JT File Format Reference
Version 9.5
Rev-A

Acknowledgments

Documents of this type typically require many hands both to author, and to ensure their correctness. However, if one single person can be identified most responsible for bringing this specification document into existence, it is Gary Lance. He authored the JT v8.1 Specification almost single-handedly over the course of four months in 2006, beginning with no knowledge of the DirectModel toolkit from which the JT format springs. This document owes much to the considerable efforts and high standards of quality Gary brought to that first version.

Equally required of documents of this type, come the inevitable erratum or two. Paul Kitchen, of Wilcox Associates, Inc. a Hexagon Metrology Company, is due special thanks for his patient, diligent, and even enthusiastic work with the authors in finding and correcting several bugs in the original JT v8.1 reference document. To our knowledge, Paul is the first outside developer to correctly read all data entities documented in the JT v8.1 specification.

This updated JT Version 9.5 document was written by the JT Format and DirectModel team's developers themselves: Michael Carter, Jianbing Huang, Sashank Ganti, Jeremy Bennett, and Bo Xu.

Table of Contents

List of Tables

List of Figures

1 Siemens JT Data Format Reference Intellectual Property License Terms

The general idea of using an interchange format for electronic documents is in the public domain. Anyone is free to devise a set of unique data structures and operators that define an interchange format for electronic documents. However, Siemens Product Lifecycle Management Software Inc. owns the copyright for the particular data structures and operators, the JT™ Data Format Reference and the written specification constituting the interchange format called the JT Data Format. Thus, these elements of the JT Data Format may not be copied without Siemens¶V SHUPLVVLRQ

Siemens will enforce its copyright. Siemens¶V LQWHQWLRQ LV WR PDLQWDLQ WKH LQWHJULW\ RI WKH -7 'DWD)RUPDW VWDQGDUG, enabling the public to distinguish between the JT Data Format and other interchange formats for electronic documents. However, Siemens desires to promote the use of the JT Data Format for information interchange among diverse products and applications. Accordingly, Siemens gives anyone copyright permission, subject to the conditions stated below, to:

- Prepare and distribute files whose content conforms solely to the JT Data Format.
- Write and distribute software applications that produce discreet output represented in the JT Data Format. Write and distribute software applications that accept input in the form of the JT Data Format and display, print, or otherwise interpret the contents
- Copy Siemens¶V FRS\ULJKWHG OLVW RI GDWD VWUXFWXUHV DQG RSHUDWRUV LQ WKH ZULWWHQ VSHFLILFDWLRQ WR WKH H[WHQW necessary to use the JT Data Format for the purposes above.
- For avoidance of doubt, the permissions granted in the preceding sentences do not include the reading, writing or distribution of files whose content contains output in the JT Data Format and any other data in any other format and do not include the right to incorporate, integrate, or combine the JT Data Format, structure, or schema into any other data format, structure, or schema.

The conditions of such copyright permission are:

- Anyone who uses the copyrighted list of data structures and operators, as stated above, must include an appropriate copyright notice.

This limited right to use the copyrighted list of data structures and operators does not include the right to copy this document, other copyrighted material from Siemens, or the software in any of Siemens¶V SURGXFWV WKDW XVH WKH JT Data Format, in whole or in part, nor does it include the right to use any Siemens patents, except as may be permitted by an official Siemens JT Data Format Reference Patent Clarification Notice.

Nothing in this book is intended to grant you any right or license to use the Marks for any purpose.

2 Scope

This reference defines the syntax and semantics of the JT Version 9.5 file format.

The JT format is an industry focused, high-performance, lightweight, flexible file format for capturing and repurposing 3D Product Definition data that enables collaboration, validation and visualization throughout the extended enterprise. JT format is the de-facto standard 3D Visualization format in the automotive industry, and the single most dominant 3D visualization format in Aerospace, Heavy Equipment and other mechanical CAD domains.

The JT format is both robust, and streamable, and contains best-in-class compression for compact and efficient representation. The JT format was designed to be easily integrated into enterprise translation solutions, producing a single set of 3D digital assets that support a full range of downstream processes from lightweight web-based viewing to full product digital mockups.

