaders</b>      | <i>UnsignedInteger</i>                                                     | (Required)                       |
| <b>Leaders</b>                | Array<br><PRC_TYPE_MKP_Leader>[ <b>number_of_leaders</b> ]                 | (Required) array of leaders      |
| <b>number_of_markups</b>      | <i>UnsignedInteger</i>                                                     | (Required)                       |

Table 65 (continued)

|                                      |                                                                                |                                         |
|--------------------------------------|--------------------------------------------------------------------------------|-----------------------------------------|
| <b>Markups</b>                       | ArrayOf[ <i>PRC_TYPE_MKP_Markup</i> ]                                          | (Required) array of markups             |
| <b>number_of_annotation_entities</b> | <i>UnsignedInteger</i>                                                         | (Required)                              |
| <b>annotation_entities</b>           | Array<br>< <i>AnnotationEntities</i> >[ <b>number_of_annotation_entities</b> ] | (Required) array of annotation entities |

### 8.3.10.5 AnnotationEntities

An annotation entity can be a:

*PRC\_TYPE\_MKP\_AnnotationItem*

*PRC\_TYPE\_MKP\_AnnotationSet*

*PRC\_TYPE\_MKP\_AnnotationReference*

### 8.3.11 PRC\_TYPE\_ASM\_PartDefinition

This represents a part definition.

A part consists of:

- A **Bounding\_box**
- **views** represents the views which can contain annotations or specific display parameters.

Table 66 — *PRC\_TYPE\_ASM\_PartDefinition*

| Name                                  | Data Type                                                                | Data Description                                                                            |
|---------------------------------------|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
|                                       | <i>UnsignedInteger</i>                                                   | (Required)<br><i>PRC_TYPE_ASM_PartDefinition</i>                                            |
| <b>Base</b>                           | <i>PRC_TYPE_ROOT_PRCBaseWithGraphics</i>                                 | (Required)                                                                                  |
| <b>bounding_box</b>                   | <i>BoundingBox</i>                                                       | (Required)                                                                                  |
| <b>number_of_representation_items</b> | <i>UnsignedInteger</i>                                                   | (Required)                                                                                  |
| <b>representation_items</b>           | Array<br>< <i>PRC_TYPE_RI</i> >[ <b>number_of_representation_items</b> ] | (Required) A collection of visible <b>representation_items</b> containing geometrical data; |

Table 66 (continued)

| Name                   | Data Type                                                       | Data Description                                                                                                                                           |
|------------------------|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Markups</b>         | <i>MarkupData</i>                                               | (Required) <b>markups</b> representing the markups of this part definition (as opposed to product occurrence markups); these are grouped into annotations. |
| <b>number_of_views</b> | <i>UnsignedInteger</i>                                          | (Required) Number of views                                                                                                                                 |
| <b>Views</b>           | Array<br>< <i>PRC_TYPE_MKP_View</i> >[ <b>number_of_views</b> ] | (Required) Array of <b>views</b> data                                                                                                                      |
| <b>user_data</b>       | <i>UserData</i>                                                 | (Required) User defined data                                                                                                                               |

### 8.3.12 *PRC\_TYPE\_ASM\_Filter*

#### 8.3.12.1 General

This entity specifies the filtering between parts and assemblies. A filter denotes the particular usage of a subpart or product occurrence within a more complex one. It has the following purposes:

- To represent only those items that are of interest in the complex assembly.
- To configure the display accordingly.

Table 67 — *PRC\_TYPE\_ASM\_Filter*

| Name                 | Data Type                       | Data Description                                                                                                    |
|----------------------|---------------------------------|---------------------------------------------------------------------------------------------------------------------|
|                      | <i>UnsignedInteger</i>          | (Required) <i>PRC_TYPE_ASM_Filter</i>                                                                               |
| <b>Base</b>          | <i>ContentPRCBase</i>           | (Required) Base information associated with the entity                                                              |
| <b>is_active</b>     | <i>Boolean</i>                  | (Required) <b>is_active</b> : indicates whether this filter corresponds to the active layout when loading the file. |
| <b>layer_filter</b>  | <i>ContentLayerFilterItems</i>  | (Required) Layer filter                                                                                             |
| <b>entity_filter</b> | <i>ContentEntityFilterItems</i> | (Required) Entity filter                                                                                            |
| <b>user_data</b>     | <i>UserData</i>                 | (Required) User defined data                                                                                        |

#### 8.3.12.2 *ContentLayerFilterItems*

This saves information for filtering of entities by layer: only entities having certain layer specifications will keep the show status they have without the filter. All other entities will be set to no show.