



BSI Standards Publication

Industrial automation systems and integration — JT file format specification for 3D visualization

Part 3: Version 2

National foreword

This British Standard is the UK implementation of ISO 14306-3:2025. It supersedes BS ISO 14306:2017, which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee AMT/4, Industrial data and manufacturing interfaces.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2025
Published by BSI Standards Limited 2025

ISBN 978 0 539 32350 4
ICS 25.040.01

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 July 2025.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------



**International
Standard**

ISO 14306-3

**Industrial automation systems
and integration — JT file format
specification for 3D visualization —**

**Part 3:
Version 2**

**First edition
2025-06**

Reference number
ISO 14306-3:2025(en)

Full document details: BS ISO 14306-3:2025

© ISO 2025  **Documents**
Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, 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
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	vi
Introduction	viii
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	2
3.1 Terms and definitions	2
3.2 Abbreviated terms	2
4 Notational conventions	3
4.1 Diagrams and field descriptions	3
4.2 Data Types	8
4.3 Empty field	11
5 File Format	12
5.1 General File Format	12
5.2 File Structure	12
5.2.1 File Header	12
5.2.2 TOC Segment	14
5.2.3 Data Segment	16
5.3 Logical Element Header ZLIB	20
5.4 Data Segments	21
6 LSG Segment	23
6.1 LSG Data Collection	23
6.2 Graph Elements	23
6.2.1 Node Elements	24
6.2.2 Attribute Elements	49
6.3 Property Atom Elements	93
6.3.1 Base Property Atom Element	93
6.3.2 String Property Atom Element	94
6.3.3 Integer Property Atom Element	95
6.3.4 Floating Point Property Atom Element	96
6.3.5 JT Object Reference Property Atom Element	96
6.3.6 Date Property Atom Element	97
6.3.7 Late Loaded Property Atom Element	99
6.3.8 Vector4f Property Atom Element	100
6.4 Property Table	101
6.4.1 Element Property Table	102
7 Shape LOD Segment	103
7.1 General Shape Lod Segment	103
7.2 Shape LOD Element	103
7.2.1 Base Shape LOD Element	103
7.2.2 Vertex Shape LOD Element	104
7.2.3 Tri-Strip Set Shape LOD Element	119
7.2.4 Polyline Set Shape LOD Element	119
7.2.5 Point Set Shape LOD Element	120
7.2.6 Null Shape LOD Element	121
7.3 Primitive Set Shape Element	121
7.3.1 Lossless Compressed Primitive Set Data	123
7.3.2 Lossy Quantized Primitive Set Data	125

8	Geometry Segments.....	130
8.1	General Geometry Segments	130
8.2	STEP B-Rep Segment	130
8.3	XT B-Rep Segment.....	130
8.4	JT ULP Segment.....	130
8.5	JT LWPA Segment.....	130
8.6	Wireframe Segment.....	130
8.7	JT B-Rep Element (deprecated)	131
9	Meta Data Segment	132
9.1	General Meta Data Segment.....	132
9.2	Property Proxy Meta Data Element.....	132
9.2.1	Date Property Value	134
9.3	PMI Manager Meta Data Element.....	135
9.3.1	PMI Entities	138
9.3.2	PMI Associations.....	160
9.3.3	PMI User Attributes	162
9.3.4	PMI String Table	163
9.3.5	PMI Model Views	164
9.3.6	Generic PMI Entities.....	166
9.3.7	PMI CAD Tag Data.....	171
9.3.8	PMI Polygon Data	173
10	Data Compression and Encoding.....	177
10.1	General Data Compression and Encoding.....	177
10.2	Common Compression Data Collection Formats.....	177
10.2.1	Int32 Compressed Data Packet.....	177
10.2.2	Int32 Compressed Data Packet Mk. 2	183
10.2.3	Float64 Compressed Data Packet	189
10.2.4	Compressed Vertex Coordinate Array.....	193
10.2.5	Compressed Vertex Normal Array.....	194
10.2.6	Compressed Vertex Texture Coordinate Array	197
10.2.7	Compressed Vertex Colour Array	198
10.2.8	Compressed Vertex Flag Array	201
10.2.9	Point Quantizer Data.....	201
10.2.10	Texture Quantizer Data.....	202
10.2.11	Colour Quantizer Data	202
10.2.12	Uniform Quantizer Data.....	204
10.2.13	Compressed Entity List for Non-Trivial Knot Vector	204
10.2.14	Compressed Control Point Weights Data	208
10.2.15	Compressed Curve Data	209
10.2.16	Compressed CAD Tag Data	212
10.3	Encoding Algorithms.....	215
10.3.1	Uniform Data Quantization.....	215
10.3.2	Bitlength CODEC	216
10.3.3	Arithmetic CODEC	217
10.3.4	Deering Normal CODEC	222
10.4	zlib compression	224
11	Common Data Conventions and Constructs.....	225
11.1	General Data Conventions and Constructs.....	225
11.2	Late-Loading Data	225
11.3	TOC Segment Location	225
11.4	Bit Fields	225
11.5	Empty Field	225
11.6	Local version numbers.....	225
11.6.1	Version numbers	226

11.7	Hash Value.....	228
11.8	Scene graph construction.....	228
11.9	Metadata Conventions.....	229
11.9.1	CAD Properties.....	229
11.9.2	PMI Properties.....	234
11.9.3	Tessellation Properties.....	331
11.9.4	Miscellaneous Properties.....	332
11.10	LSG Attribute Accumulation Semantics.....	333
11.11	LSG Part Structure.....	334
11.12	Range LOD Node Alternative Rep Selection.....	334
11.13	Brep Face Group Associations.....	334
11.14	Handling of different states / variants in a ISO 14306 file.....	335
11.14.1	Reference Sets.....	336
11.15	Watermarks.....	337
11.15.1	Background.....	337
11.15.2	Issue – Missing specification in the ISO.....	337
11.15.3	Recommended Solution.....	337
11.15.4	Issue – Protection of data.....	339
12	Conformance requirements.....	341
Annex A (normative)	Information object registration.....	342
Annex B (informative)	Object Type Identifiers.....	343
Annex C (normative)	STEP B-Rep.....	346
Annex D (normative)	STEP schema.....	350
Annex E (informative)	Coding Algorithms – An Implementation.....	422
Annex F (informative)	Hashing – An Implementation.....	458
Annex G (informative)	Polygon Mesh Topology Coder.....	461
Annex H (informative)	XT B- Rep segment.....	480
Annex I (informative)	XT B- Rep data segment.....	486
Annex J (informative)	PMI Data Segment.....	570
Annex K (informative)	Mapping table from ISO 10303-42 to XT B-Rep.....	571
Annex L (informative)	JT B-rep Segment.....	577
Annex M (informative)	Wireframe Segment.....	604
Annex N (informative)	JT ULP Segment.....	608
Annex O (informative)	Change History.....	663
Bibliography		664

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 document 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).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

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

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data*, in collaboration with Technical Committee ISO/IEC JTC 1 *Information Technology*, Subcommittee SC 24, *Computer graphics, image processing and environmental data representation* and Technical Committee ISO/TC 171, *Document management applications*, Subcommittee SC 2, *Document file formats, EDMS systems and authenticity of information*.

This first edition cancels and replaces the second edition (ISO 14306:2017), which has been technically revised.

The main changes are as follows:

- correction of specification of TRANSMIT FILE in section H.2 which previously stated the definition to be SCH_1200000_12. This has been corrected to read SCH_1200000_12006;
- added a normative reference to ISO 14306-2;
- updated reference to STEP schema to read {1 0 14306 3 114};
- changed references to JT to read ISO 14306 where applicable.

A list of all parts in the ISO 14306 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The ISO 14306 format is an industry focused, high-performance, lightweight, flexible file format for capturing and repurposing 3D product definition data for visualization to enable collaboration and validation throughout the extended enterprise. The ISO 14306 format is streamable and contains compression for compact and efficient representation.

Some of the highlights of the ISO 14306 format include:

- built-in support for assemblies, sub-assemblies and part constructs;
- a flexible partitioning scheme, supporting single or multiple files;
- b-rep solid shape representations to provide precision to the light-weight viewing processes;
- product manufacturing information in support of paperless manufacturing initiatives;
- precise and imprecise wireframe shape representations;
- discrete purpose-built levels of detail;
- triangle sets, polygon sets, point sets, line sets and implicit primitive sets (such as cylinder, cone and sphere);
- a full array of visual attributes such as for materials, textures, lights;
- hierarchical bounding box and bounding spheres;
- data compression that allows producers of ISO 14306 files to fine tune the trade-off between compression ratio and fidelity of the data.

Beyond the data contents description of the ISO 14306 format, the overall physical structure/organization of the format is also designed to support operations such as:

- offline optimizations of the data contents, therefore file granularity and flexibility optimized to meet the needs of enterprise data translation solutions;
- asynchronous streaming of content, therefore viewing optimizations such as view frustum and occlusion culling and fixed-framerate display modes;
- layers, and layer filters.

NOTE This document is based on the JT Open version 9.5 specification, which defines a scene graph with geometry specific node and attribute support, extended to support the ISO 10303 series.

Annex A contains an identifier that conforms to ISO/IEC 8824-1. The identifier unambiguously identifies this document and the schema in an open information system. The ISO 14306 series applies the method as described by ISO 10303-1.

Industrial automation systems and integration — JT file format specification for 3D visualization —

Part 3: Version 2

1 Scope

This document defines the syntax and semantics of a file format for the 3D visualization and interrogation of lightweight geometry and product manufacturing information derived from CAD systems, using visualization software tools that do not need the full capability of a CAD system.

This document has been adopted as a 3D visualization capability in addition to the ISO 10303 series.

The ISO 10303 series are the ISO standards adopted for the engineering data exchange, sharing and long-term archiving of product definition information throughout the product lifecycle.

In this document 3D visualization is defined as the visual presentation on a screen or another media of graphical and textual 3 dimensional representations of a set of data representing an object, information or results of a computational process in order to enable decision process by a human looking at the data visualized in a medium.