At its core the JT format is a scene graph with CAD specific node and attributes support. Facet information (triangles), is stored with sophisticated geometry compression techniques. Visual attributes such as lights, textures, materials and shaders (Cg and OGLSL) are supported. Product and Manufacturing Information (PMI), Precise Part definitions (B-Rep) and Metadata as well as a variety of representation configurations are supported by the format. The JT format is also structured to enable support for various delivery methods including asynchronous streaming of content.

Some of the highlights of the JT format include:

- Built-in support for assemblies, sub-assemblies and part constructs
- Flexible partitioning scheme, supporting single or multiple files
- B-Rep, including integrated support for industry standard Parasolid® (XT) format
- Product Manufacturing Information in support of paperless manufacturing initiatives
- Precise and imprecise wireframe
- Discrete purpose-built Levels of Detail
- Wire harness information
- Triangle sets, Polygon sets, Point sets, Line sets and Implicit Primitive sets (c\0LQGHU FRQH VSKHUH HWF «
- Full array of visual attributes: Materials, Textures, Lights, Shaders
- Hierarchical Bounding Box and Bounding Spheres
- Advanced data compression that allows producers of JT files to fine tune the tradeoff between compression ratio and fidelity of the data.

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

Offline optimizations of the data contents

- File granularity and flexibility optimized to meet the needs of Enterprise Data Translation Solutions

Asynchronous streaming of content

- Viewing optimizations such as view usiu

This specification is based on the Version 8.1 Rev D specification, but with major changes to all sections, and as such is a completely new, standalone document.

3 References and Additional Information

[1] *JT Open Program* (http://www.jtopen.com) --- A program to help members leverage the benefits of open collaboration across the extended enterprise through the adoption of the JT format, a technology that makes it possible to view and share product information throughout the product lifecycle. Membership in the JT Open Program provides access to the JT Open Toolkit library, which among other things, provides read and write access to JT data and enforces certain JT conventions to ensure data compatibility with other JT-enabled applications.

[2] *JT2Go download* (http://www.jt2go.com) --- JT2Go is the no-charge 3D JT viewer from Siemens. JT2Go puts 3D data at your fingertips by allowing anyone to download the no-charge viewer. JT2Go also allows anyone to embed 3D JT data directly into Microsoft Office documents. JT2Go offers full 3D interactivity on parts, assemblies, and even 2D drawings (CGM & TIF).

[3] *Siemens: PLM Components: Parasolid: XT Pipeline* (http://www.ugs.com/products/open/parasolid/pipeline.shtml) --- This web page provides information on the Parasolid precise boundary representation format (XT) and how this XT format fits within the Siemens vision of seamless exchange of digital product models across enterprises, between different disciplines, using their PLM applications of choice.

[4] *OpenGL Programming Guide : The official guide to learning OpenGL Version 2*, Fifth Edition, by OpenGL Architecture Review Board, Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis (Addison-Wesley 2005) --- This book gives in-depth explanation of the OpenGL Specification and will provide further insight into the significance of some of the data (e.g. Materials, Textures) that can exist in a JT file. Information in this book may also serve as a guide for how one could process the data contained in a JT file to produce/render an image on the screen.

[5] Michael Deering, *Geometry Compression*, Computer Graphics, Proceedings SIGGRAPH µ $XJXW SS 13-20.

[6] Michael Deering, Craig Gotsman, Stefan Gumhold, Jarek Rossignac, and Gabriel Taubin, *3D Geometry Compression*, Course Notes for SIGGRAPH 2000, July 25, 2000.

[7] *OpenGL Shading Language Specification* (http://www.opengl.org/documentation/glsl/) --- OpenGL Shading Language (GLSL) as defined by the OpenGL Architectural Review Board, the governing body of

[14] Michael Schindler, *Practical Huffman Coding* (http://www.compressconsult.com/huffman/#encoding) --- This web page provides some coding hints for implementing Huffman coding which is one form of data encoding used within the JT format.

[15] Glen G. Langdon Jr., *An Introduction to Arithmetic Coding*, IBM Journal of Research and Development, Volume 28, Number 2, March 1984, pp. 135-149.

[16] Paul G. Howard and Jeffrey Scott Vitter, *Practical Implementation of Arithmetic Coding. Image and Text Compression*, ed. J. A. Storer, Kluwer Academic Publishers, April 1992, pp. 85-112.

[17] zlib.net (http://www.zlib.net/) --- This web page provides (either directly or through links) complete detailed information on ZLIB compression including frequently asked questions, technical documentation, source code downloads, etc.

[18] Andrei Khodakovsky, Pierre Alliez, Mathieu Desbrun, and Peter Schröder, *Near-Optimal Connectivity Encoding of 2-Manifold Polygon Meshes*, Graphical Models, Vol. 64, No. 3-4, Pages: 147 - 168, 2002.

[19] B. Schneier, *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*, Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.

4 Definitions

4.1 Terms

It is assumed that readers of this document are familiar with concepts in the area of computer graphics and solid modeling. The intention of this section is not to provide comprehensive definitions, but is to provide a short introduction and clarification of the usage of terms within this document.

Assembly	A related collection of *model* parts, represented in a JT format logical scene graph as a logical graph branch
Attribute	Objects associated with nodes in a *logical scene graph* and specifying one of several appearances, positioning, or rendering characteristics of a *shape*.
Boundary Representation	A solid model representation where the solid volume is specified by its surface boundary (both its geometric and topological boundaries).
CodeText	A collection of data in encoded form.
Directed Acyclic Graph	A *graph* is a set of nodes, and a set of edges connecting the nodes in a tree like structure. A *directed graph* is one in which every edge has a direction such that edge (u,v), connecting node-u with node-v, is different from edge (v,u). A *Directed Acyclic Graph* is a directed graph with no cycles; where a cycle is a path (sequence of edges) from a node to itself. So with a *Directed Acyclic Graph* there is no path that can be followed within the graph such that the first node in the path is the same as the last node in the path.
JT Enabled Application	Application which supports reading and/or writing reference compliant JT Format files.
Level of Detail	One alternative graphical representation for some *model* component (e.g. part).
Logical Scene Graph	A *scene graph* representing the logical organization of a *model*. Contains *shapes* and *attributes* representing the PRGH0¶V physical

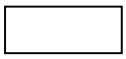
	components, *properties* identifying arbitrary metadata (e.g. names, semantic roles) of those components, and a hierarchical structure expressing the component relationships.
Mipmap	A reduced resolution version of a texture map. Mipmaps are used to texture a geometric primitive whose screen resolution differs from the resolution of the source texture map originally applied to the primitive.
Model	Representation, in JT format, of a physical or virtual product, part, assembly; or collections of such objects.
Parasolid XT Format	Parasolid boundary representation format
Product and Manufacturing Information	Collection of information created on a 3D/2D CAD Model to completely document the product with respect to design, manufacturing, inspection, etc. This may includes data such as: Dimensions (tolerances for each dimension) Geometric tolerances of feature (datums, feature control frames) Manufacturing information (surface finish, welding notations) Inspection information (key locations points) Assembly instructions Product information (materials, suppliers, part numbers)
Property	An object associated with a logical scene graph node and identifying arbitrary application or enterprise specific information (meta-data) related to that node.
Quantize	Constrain something to a discrete set of values, such as an integer or integral multiplier of a common factor, rather than a continuous set of values, such as a real number.
Scene Graph	In the context of the JT format, a scene graph is a *directed acyclic graph* that arranges the logical and often (but not necessarily) spatial representation of a graphical scene.
Shader	A user-definable program, expressed directly in a target assembly language, or in high-level form to be compiled. A shader program replaces a portion of the otherwise fixed-functionality graphics pipeline with some user-defined function. At present, hardware manufacturers have made it possible to run a shader for each vertex that is processed or each pixel that is rendered.
Streaming	In the context of the JT format, streaming refers to both: Loading from disk based medium only the portions of data that are required by the user to perform the tasks at hand. The motivation being to more efficiently manage system memory. Transfer of data in a stream of packets, over the internet on an on-demand basis, where the data is interpreted in real-time by the application as the data packets arrive. The motivation being that the user can begin using or interacting with the data almost immediately - no waiting for the entire data file(s) to be transferred before beginning The desired end result of both being to *deliver only the JT data that the user needs, where the user needs it, when the user needs it.* A

LCS	Local Coordinate System
LOD	Level of Detail
LsbFirst	Least Significant Byte First
LSG	Logical Scene Graph
Max	Maximum
MCS	Model Coordinate System
Min	Minimum
MsbFirst	Most Significant Byte First
N/A	Not Applicable
NCS	Node Coordinate System
PCS	Parameter Coordinate Space
PLM	Product Lifecycle Management
PMI	Product and Manufacturing Information
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha
TOC	Table of Contents.
VPCS	Viewpoint Coordinate System
URL	Uniform Resource Locator
WCS	World Coordinate System

6 Notational Conventions

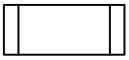
6.1 Diagrams and Field Descriptions

Symbolic diagrams are used to describe the structure of the JT file. The symbols used in these diagrams have the following meaning:

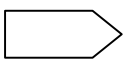


Rectangles represent a data field of one of the standard data types.