The file format supports the following information:

- facet information (triangles), stored with geometry compression techniques;
- visual attributes such as lights, textures and materials;
- product manufacturing information, such as dimensions, tolerances and other attributes;
- boundary representation (b-rep) solid model shape representations. Several alternatives are available, including a representation based on the geometry standard defined in ISO 10303;
- configuration representations;
- delivery methods such as asynchronous streaming of content.

The document does not specify the implementation of, or definition of a run-time architecture for viewing or processing of the file format.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 10303-21, *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure*

ISO 10303-42, *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation*

ISO 14306-2, *Industrial automation systems and integration — JT file format specification for 3D visualization — Part 2: Vocabulary*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 14306-2 apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.2 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

Abs	Absolute Value
Bbox	Bounding Box
B-Rep	Boundary Representation
CAD	Computer Aided Design
CODEC	Coder-Decoder
GD&T	Geometric Dimensioning and Tolerancing
GUID	Globally Unique Identifier
HSV	Hue, Saturation, Value
JT	Jupiter Tessellation
LsbFirst	Least Significant Byte First
Max	Maximum
Min	Minimum
MsbFirst	Most Significant Byte First
N/A	Not Applicable
PCS	Parameter Coordinate Space
PLM	Product Lifecycle Management
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha

TOC	Table of Contents
URL	Uniform Resource Locator

4 Notational conventions

4.1 Diagrams and field descriptions

Symbolic diagrams are used to describe the structure of the ISO 14306 file, see Table 1. The symbols used in these diagrams have the following meaning:

Table 1 — Symbols

Symbol	Description
	Rectangles represent a data field of one of the standard data types.
	Folders represent a logical collection of one or more of the standard data types. This information is grouped for clarity and the basic data types that compose the group are detailed in following sections of the document.
	Rectangles with extra lines at left and the right sides corners clipped off represent information logical steps that has been compressed.
	Rectangles with the right side corners clipped off represent information that has been compressed.
	Arrows convey the ordering of the information.

The format used to title the diagram symbols is dependent upon the symbol type as follows:

Diagram “Rectangle box diagram” (standard data types) symbols are titled using a format of “Data_Type : Field_Name.” The Data_Type is an abbreviated data type symbol as defined in 3.2 Data Types. In the example shown in Figure 1 the Data_Type is “I32” (a signed 32 bit integer) and Field_Name is “Count.”

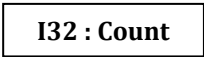


Figure 1 — Rectangle box diagram

Diagram “folder” (logical data collections) symbols are simply titled with a collection name. In the example shown in Figure 2 the collection name is “Graph Elements.”

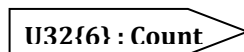
 Graph Elements
Figure 2 — Folder diagram

Diagram “rectangle box with lines at left and right sides” are simply titled with a logic step name. In the example shown in Figure 3 the logic step name is “Recover First Shell Indices”.

**Figure 3 — Rectangle box with lines at left and right sides diagram**

Diagram “rectangle box with clipped right side corners” (compressed/encoded data fields) are titled using one of the following three formats:

Data Type; followed by open brace “{”, number of bits used to store value, closed brace “}”, and a colon “:”; followed by the Field Name. This format for titling the diagram symbol indicates that the data is compressed but not encoded. The compression is achieved by using only a portion of the total bit range of the data type to store the value (for example if a count value can never be larger than the value “63” then only 6 bits are needed to store all possible count values). In the example shown in Figure 4 the Data Type is “U32”, “6” bits are used to store the value, and Field Name is “Count”.

**Figure 4 — Rectangle box with clipped right side corners**

Data Type followed by open brace “{”, compressed data packet type, “,”, Predictor Type, closed brace “}”, and a colon “:”; followed by the field name. This format for titling the diagram indicates that a vector of “Data Type” data (primal values) is ran through predictor type algorithm and the resulting output array of residual values is then compressed and encoded into a series of symbols using one of the two supported compressed data packet types.

The two supported compressed data packet types are:

Int32CDP – The Int32CDP (Int32 Compressed Data Packet) represents the format used to encode/compress a collection of data into a series of Int32 based symbols. A complete description for Int32 Compressed Data Packet can be found in 10.1.1 Int32 Compressed Data Packet.

Int32CDP2 – The Int32CDP2 (.Int32 Compressed Data Packet Mk. 2) represents a second-generation version of the above compressed data packet, and sports a simplified and more compact file layout, and the ability to more efficiently encode clustered data and bitfields. A complete description for Int32 Compressed Data Packet Mk. 2 can be found in 10.1.2 Int32 Compressed Data Packet Mk. 2.

Float64CDP – The Float64CDP (therefore Float64 Compressed Data Packet) represents the format used to encode/compress a collection of data into a series of Float64 based symbols. A complete description for Float64 Compressed Data Packet can be found in 10.1.3 Float64 Compressed Data Packet.

The Int32 Compressed Data Packet type is used for compressing/encoding both “integer” and “float” (through quantization) data. While the Float64 Compressed Data Packet type is used for compressing/encoding “double” data.

In the example shown in Figure 5 the data type is “VecU32”, Int32 Compressed Data Packet type is used, Lag1 predictor type is used, and Field Name is “First Shell Index.”

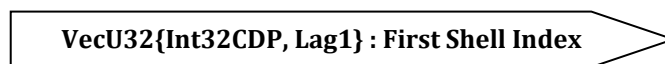


Figure 5 — Compressed data packet diagram

As mentioned above (with predictor type algorithm), the primal input data values are not always what is encoded/compressed. This is because the primal input data is first run through a predictor type algorithm, which produces an output array of residual values (therefore difference from the predicted value), and this resulting output array of residual values is the data which is actually encoded/compressed. The ISO 14306 format supports several predictor type algorithms and each use of Int32CDP or Float64CDP specifies, using the above described notation format, what predictor type algorithm is being used on the data. The ISO 14306 format supported predictor type algorithms are as follows (note that a sample implementation of decoding the predictor residual values back into the primal values can be found in Annex C).

Table 2 — Predictor Type

Predictor Type	Description
Lag1	Predicts as last value
Lag2	Predicts as value before last
Stride1	Predicts using stride from last two values
Stride2	Predicts using stride from values 2 and 4 back
StripIndex	This is a completely empirical predictor. Looks at the values two back and four back in the stream, and uses the stride between these two values to predict the current value if and only if the stride lays between -8 and 8 noninclusive, else it predicts the value as the one two back plus two. In pseudo-code form the predicted values is computed as follows: <pre> if(val2back - val4back < 8 && val2back - val4back > -8) iPredicted = val2back + (val2back - val4back); else iPredicted = val2back + 2; </pre>
Ramp	Predict value “i” as values “i’s” index
Xor1	Predict as last, but use XOR instead of subtract to compute residual
Xor2	Predict as value before last, but use XOR instead of subtract to compute residual
NULL	No prediction applied

Each predictor type, shown in Table 2, can be combined with additional processing steps, and in such case the predictor type is prefixed with “Combined:” For example, “Combined:Lag1” means that predictor type “Lag1” is combined with additional preprocessing steps. Additional description about the processing steps is provided whenever such combined predictor is used.

“Data Type : Field Name”. This format for titling the diagram symbol indicates that the data is both compressed and encoded. The data_type is an abbreviated data type symbol as defined in data types and usually represent a vector/array of data. How the data is compressed and encoded into the data type is indicated by a CODEC type and other information stored before the particular data in the file. In the example shown in Figure 6 the Data_Type is “VecU32” and Field_Name is “CodeText.”.

VecU32 : CodeText

Figure 6 — Data type : Field Name diagram

For some ISO 14306 file segment types there is ZLIB compression also applied to all bytes of element data stored in the segment. This ZLIB compression applied to all the segment’s data is not indicated in the diagrams through the use of “rectangle box with clipped right side corners”. Instead, one shall examine information stored with the first element in the file segment to determine if ZLIB compression is applied to all data in the segment. A complete description of the ISO 14306 format data compression and encoding can be found in 5.2.3 Data Segment and 10 Data Compression and Encoding.

Following each data collection diagram is detailed descriptions for each entry in the data diagram.

For rectangles this detail includes the abbreviated data type symbol, field name, verbal data description, and compression technique/algorithm where appropriate. If the data field is documented as a collection of flags, then the field is to be treated as a bit mask where the bit mask is formed by combining the flags using the binary OR operator. Each bits usage is documented, and bit ON indicates flag value is TRUE and bit OFF indicates flag value is FALSE. All bits fields that are not defined as in use should be set to “0”.

For folders (data collections), if the collection is not detailed under a sub-section of the particular document section referencing the data collection, then a comment is included following the diagram indicating where in the document the particular data collection is detailed.

If an arrow appears with a branch in its shaft, then there are two or more options for data to be stored in the file. Which data is stored will depend on information previously read from the file. The following example, shown in Figure 7, shows data field A followed by (depending on value of A) either data field B, C, or D.

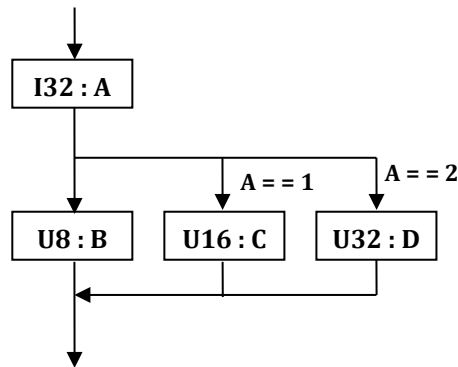


Figure 7 — Data field dependency example

In cases where the same data type repeats, a loop construct is used where the number of iterations appears next to the loop line. There are two forms of this loop construct. The first form is used when the number of iterations is not controlled by some previous read count value. Instead, the number of iterations is either a hard coded count (for example always 80 characters) or is indicated by some end-of-list marker in the data itself (thus the count is always minimum of 1). This first form of the loop construct is shown in Figure 8.

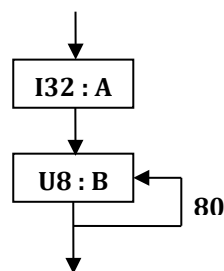


Figure 8 — Loop construct example

The second form of this loop construct is used when the number of iterations is based on data (for example count) previously read from the file. In this case it is valid for there to be zero data iterations (zero count). This second form of the loop construct, shown in Figure 9, looks as follows (data field D is repeated C value times).