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 stepsthat 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 ³rectangle box´ (i.e. standard data types) symbols are titled using a format of ³Data_Type : Field_Name´ 7KH Data_Type is an abbreviated data type symbol as defined in 6.2 Data Types ,Q WKH H[DPSOH EHORZ WKH 'DWDB7\SH LV ³, ´ (a signed 32 bit integer) and Field_Name iV ³&RXQW´

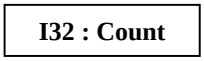
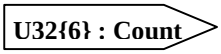


Diagram ³folder´ (i.e. logical data collections) symbols are simply titled with a collection name. In the example below the FROOHFWLRQ QDPH LV ³*UDSK (OHPHQWV´

'LDJUDP ³UHFWDQJOH ER[ZLWK lines at left and right sides´ DUH VLPSO\ WLWOHG ZLWK D ORJLF VWHS QDPH ,Q WKH H[DPSOH EHORZ WKH ORJLF VWHS QDPH LV ³5HFRYHU)LUVW 6KHOO ,QGLFHV´

Diagram ³UHFWDQJOH ER[ZLWK FOLSSHG ULJKW VLGH FRUQHUV´ L H FRPSUHVVHG/encoded data fields) are titled using one of the following three formats:

Data Type; followed by open brace ³^³, number of bits used to store value, closed brace ³`´ DQG D FRORQ ³ ´ IROORZHG E\ WKH 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 (e.g. if a count value FDQ QHYHU EH ODUJHU WKDQ WKH YDOXH ³ ´ WKHQ RQO\ ELWV DUH QHHGHG WR VWRUH D SRVVLEOH FRXQW YDOXHV . In the example below WKH 'WD 7\SH LV ³8 ´, ³6´ bits are used to store the value, and Field Name is ³&RXQW´



'WD 7\SH IROORZHG E\ RSHQ EUDFH ³^³ compressed data packet type, ³ ´ 3UHGLFWRU 7\SH FORVHG EUDFH ³`´ DQG D FRORQ ³ ´ followed by the field name. This format for titling the diagram indicates that a vector of ³'WD 7\SH´ data (i.e. *primal* values) LV UDQ WKURXJK ³Predictor 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 (i.e. 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 8.1.1 Int32 Compressed Data Packet.

Int32CDP2 ± The Int32CDP2 (i.e.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 8.1.2 Int32 Compressed Data Packet Mk. 2.

Float64CDP ± The Float64CDP (i.e. 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 8.1.3 Float64 Compressed Data Packet.

The Int32 Compressed Data Packet W\SH LV XVHG IRU FRPSUHVVLQJ HQFRGLQJ ERWK ³LQWHJHU´ DQG ³IORDW´ WKURXJK TXDQWL]DWLRQ data. While the Float64 Compressed Data Packet W\SH LV XVHG IRU FRPSUHVVLQJ HQFRGLQJ ³GRXEOH´ GDWD

,Q WKH H[DPSOH EHORZ WKH 'WD 7\SH LV ³9HF8 ´ Int32 Compressed Data Packet type is used, Lag1 Predictor Type is used, DQG)LHOG 1DPH LV ³First Shell IQGH[´

VecU32{Int32CDP, Lag1} : First Shell

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 (i.e. difference from the predicted value), and this resulting output array of *residual* values is the data which is actually encoded/compressed. The JT 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 JT 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 Appendix C: Decoding Algorithms ± An Implementation):

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:
	`if(val2back - val4back < 8 && val2back - val4back > -8)`
	`  iPredicted = val2back + (val2back - val4back);`
	`else`
	`  iPredicted = val2back + 2;`

Ramp	3UHGLFW YDOXH ³L´ DV values ³L¶V´ 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 can be combined with additional processing steps, and in such case the predictor type is prefixed with ³&RPELQHG´)RU H[DPSOH ³&RPELQHG /DJ ´ PHDQV WKDW SUHGLFWRU W\SH ³/DJ ´ LV FRPELQHG ZLWK DGGLWLRQDO SUHSURcessing steps. Additional description about the processing steps is provided whenever such combined predictor is used.