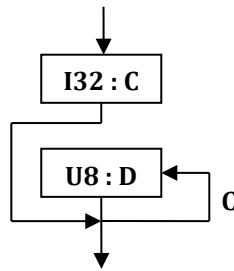


Figure 9 — Loop construct with iterations example

4.2 Data Types

The data types that can occur in the ISO 14306 binary files are listed in the following two tables, Table 3 and Table 4.

Table 3 lists the basic/standard data types which can occur in ISO 14306 file.

Table 3 — Basic Data Types

Type	Description
UChar	An unsigned 8-bit byte.
U8	An unsigned 8-bit integer value.
U16	An unsigned 16-bit integer value.
U32	An unsigned 32-bit integer value.
U64	An unsigned 64-bit integer value.
I16	A signed two's complement 16-bit integer value.
I32	A signed two's complement 32-bit integer value.
I64	A signed two's complement 64-bit integer value.
F32	An IEEE 32-bit floating point number.
F64	An IEEE 64-bit double precision floating point number

Table 4 lists some composite data types which are used to represent some frequently occurring groupings of the basic data types (for example Vector, RGBA colour). The composite data types are defined in this reference simply for convenience/brevity in describing the ISO 14306 file contents.

Table 4 — Composite Data Types

Type	Description	Symbolic Diagram
BBoxF32	The BBoxF32 type defines a bounding box using two CoordF32 types to store the XYZ coordinates for the bounding box minimum and maximum corner points.	
CoordF32	The CoordF32 type defines X, Y, Z coordinate values. So a CoordF32 is made up of three F32 base types.	
CoordF64	The CoordF64 type defines X, Y, Z coordinate values. So a CoordF64 is made up of three F64 base types.	
DirF32	The DirF32 type defines X, Y, Z components of a direction vector. So a DirF32 is made up of three F32 base types.	
GUID	The GUID type is a 16 byte (128-bit) number. GUID is stored/written to the ISO 14306 file using a four-byte word (U32), 2 two-byte words (U16), and 8 one-byte words (U8) such as: {3F2504E0-4F89-11D3-9A-0C-03-05-E8-2C-33-01} In the ISO 14306 format GUIDs are used as unique identifiers (for example Data Segment ID, Object Type ID, etc.)	
HCoordF32	The HCoordF32 type defines X, Y, Z, W homogeneous coordinate values. So an HCoordF32 is made up of four F32 base types.	
HCoordF64	The HCoordF64 type defines X, Y, Z, W homogeneous coordinate values. So an HCoordF64 is made up of four F64 base types	
MbString	The MbString type starts with an I32 that defines the number of characters (NumChar) the string contains. The number of bytes of character data is "2 * NumChar" (therefore the strings are written out as multi-byte characters where each character is U16 size).	
Mx4F32	Defines a 4-by-4 matrix of F32 values for a total of 16 F32 values. The values are stored in row major order (right most subscript, column varies fastest), that is, the first 4 elements form the first row of the matrix.	

PlaneF32	The PlaneF32 type defines a geometric Plane using the General Form of the plane equation ($Ax + By + Cz + D = 0$). The PlaneF32 type is made up of four F32 base types where the first three F32 define the plane unit normal vector (A, B, C) and the last F32 defines the negated perpendicular distance (D), along normal vector, from the origin to the plane.	
Quaternion	The Quaternion type defines a 3-dimensional orientation (no translation) in quaternion linear combination form ($a + bi + cj + dk$) where the four scalar values (a, b, c, d) are associated with the 4 dimensions of a quaternion (1 real dimension, and 3 imaginary dimensions). So the Quaternion type is made up of four F32 base types.	
RGB	The RGB type defines a colour composed of Red, Green, Blue components, each of which is a F32. So a RGB type is made up of three F32 base types. The Red, Green, Blue colour values typically range from 0.0 to 1.0.	
RGBA	The RGBA type defines a colour composed of Red, Green, Blue, Alpha components, each of which is a F32. So a RGBA type is made up of four F32 base types. The Red, Green, Blue colour values typically range from 0.0 to 1.0. The Alpha value ranges from 0.0 to 1.0 where 1.0 indicates completely opaque.	
String	The String type starts with an I32 that defines the number of characters (NumChar) the string contains. The number of bytes of character data is "NumChar" (therefore the strings are written out as single-byte characters where each character is U8 size).	
VecF32	The VecF32 type defines a vector/array of F32 base type. The type starts with an I32 that defines the count of following F32 base type data. So a VecF32 is made up of one I32 followed by that number of F32. Note that it is valid for the I32 count number to be equal to "0", indicating no following F32.	
VecF64	The VecF64 type defines a vector/array of F64 base type. The type starts with an I32 that defines the count of following F64 base type data. So a VecF64 is made up of one I32 followed by that number of F64. Note that it is valid for the I32 count number to be equal to "0", indicating no following F64.	
VecI32	The VecI32 type defines a vector/array of I32 base type. The type starts with an I32 that defines the count of following I32 base type data. So a VecI32 is made up of one I32 followed by that number of I32. Note that it is valid for the I32 count number to be equal to "0", indicating no following I32.	

VecU32	The VecU32 type defines a vector/array of U32 base type. The type starts with an I32 that defines the count of following U32 base type data. So a VecU32 is made up of one I32 followed by that number of U32. Note that it is valid for the I32 count number to be equal to “0”, indicating no following U32.	
--------	--	--

4.3 Empty field

When writing an ISO 14306 file whose data did not originate from reading a previous ISO 14306 file, an empty field should be set to a value of “0”.

When writing an ISO 14306 file whose data originated from reading a previous ISO 14306 file (therefore rewriting an ISO 14306 File), empty fields should be written with the same value that was read from the originating ISO 14306 file.

5 File Format

5.1 General File Format

All objects represented in the ISO 14306 format are assigned an “object identifier” (for example see 6.2.1.1.1 Base Node Data, or 6.2.2.1.1 Base Attribute Data) and all references from one object to another object are represented in the ISO 14306 format using the referenced object’s “object identifier”. It is the responsibility of ISO 14306 format readers/writers to maintain the integrity of these object references by doing appropriate pointer unswizzling/swizzling as ISO 14306 format data is read into memory or written out to disk. Where “pointer swizzling” refers to the process of converting references based on object identifiers into direct memory pointer references and “pointer unswizzling” is the reverse operation (therefore replacing references based on memory pointers with object identifier references).

All objects stored in an ISO 14306 file are classified by type and thus include an object type identifier as part of their persisted data.

Object Type Identifiers specifications are defined in Annex B.

5.2 File Structure

An ISO 14306 file is structured as a sequence of blocks/segments, see Figure 10. The File Header block is always the first block of data in the file. The File Header is followed (in no particular order) by a TOC Segment and a series of other Data Segments. The two Data Segments which shall always exist to have a reference compliant ISO 14306 file are the LSG Segment and one Shape LOD Segment. The TOC Segment is located within the file using data stored in the File Header. Within the TOC Segment is information that locates all other Data Segments within the file.

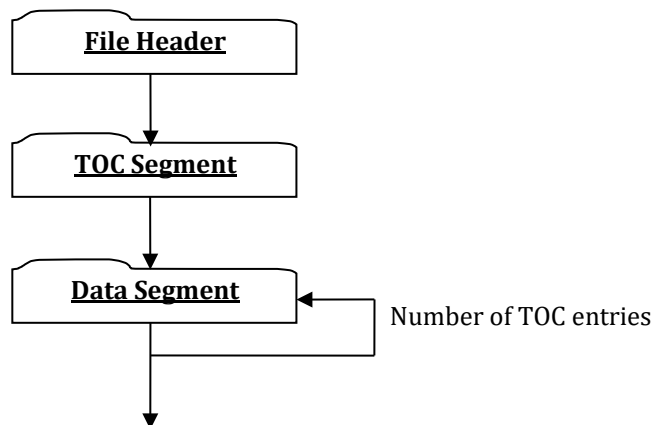


Figure 10 — ISO 14306 File Structure

5.2.1 File Header

The File Header is always the first block of data in an ISO 14306 file. The File Header contains information about the ISO 14306 file version and TOC location, which Loaders use to determine how to read the file. The exact contents of the File Header, as shown in Figure 11, are as follows:

UChar : Version

Version M.n Comment

The version string is padded with spaces to a length of 75 ASCII characters and then the final five characters shall be filled with the following linefeed and carriage return character combination (shown using c-style syntax):

```
Version[75] = ''
Version[76] = '\n'
Version[77] = '\r'
Version[78] = '\n'
Version[79] = ''
```

As an example, for an ISO 14306 Version 9.5 file written by a Direct Model (DM) library version 7.3.4.0 this string will look as follows:

“Version 9.5 JT DM 7.3.4.0 \n\r\n “

UChar : Byte Order

Defines the file byte order and thus can be used by the loader to determine if there is a mismatch (thus byte swapping required) between the file byte order and the machine (on which the loader is being run) byte order. Valid values for Byte Order are:

0 – Least Significant byte first (LsbFirst)

1 – Most Significant byte first (MsbFirst)

I32 : Empty Field

Refer to best practice in 11.4.

I32 : TOC Offset

Defines the byte offset from the top of the file to the start of the TOC Segment.

GUID : LSG Segment ID

LSG Segment ID specifies the globally unique identifier for the Logical Scene Graph Data Segment in the file. This ID along with the information in the TOC Segment can be used to locate the start of LSG Data Segment in the file. This ID is needed because without it a loader would have no way of knowing the location of the root LSG Data Segment. All other Data Segments shall be accessible from the root LSG Data Segment.

GUID: Empty Field

Refer to best practice in 11.4.

5.2.2 TOC Segment

The TOC Segment, shown in Figure 12, contains information identifying and locating all individually addressable Data Segments within the file. A TOC Segment is always required to exist somewhere within an ISO 14306 file. The actual location of the TOC Segment within the file is specified by the File Header segment's "TOC Offset" field. The TOC Segment contains one TOC Entry for each individually addressable Data Segment in the file.

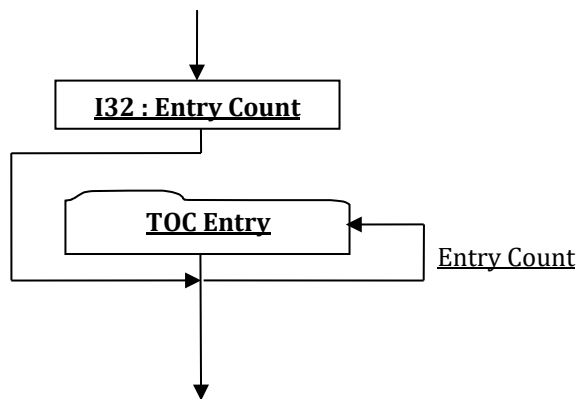


Figure 12 — TOC Segment data collection

I32 : Entry Count

Entry Count is the number of entries in the TOC.

TOC Entry

Each TOC Entry, as shown in Figure 13, represents a Data Segment within the ISO 14306 File. The essential function of a TOC Entry is to map a Segment ID to an absolute byte offset within the file.

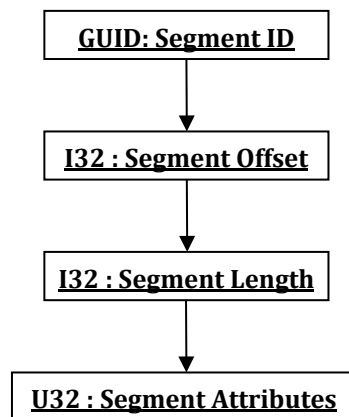


Figure 13 — TOC Entry data collection

GUID : Segment ID

Segment ID is the globally unique identifier for the segment.

I32 : Segment Offset

Segment Offset defines the byte offset from the top of the file to start of the segment.

I32 : Segment Length

Segment Length is the total size of the segment in bytes.

U32 : Segment Attributes

Segment Attributes, as shown in Table 5, is a collection of segment information encoded within a single U32 using the following bit allocation.

Table 5 — Segment attributes

Bits 0 - 23	Reserved for future use.
Bits 24 - 31	Segment type. Complete list of Segment types can be found in Table 6 — Segment Types.

5.2.3 Data Segment

All data stored in an ISO 14306 file shall be defined within a Data Segment. Data Segments are “typed” based on the general classification of data they contain. See Segment Type field description below for a complete list of the segment types.

Beyond specific data field compression/encoding, some Data Segment types also have a ZLIB compression conditionally applied to all the Data bytes of information persisted within the segment. Whether ZLIB compression is conditionally applied to a segment’s Data bytes of information is indicated by information stored with the first “Element” in the segment. Also Table 6 — Segment Types has a column indicating whether the Segment Type may have ZLIB compression applied to its Data bytes.

All Data Segments have the same basic structure, as shown in Figure 14.

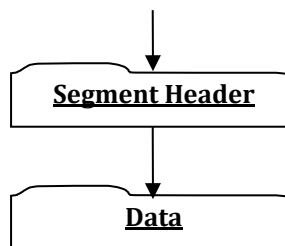


Figure 14 — Data Segment data collection

5.2.3.1 Segment Header

Segment Header, as shown in Figure 15, contains information that determines how the remainder of the Segment is interpreted by the loader.

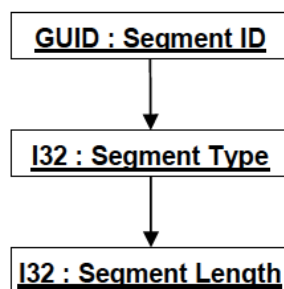


Figure 15 — Segment Header data collection

GUID : Segment ID

Global Unique Identifier for the segment.

I32 : Segment Type

Segment Type defines a broad classification of the segment contents. For example, a Segment Type of “1” denotes that the segment contains Logical Scene Graph material; “2” denotes contents of a B-Rep, etc.

The complete list of segment types, as shown in Table 6, is as follows. The column labelled "ZLIB Applied?" denotes whether ZLIB compression is conditionally applied to the entirety of the segment's Data payload. The column Mandatory status specifies whether the segment shall be present in the file according to the conformance clause.

Table 6 — Segment Types

Type	Data Contents	ZLIB Applied?	Mandatory status
1	Logical Scene Graph	Yes	Mandatory
2	B-Rep description in JT	Yes	Deprecated
3	PMI Data	Yes	Deprecated
4	Meta Data and PMI	Yes	Optional
6	Shape	No	Mandatory
7	Shape LOD0	No	Optional
8	Shape LOD1	No	Optional
9	Shape LOD2	No	Optional
10	Shape LOD3	No	Optional
11	Shape LOD4	No	Optional
12	Shape LOD5	No	Optional
13	Shape LOD6	No	Optional
14	Shape LOD7	No	Optional
15	Shape LOD8	No	Optional
16	Shape LOD9	No	Optional
17	XT B-Rep	Yes	Optional
18	Wireframe Representation	Yes	Optional
20	ULP	Yes	Optional
24	LWPA	Yes	Optional
32	STEP B-rep	Yes	Optional

NOTE 1 Segment Types 7-16 all identify the contents as LOD Shape data, where the increasing type number is intended to convey some notion of how high an LOD the specific shape segment represents. The lower the type in this 7-16 range the more detailed the Shape LOD (therefore Segment Type 7 is the most detailed Shape LOD Segment). For the rare case when there are more than 10 LODs, LOD9 and greater are all assigned Segment Type 16.

NOTE 2 The more generic Shape Segment type (therefore Segment Type 6) is used when the Shape Segment has one or more of the following characteristics:

- not a descendant of an LOD node;
- is referenced by (therefore is a child of) more than one LOD node;
- Shape has its own built-in LODs;
- no way to determine what LOD a Shape Segment represents.

I32 : Segment Length

Segment Length is the total size of the segment in bytes. This length value includes all segment Data bytes plus the Segment Header bytes (therefore it is the size of the complete segment) and should be equal to the length value stored with this segment's TOC Entry.

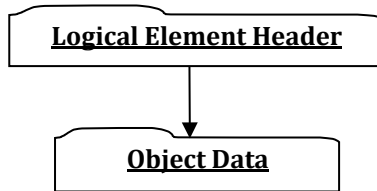
5.2.3.2 Data

The interpretation of the Data section depends on the Segment Type. See 5.4 Data Segments for a complete description for all Data Segment that may be contained in an ISO 14306 file.

Although the Data section is Segment Type dependent there is a common structure which often occurs within the Data section. This structure is a list or multiple lists of Elements where each Element has the same basic structure which consists of some fixed length header information describing the type of object contained in the Element, followed by some variable length object type specific data.

Individual data fields of an Element data collection (and its children data collections) may have advanced compression/encoding applied to them as indicated through compression related data values stored as part of the particular Element's storage format. In addition, another level of compression (therefore ZLIB compression) may be conditionally applied to all bytes of information stored for all Elements within a particular Segment. Not all Segment types support ZLIB compression on all Segment data as indicated Table 8 — Segment Types. If a particular file Segment is of the type which supports ZLIB compression on all the Segment data, whether this compression is applied or not is indicated by data values stored in the Logical Element Header ZLIB data collection of the first Element within the Segment. A diagram of each is shown in Figure 16. An in-depth description of ISO 14306 file compression/encoding techniques can be found in 10 Data Compression and Encoding.

For Segment Types that do **NOT** support ZLIB compression on all Segment Data.
(see [Table 6 — Segment Types.](#))



For Segment Types that support ZLIB compression on all Segment Data
(see [Table 6 — Segment Types.](#))

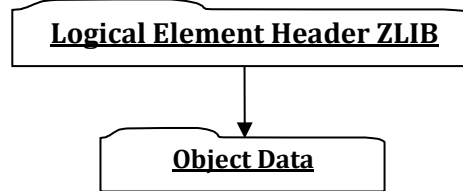


Figure 16 — Data collection

5.2.3.3 Logical Element Header

Logical Element Header, as shown in Figure 17, contains data defining the length in bytes of the Element along with the Element Header.

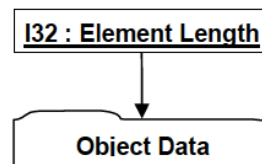


Figure 17 — Logical Element Header data collection

A complete description for Logical Element Header can be found in the Element Header.

I32 : Element Length

Element Length is the total length in bytes of the element Object Data.

Element Header

Element Header, as shown in Figure 18, contains data describing the object type contained in the Element.

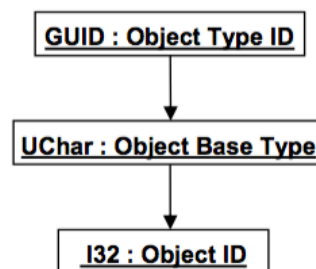


Figure 18 — Element Header data collection

GUID : Object Type ID

Object Type ID is the globally unique identifier for the object type. A complete list of the assigned GUID for all object types stored in an ISO 14306 file can be found in Annex B. If the GUID is not found in this annex, the reader should skip Element Length + 1 number of bytes.

UChar : Object Base Type

Object Base Type identifies the base object type. If the Object Base Type is not present in **Table 7 — Object Base Types** then the loader should simply skip (read pass) Element Length number of bytes.

Valid Object Base Types include the following:

Table 7 — Object Base Types

Base Type	Description	Base Type's Data Format
255	None	None
0	Base Graph Node Object	6.2.1.1.1 Base Node Data
1	Group Graph Node Object	6.2.1.3.1 Group Node Data
2	Shape Graph Node Object	6.2.1.11 Base Shape Data
3	Base Attribute Object	6.2.2.1.1 Base Attribute Data
4	Shape LOD	None
5	Base Property Object	6.3.1.1 Base Property Atom Data
6	JT Object Reference Object	6.3.5 JT Object Reference Property Atom Element without the Logical Element Header ZLIB data collection.
8	JT Late Loaded Property Object	6.3.7 Late Loaded Property Atom Element without the Logical Element Header ZLIB data collection.
9	JtBase (none)	None

I32 : Object ID

Object ID is the identifier for this Object. Other objects referencing this particular object do so using the Object ID.