³'DWD 7\SH)LHOG 1DPH´ 7KLV IRUPDW IRU WLWOLQJ WKH GLJUDP V\PERO LQGLFDWHV WKDW WKH GDWD LV ERWK FRPSUHVVHG DQG encoded. The Data_Type is an abbreviated data type symbol as defined in 6.2 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 below the 'DWDB7\SH LV ³9HF8 ´ DQG)LHOGB1DPH LV ³&RGH7H[W´

VecU32 : CodeText

Note that for some JT file Segment Types there is ZLIB compression also applied to all bytes of element data stored in the VHJPHQW 7KLV =/,% FRPSUHVVLRQ DSSOLHG WR DOO WKH VHJPHQW¶V GDWD LV QRW LQGLFDWHG LQ WKH GLJUDPV WKURXJK WKH XVH RI

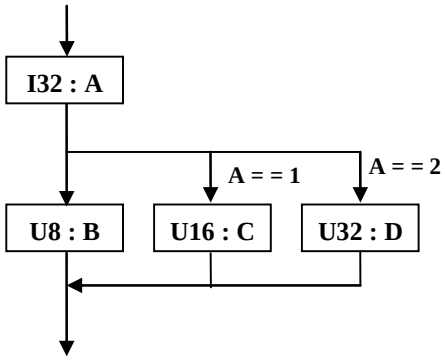
³UHFWDQJOH ER[ZLWK FOLSSHG ULJKW VLGH FRUQHUV´ ,QVWead, one must examine information stored with the first Element in the file segment to determine if ZLIB compression is applied to all data in the segment. A complete description of the JT format data compression and encoding can be found in 7.1.3 Data Segment and 8 Data Compression and Encoding.

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

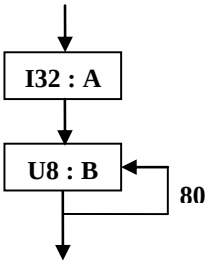
For rectangles this detail includes the abbreviated data type symbol, field name, verbal data description, and compression technique/algorithm where appropriate. If the data field is documented as a collection of flags, then the field is to be treated as a bit mask where the bit mask is formed by combining 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. Any undocumented bits are reserved.

For folders (i.e. 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.

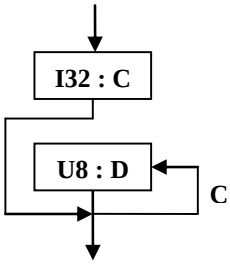
If an arrow appears with a branch in its shaft, then there are two or more options for data to be stored in the file. Which data is stored will depend on information previously read from the file. The following example shows data field A followed by (depending on value of A) either data field B, C, or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



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 (e.g. 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 looks as follows:



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



6.2 Data Types

The data types that can occur in the JT binary files are listed in the following two tables.

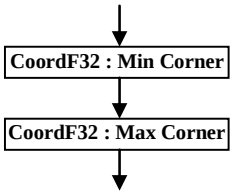
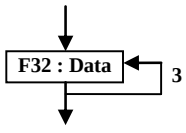
Table 1: Basic Data Types lists the basic/standard data types which can occur in JT file.

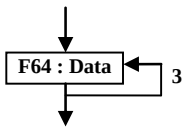
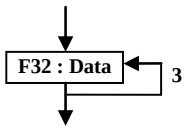
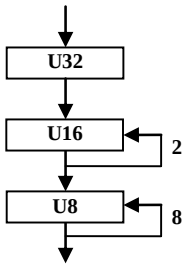
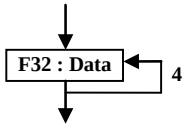
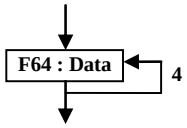
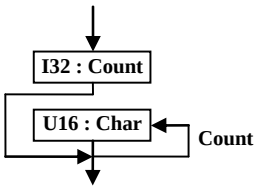
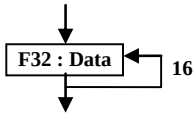
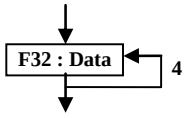
Table 1: Basic Data Types