5.3 Logical Element Header ZLIB

Logical Element Header ZLIB data collection, as shown in Figure 19, is the format of Element Header data used by all Elements within Segment Types that support ZLIB compression on all data in the Segment. See Table 8 — Segment Type for information on whether a particular Segment Type supports ZLIB compression on all data in the Segment.

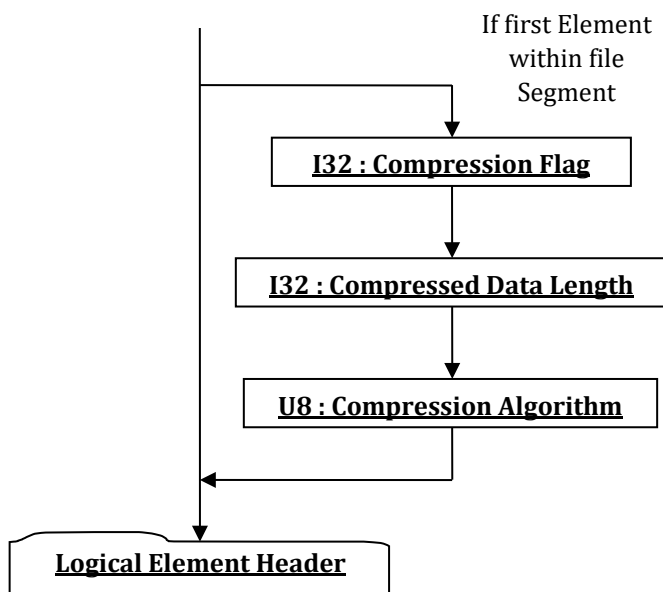


Figure 19 — Logical Element Header ZLIB data collection

A complete description for Logical Element Header can be found in Logical Element Header. Note that if Compression Flag indicates that ZLIB compression is ON for all element data in the Segment, then the Logical Element Header data collection is also compressed accordingly.

I32 : Compression Flag

Compression Flag is a flag indicating whether ZLIB compression is ON/OFF for all data elements in the file Segment. Valid values, as shown in Table 8, include the following:

Table 8 — Compression flag values

= 2	ZLIB compression is ON
!= 2	ZLIB compression is OFF.

I32 : Compressed Data Length

Compressed Data Length specifies the compressed data length in number of bytes. Note that data field Compression Algorithm is included in this count.

U8 : Compression Algorithm

Compression Algorithm specifies the compression algorithm applied to all data in the Segment. Valid values, as shown in Table 9, include the following:

Table 9 — Compression algorithm values

= 1	No compression
= 2	ZLIB compression

Object Data

The interpretation of the Object Data section depends upon the Object Type ID stored in the Logical Element Header (see Logical Element Header).

5.4 Data Segments

An ISO 14306 file consists of the following mandatory segments of data:

- LSG segment contains a collection of objects (therefore elements) connected through directed references to form a directed acyclic graph structure (therefore the LSG). The LSG is the graphical description of the model and contains graphics shapes and attributes representing the model's physical components, properties identifying arbitrary metadata (for example names, semantic roles) of those components, and a hierarchical structure expressing the component relationship.
- Shape LOD segment contains an element that defines the geometric shape definition data (for example vertices, polygons, normals, etc.) for a particular shape level of detail or alternative representation;

In addition of these mandatory segments a set of optional segments for geometry representation, PMI and meta data are available:

For geometry representation:

- STEP B-Rep segment contains an element that defines the precise geometric boundary representation data for a particular part in normative STEP Part 42 (ISO 10303-42) boundary representation format as a part 21 compressed file (ISO 10303-21). See Annex C. A mapping between STEP B-Rep segment data and XT B-Rep segment data is shown in Annex K
- JT B-Rep Segment contains an element that defines the precise geometric boundary representation data for a particular part in boundary representation format. See Annex L.
- XT B-Rep Segment contains an element that defines the precise geometric boundary representation data for a particular part in boundary representation format. See Annex H.
- ULP Segment contains an element that defines the approximate geometric boundary representation data for a particular part in JT ULP format. See Annex N.
- LWPA Segment contains an element that defines light weight precise analytic data for a particular part. More specifically LWPA contains the collection of analytic surfaces in the b-rep definition of the part. See Annex O.
- WireFrame Segment contains an element that defines the precise 3D wireframe data for a particular part. See Annex M.

For PMI and metadata:

- Meta Data and PMI Segment is used to store large collections of meta-data in separate addressable segments of the ISO 14306 file. Storing meta-data in a separate addressable segment allows references (from within the ISO 14306 file) to these segments to be constructed such that the meta-data can be late-loaded.
- In pre ISO 14306 8 version, a deprecated PMI Segment, see Annex J, was used to store PMI manager meta data element. From a parsing point of view, a PMI data segment should be treated exactly the same as a meta data and PMI segment. Reference PMI Data Segments for best practice.

6 LSG Segment

6.1 LSG Data Collection

LSG Segment, as shown in Figure 20, contains a collection of objects (Elements) connected through directed references to form a directed acyclic graph structure (therefore the LSG). The LSG is the graphical description of the model and contains graphics shapes and attributes representing the model's physical components, properties identifying arbitrary metadata (for example names, semantic roles) of those components, and a hierarchical structure expressing the component relationships. The “directed” nature of the LSG references implies that there is by default “state/attribute” inheritance from ancestor to descendant (therefore predecessor to successor).

The first Graph Element in a LSG Segment should always be a Partition Node. The LSG Segment type supports ZLIB compression on all element data, so all elements in LSG Segment use the Logical Element Header ZLIB form of element header data.

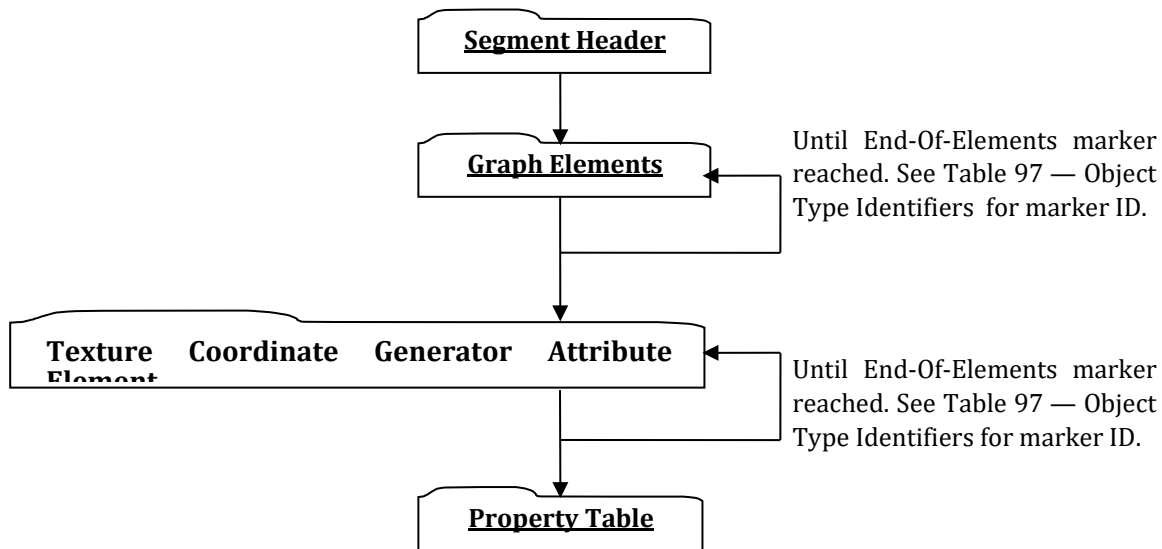


Figure 20 — LSG Segment data collection

A complete description for Segment Header can be found in 5.2.3.1.

6.2 Graph Elements

Graph Elements form the backbone of the LSG directed acyclic graph structure and in doing so serve as the ISO 14306 model's fundamental description. There are two general classifications of Graph elements, Node Elements and Attribute Elements.

Node Elements are nodes in the LSG and in general can be categorized as either an internal or leaf node. The leaf nodes are typically shape nodes used to represent a model's physical components and as such either contain or reference some graphical representation or geometry. The internal nodes define the hierarchical organization of the leaf nodes, forming both spatial and logical model relationships, and often contain or reference information (for example Attribute Elements) that is inherited down the LSG to all descendent nodes.

Attribute Elements represent graphical data (like appearance characteristics (for example colour), or positional transformations) that can be attached to a node, and inherit down the LSG.

Each of these general Graph Element classifications (therefore Node/Attribute Elements) is sub-typed into specific/concrete types based on data content and implied specialized behaviour. The following subclauses describe each of the Node and Attribute Element types.

6.2.1 Node Elements

Node Elements represent the relationships of a model's components. The model's component hierarchy is formed via certain types of Node Elements containing collections of references to other Node Elements who in turn may reference other collections of Node Elements. Node Elements are also the holders (either directly or indirectly) of geometric shape, properties, and other information defining a model's components and representations.

6.2.1.1 Base Node Element

Object Type ID: 0x10dd1035, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Base Node Element, as shown in Figure 21, represents the simplest form of a node that can exist within the LSG. The Base Node Element has no implied LSG semantic behaviour nor can it contain any children nodes.

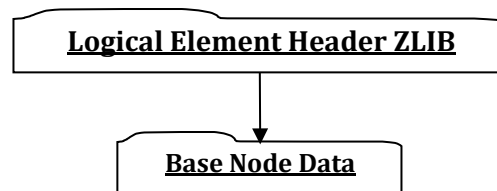


Figure 21 — Base Node Element data collection

A complete description for **Logical Element Header ZLIB** can be found in 5.3.

6.2.1.1.1 Base Node Data

A diagram of the Base Node Data is shown in Figure 22.

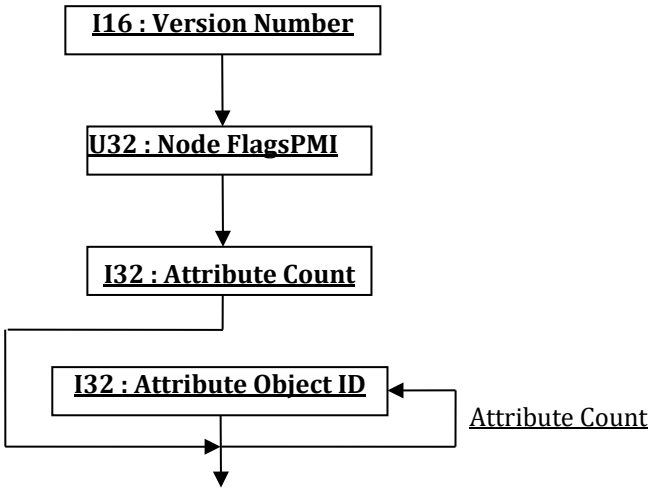


Figure 22 — Base Node Data collection

I16 : Version Number

Version Number is the version identifier for this node.

U32 : Node FlagsPMI

Node Flags, as shown in Table 10, is a collection of flags. The flags are combined using the binary OR operator. These flags store various state information of the node object. All bits fields that are not defined as in use should be set to “0”.

Table 10 — Node Flag values

0x00000001	Ignore Flag = 0 – Algorithms traversing the LSG structure should include/process this node. = 1 – Algorithms traversing the LSG structure should skip the whole subgraph rooted at this node. Essentially the traversal should be pruned.
------------	---

I32 : Attribute Count

Attribute Count indicates the number of Attribute Objects referenced by this Node Object. A node may have zero Attribute Object references.

I32 : Attribute Object ID

Attribute Object ID is the identifier for a referenced Attribute Object.

6.2.1.2 Partition Node Element

Object Type ID: 0x10dd103e, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A partition in an ISO 14306 file shall always be either the root or leaf node. A leaf partition node represents an external ISO 14306 file reference and provides a means to partition a model into multiple physical ISO 14306 files (for example separate ISO 14306 file per part in an assembly).

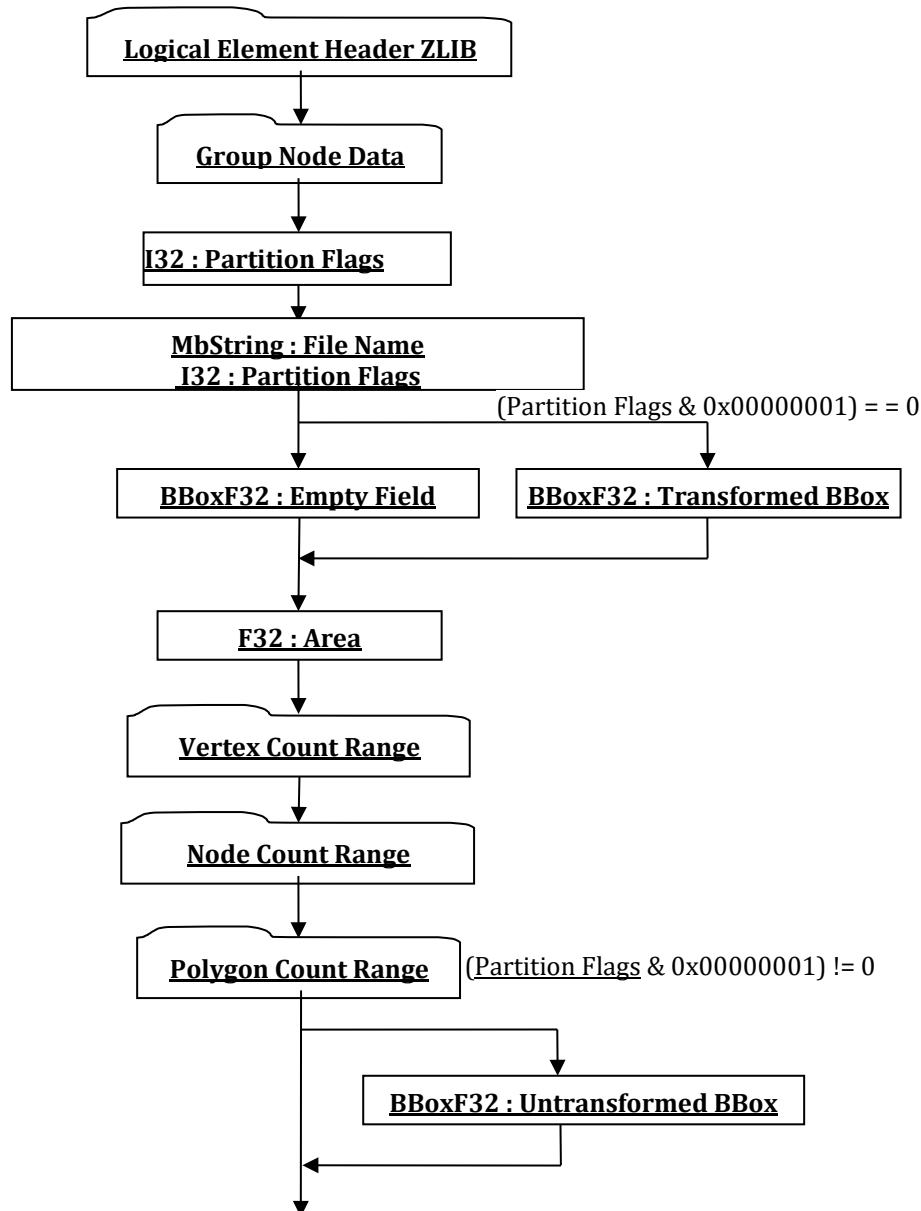


Figure 23 — Partition Node Element data collection

A complete description for **Logical Element Header ZLIB** can be found in 5.3.

I32 : Partition Flags

Partition Flags, as shown in Table 11, is a collection of flags. The flags are combined using the binary OR operator. These flags store various state information of the Partition Node Object such as indicating the presence of optional data. All bits fields that are not defined as in use should be set to “0”.

Table 11 — Partition flag bits

0x00000001	Untransformed bounding box is written.
------------	--

MbString : File Name

File Name is the relative path portion of the Partition's file location. Where "relative path" should be interpreted to mean the string contains the file name along with any additional path information that locates the partition ISO 14306 file relative to the location of the referencing ISO 14306 file.

BBoxF32 : Empty Field

Refer to best practice in 11.4.

BBoxF32 : Transformed BBox

The Transformed BBox is an MCS axis aligned bounding box and represents the transformed geometry extents for all geometry contained in the Partition Node. This bounding box information may be used by a renderer of ISO 14306 data to determine whether to load the data contained within the Partition node (therefore is any part of the bounding box within the view frustum).

F32 : Area

Area is the total surface area for this node and all of its descendants. This value is stored in MCS coordinate space (therefore values scaled by MCS scaling).

BBoxF32 : Untransformed BBox

The Untransformed BBox is only present if Bit 0x00000001 of Partition Flags data field is ON. The Untransformed BBox is an LCS axis-aligned bounding box and represents the untransformed geometry extents for all geometry contained in the Partition Node. This bounding box information may be used by a renderer of ISO 14306 data to determine whether to load the data contained within the Partition node (therefore is any part of the bounding box within the view frustum).

6.2.1.2.1 Vertex Count Range

Vertex Count Range, as shown in Figure 24, is the aggregate minimum and maximum vertex count for all descendants of the Partition Node. There is a minimum and maximum value to accommodate descendent branches having LOD nodes, which encompass a range of count values within the branch, and to accommodate nodes that can themselves generate varying representations. The minimum value represents the least vertex count that can be achieved by the Partition Node's descendants. The maximum value represents the greatest vertex count that can be achieved by the Partition Node's descendants.

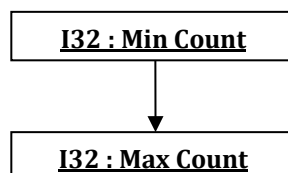


Figure 24 — Vertex Count Range data collection

I32 : Min Count

Min Count is the least vertex count that can be achieved by the Partition Node's descendants.

I32 : Max Count

Max Count is the maximum vertex count that can be achieved by the Partition Node's descendants.

6.2.1.2.2 Node Count Range

Node Count Range is the aggregate minimum and maximum count of all node descendants of the Partition Node. There is a minimum and maximum value to accommodate descendant branches having LOD nodes, which encompass a range of descendant node count values within the branch. The minimum value represents the least node count that can be achieved by the Partition Node's descendants. The maximum value represents the greatest node count that can be achieved by the Partition Node's descendants.

The data format for Node Count Range is the same as that described in 6.2.1.2.1 Vertex Count Range.

6.2.1.2.3 Polygon Count Range

Polygon Count Range is the aggregate minimum and maximum polygon count for all descendants of the Partition Node. There is a minimum and maximum value to accommodate descendent branches having LOD nodes, which encompass a range of count values within the branch, and to accommodate nodes that can themselves generate varying representations. The minimum value represents the least polygon count that can be achieved by the Partition Node's descendants. The maximum value represents the greatest polygon count that can be achieved by the Partition Node's descendants.

The data format for Polygon Count Range is the same as that described in 6.2.1.2.1 Vertex Count Range.

6.2.1.3 Group Node Element

Object Type ID: 0x10dd101b, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Group Nodes, as shown in Figure 25, contain an ordered list of references to other nodes, called the group's children. Group nodes may contain zero or more children; the children may be of any node type. Group nodes may not contain references to themselves or their ancestors.

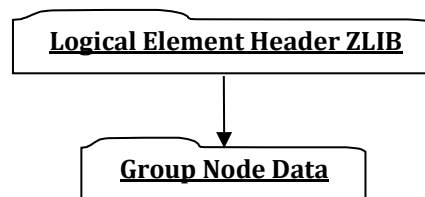


Figure 25 — Group Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.

6.2.1.3.1 Group Node Data

A diagram of the Group Node Data collection is shown in Figure 26.

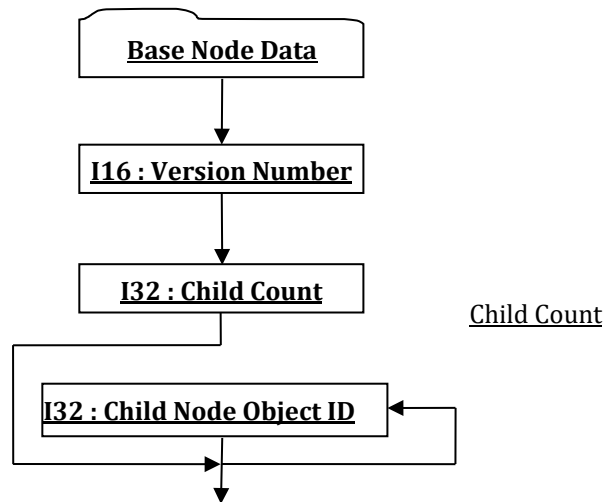


Figure 26 — Group Node Data collection

A complete description for Base Node Data can be found in 6.2.1.1.1 Base Node Data.

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

I32 : Child Count

Child Count indicates the number of child nodes for this Group Node Object. A node may have zero children.

I32 : Child Node Object ID

Child Node Object ID is the identifier for the referenced Node Object.

6.2.1.4 Instance Node Element

Object Type ID: 0x10dd102a, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

An Instance Node, as shown in Figure 27, contains a single reference to another node. Their purpose is to allow sharing of nodes and assignment of instance-specific attributes for the instanced node. Instance Nodes may not contain references to themselves or their ancestors.

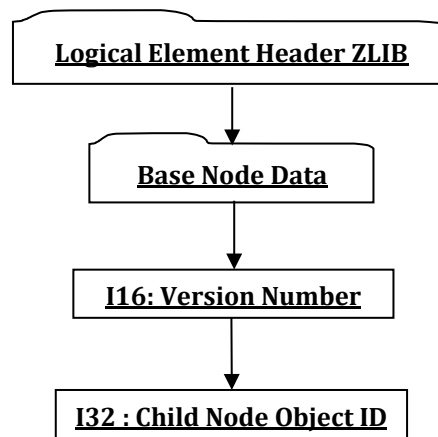


Figure 27 — Instance Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.

A complete description for Base Node Data can be found in 6.2.1.1.1 Base Node Data.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

I32 : Child Node Object ID

Child Node Object ID is the identifier for the instanced Node Object.

6.2.1.5 Part Node Element

Object Type ID: 0xce357244, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1

A Part Node Element represents the root node for a particular Part within a LSG structure. Every unique Part represented within a LSG structure should have a corresponding Part Node Element. A Part Node Element typically references (using Late Loaded Property Atoms) additional Part specific geometric data and/or properties (for example B-Rep data, PMI data).

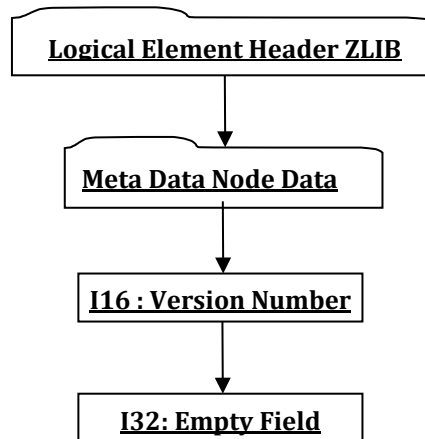


Figure 28 — Part Node Element data collection

A complete description for **Logical Element Header ZLIB** can be found in 5.3.

A complete description for Meta Data Node Data can be found in 6.2.1.6.1.

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

I32: Empty Field

Refer to best practice in 11.4.

A SUBNODE property, as shown in Figure 29 and Figure 30, can be defined on a part or assembly node in an ISO 14306 file. By convention, the node with this property is considered to be part of the parent as opposed to being a normal node entity in an assembly. SUBNODE properties can be used to represent a number of CAD constructs in ISO 14306 files including reference sets (see 11.13.1 for more information about Reference sets).

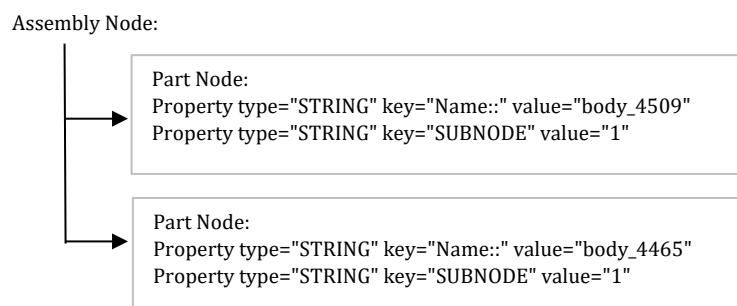


Figure 29 — Assembly node with SUBNODE

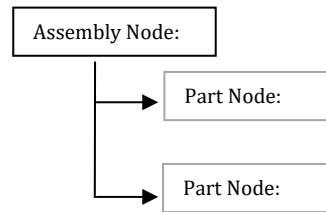


Figure 30 — Assembly node without SUBNODE

The subnode property can be used for viewing applications in order to display a different granularity of the tree, as shown in Figure 31.

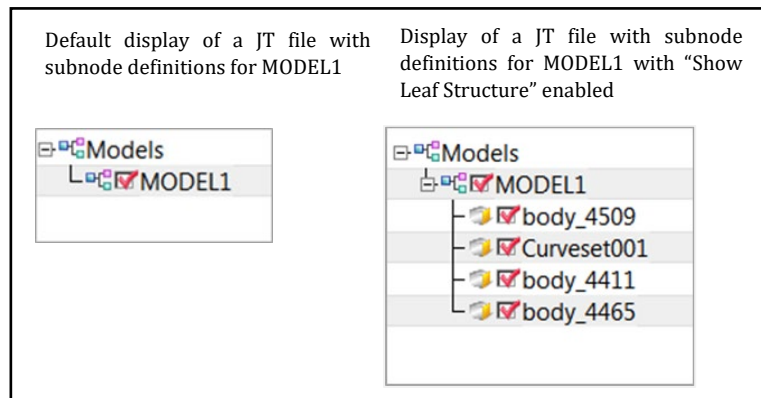


Figure 31 — Displaying Nodes that have SUBNODE properties

The value assigned to the SUBNODE property, as shown in Table 12, has no specific meaning. The convention is to set the property to a string value of “1”. Parts that contain the SUBNODE property shall contain a string property with a name for the part.

Table 12 — SUBNODE Property

JT Property Key	Meaning	JT File Data Type	Encoded Data Type	Valid Values
SUBNODE	Specifies the node as being a subnode of the parent assembly	String	MbString	0 or 1

6.2.1.6 Meta Data Node Element

Object Type ID: 0xce357245, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1

The Meta Data Node Element, as shown in Figure 32, is a node type used for storing references to specific “late loaded” meta-data (for example properties, PMI). The referenced meta-data is stored in a separate addressable segment of the ISO 14306 File (see 9 Meta Data Segment) and thus the use of this Meta Data Node Element is in support of the ISO 14306 file loader/reader “best practice” of late loading data (therefore storing the referenced meta-data in separate addressable segment of the ISO 14306 file allows an ISO 14306 file loader/reader to ignore this node’s meta-data on initial load and instead late-load the node’s meta-data upon demand so that the associated meta-data does not consume memory until needed).

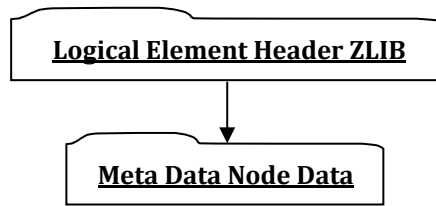


Figure 32 — Meta Data Node Element data collection

A complete description for **Logical Element Header ZLIB** can be found in 5.3.

6.2.1.6.1 Meta Data Node Data

A diagram of Meta Data Node Data is shown in Figure 33.

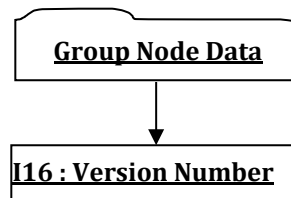


Figure 33 — Meta Data Node Data collection

A complete description for Group Node Data can be found in 6.2.1.3.1.

I16 : Version Number

Version Number is the version identifier for this data. For information on local version numbers, see best practice in 11.5.

6.2.1.7 LOD Node Element

Object Type ID: 0x10dd102c, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

An LOD Node, as shown in Figure 34, holds a list of alternate representations. The list is represented as the children of a base group node, however, there are no implicit semantics associated with the ordering. Traversers of LSG may apply semantics to the ordering as part of alternative representation selection.

Each alternative representation could be a sub-assembly where the alternative representation is a group node with an assembly of children.

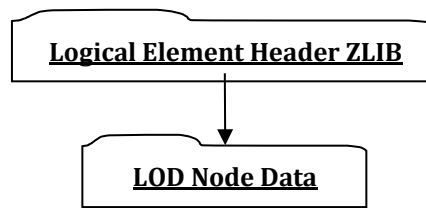


Figure 34 — LOD Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.

6.2.1.7.1 LOD Node Data

A diagram of Lod Node Data is shown in Figure 35.

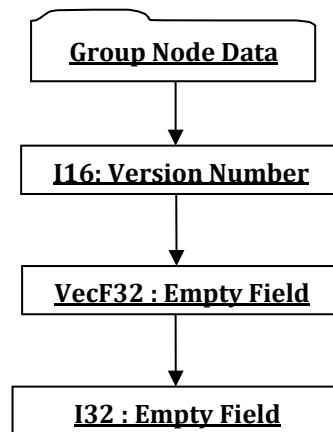


Figure 35 — LOD Node Data collection

A complete description for Group Node Data can be found in 6.2.1.3.1.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

VecF32 : Empty Field

A vector data field. Refer to best practice in 11.4.

I32 : Empty Field

Refer to best practice in 11.4.

6.2.1.8 Range LOD Node Element

Object Type ID: 0x10dd104c, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Range LOD Nodes, as shown in Figure 36, hold a list of alternate representations and the ranges over which those representations are appropriate. Range Limits indicate the distance between a specified centre point and the eye point, within which the corresponding alternate representation

is appropriate. Traversers of LSG consult these range limit values when making an alternative representation selection.

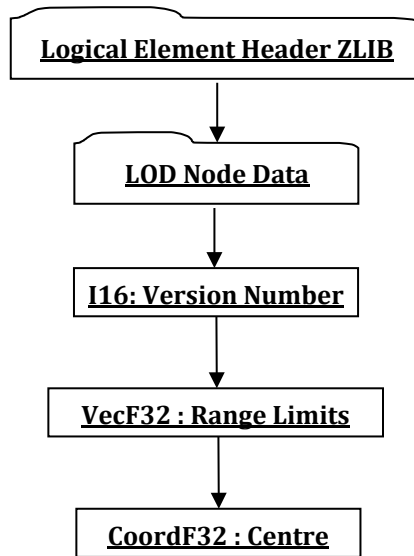


Figure 36 — Range LOD Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.

A complete description for LOD Node Data can be found in 6.2.1.7.1.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

VecF32 : Range Limits

Range Limits indicate the WCS distance between a specified centre point and the eye point, within which the corresponding alternate representation is appropriate. It is not required that the count of range limits is equivalent to the number of alternative representations. These values are considered “soft values” in that loaders/viewers of ISO 14306 data are free to throw these values away and compute new values based on their desired LOD selection semantics.

Common Data Conventions and Constructs suggest that LSG traversers apply the following strategy, at Range LOD Nodes, when making alternative representation selection decisions based on Range Limits: The first alternate representation is valid when the distance between the centre and the eye point is less than or equal to the first range limit (and when no range limits are specified). The second alternate representation is valid when the distance is greater than the first limit and less than or equal to the second limit, and so on. The last alternate representation is valid for all distances greater than the last specified limit.

CoordF32 : Centre

Centre specifies the X, Y, Z coordinates for the MCS centre point upon which alternative representation selection eye distance computations are based. Typically this location is the centre of the highest-detail alternative representation. These values are considered “soft values” in that

loaders/viewers of ISO 14306 data are free to throw these values away and compute new values based on their desired LOD selection semantics.

6.2.1.9 Switch Node Element

Object Type ID: 0x10dd10f3, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

The Switch Node, as shown in Figure 37, is very much like a Group Node in that it contains an ordered list of references to other nodes, called the children nodes. The difference is that a Switch Node also contains additional data indicating which child (one or none) a LSG traverser should process/traverse.

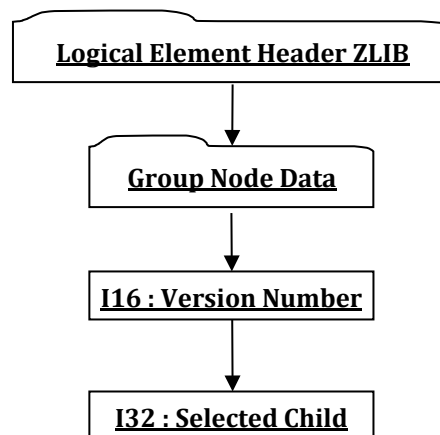


Figure 37 — Switch Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.

A complete description for Group Node Data can be found in 6.2.1.3.1.

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

I32 : Selected Child

Selected Child is the index for the selected child node. Valid Selected Child values reside within the following range: “-1 < Selected Child < Child Count”. Where “-1” indicates that no child is to be selected and “Child Count” is the data field value from 6.2.1.3.1.

6.2.1.10 Shape Node Elements

Shape Node Elements are “leaf” nodes within the LSG structure and contain or reference the geometric shape definition data (for example vertices, polygons, normals, etc.).

Typically Shape Node Elements do not directly contain the actual geometric shape definition data, but instead reference (using Late Loaded Property Atoms) Shape LOD Segments within the file for the actual geometric shape definition data. Storing the geometric shape definition data within separate independently addressable data segments in the ISO 14306 file, allows an ISO 14306 file reader to be structured to support the “best practice” of delaying the loading/reading of associated data until it is actually needed.

There are several types of Shape Node Elements which the ISO 14306 format supports. The following subclauses document the various Shape Node Element types.

6.2.1.10.1 Base Shape Node Element

Object Type ID: 0x10dd1059, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Base Shape Node Element. As shown in Figure 38, represents the simplest form of a shape node that can exist within the LSG.

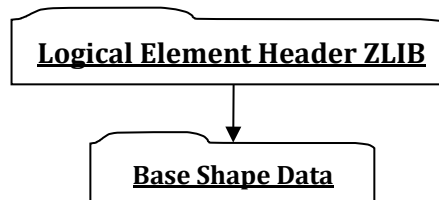


Figure 38 — Base Shape Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.

6.2.1.11 Base Shape Data

A diagram of Base Shape Data is shown in Figure 39.

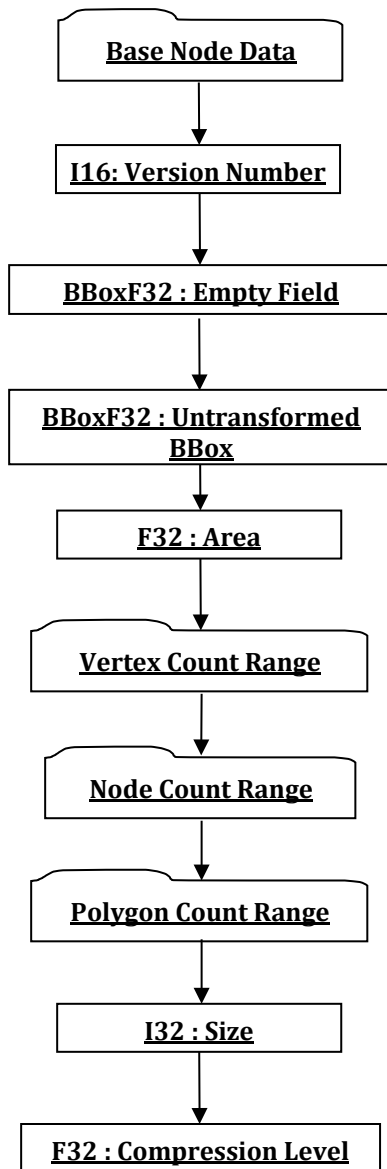


Figure 39 — Base Shape Data collection

A complete description for Base Node Data can be found in 6.2.1.1.1.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers, see best practice in 11.5.

BBoxF32 : Empty Field

Refer to best practice in 11.4.

BBoxF32 : Untransformed BBox

The Untransformed BBox is an axis-aligned LCS bounding box and represents the untransformed geometry extents for all geometry contained in the Shape Node.

F32 : Area

Area is the total surface area for this node and all of its descendants. This value is stored in MCS coordinate space (therefore values scaled by MCS scaling).

I32 : Size

Size specifies the in memory length in bytes of the associated/referenced Shape LOD Element. This Size value has no relevancy to the on-disk (ISO 14306 File) size of the associated/referenced Shape LOD Element. A value of zero indicates that the in memory size is unknown. See 7.1 Shape LOD element for a complete description of Shape LOD Elements. ISO 14306 file loaders/readers can leverage this Size value during late load processing to help pre-determine if there is sufficient memory to load the Shape LOD Element.

F32 : Compression Level

Compression Level, as shown in Table 13, specifies the qualitative compression level applied to the associated/referenced Shape LOD Element. See 7.1 Shape LOD Element for a complete description of Shape LOD Elements. This compression level value is a qualitative representation of the compression applied to the Shape LOD Element. The absolute compression (derived from this qualitative level) applied to the Shape LOD Element is physically represented in the ISO 14306 format by other data stored with both the Shape Node and the Shape LOD Element (for example, Quantization Parameters), and thus it is not necessary to understand how to map this qualitative value to absolute compression values in order to uncompress/decode the data.

Table 13 — Compression level values

= 0.0	“Lossless” compression used.
= 0.1	“Minimally Lossy” compression used. This setting generally results in modest compression ratios with little if any visual difference when compared to the same images rendered from “Lossless” compressed Shape LOD Element.
= 0.5	“Moderate Lossy” compression used. The setting results in more data loss than “Minimally Lossy” and thus higher compression ratio is obtained. Some visual difference will likely be noticeable when compared to the same images rendered from “Lossless” compressed Shape LOD Element.
= 1.0	“Aggressive Lossy” compression used. With this setting as much data as possible will be thrown away, resulting in highest compression ratio, while still maintaining a modestly useable representation of the underlying data. Visual differences may be evident when compared to the same images rendered from “Lossless” compressed Shape LOD Element.

Vertex Count Range

Vertex Count Range, as shown in Figure 40, is the aggregate minimum and maximum vertex count for this Shape Node. There is a minimum and maximum value to accommodate shape types that can themselves generate varying representations. The minimum value represents the least vertex count that can be achieved by the Shape Node. The maximum value represents the greatest vertex count that can be achieved by the Shape Node.

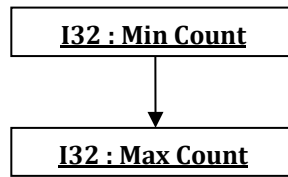


Figure 40 —Vertex Count Range data collection

I32 : Min Count

Min Count is the least vertex count that can be achieved by this Shape Node.

I32 : Max Count

Max Count is the maximum vertex count that can be achieved by this Shape Node. A value of “-1” indicates maximum vertex count is unknown.

Node Count Range

Node Count Range is the aggregate minimum and maximum count of all node descendants of the Shape Node. The minimum value represents the least node count that can be achieved by the Shape Node’s descendants. The maximum value represents the greatest node count that can be achieved by Shape Node’s descendants. For Shape Nodes the minimum and maximum count values should always be equal to “1”.

The data format for Node Count Range is the same as that described in Vertex Count Range.

Polygon Count Range

Polygon Count Range is the aggregate minimum and maximum polygon count for this Shape Node. There is a minimum and maximum value to accommodate shape types that can themselves generate varying representations. The minimum value represents the least polygon count that can be achieved by the Shape Node. The maximum value represents the greatest polygon count that can be achieved by the Shape Node.

The data format for Polygon Count Range is the same as that described in Vertex Count Range.

6.2.1.11.1 Vertex Shape Node Element

Object Type ID: 0x10dd107f, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Vertex Shape Node Element, as shown in Figure 41, represents shapes defined by collections of vertices.

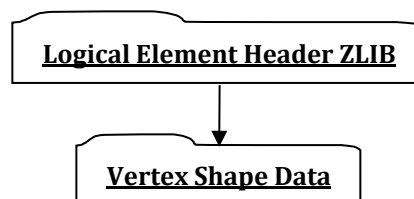


Figure 41 — Vertex Shape Node Element data collection

A complete description for Logical Element Header ZLIB can be found in 5.3.