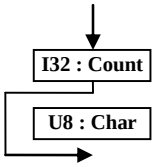
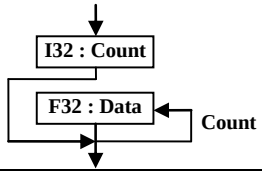
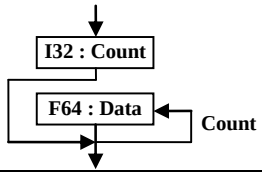
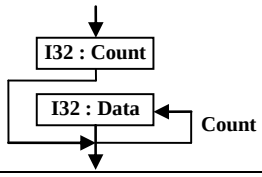
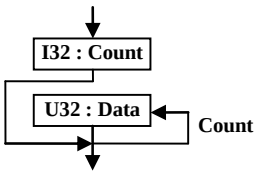
Type	Description
UChar	An unsigned 8-bit byte.
U8	An unsigned 8-bit integer value.
U16	An unsigned 16-bit integer value.
U32	An unsigned 32-bit integer value.
U64	An unsigned 64-bit integer value.
I16	$ VLJQHG ZRUG FRPSOHPHQW -bit integer value.
I32	$ VLJQHG ZRUG FRPSOHPHQW -bit integer value.
I64	A 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 2: Composite Data Types lists some composite data types which are used to represent some frequently occurring groupings of the basic data types (e.g. Vector, RGBA color). The composite data types are defined in this reference simply for convenience/brevity in describing the JT file contents.

Table 2: 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.	

Type	Description	Symbolic Diagram
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 JT 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 JT format GUIDs are used as unique identifiers (e.g. 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 E\WHV RI FKDUDFWHU GDWD LV ³ 1XP&KDU´ L H WKH VWULQJV DUH 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 + b***i*** + c***j*** + d***k***) 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	

Type	Description	Symbolic Diagram
	four F32 base types.	
RGB	The RGB type defines a color 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 color values typically range from 0.0 to 1.0.	
RGBA	The RGBA type defines a color 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 color 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 E\WHV RI FKDUDFWHU GDWD LV ³1XP&KDU´ L H WKH VWULQJV DUH 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 QXPEHU WR EH HTXDO WR ³ ´ LQGLFDWLQJ QR IRORZLQJ)	
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 QXPEHU WR EH HTXDO WR ³ ´ LQGLFDWLQJ QR IRORZLQJ)	
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 QXPEHU WR EH HTXDO WR ³ ´ LQGLFDWLQJ QR IRORZLQJ ,	
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 , FRXQW QXPEHU WR EH HTXDO WR ³ ´ LQGLFDWLQJ QR IRORZLQJ U32.	

7 File Format

All objects represented in the JT format are assigned an ³object identifier´ (e.g. see 7.2.1.1.1.1.1 Base Node Data, or 7.2.1.1.2.1.1 Base Attribute Data) and all references from one object to another object are represented in the JT format using the referenced REWHFW¶V ³object identifier´. It is the responsibility of JT format readers/writers to maintain the integrity of

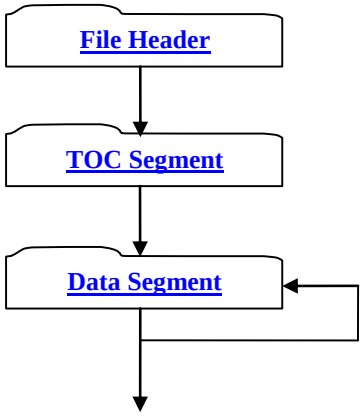
these object references by doing appropriate pointer unswizzling/swizzling as JT format data is read into memory or written RXW WR GLVN : KHUH ³SRLQWHU VZL]]OLQJ´ UHIHUV to the process of converting references based on object identifiers into direct PHPRU\ SRLQWHU UHIHUHQFHV DQG ³SRLQWHU XQVZL]]OLQJ´ LV WKH UHYHUVH RSHUDWLRQ (i.e. replacing references based on memory pointers with object identifier references).

7.1 File Structure

A JT file is structured as a sequence of blocks/segments. 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 one Data Segment which must always exist to have a reference compliant JT file is the 7.2.1 LSG 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. Although there are no JT format compliance rules about where the TOC Segment must be located within the file, in practice the TOC Segment is typically located either immediately following the File header (as shown in the below Figure) or at the very end of the file following all other Data Segments.

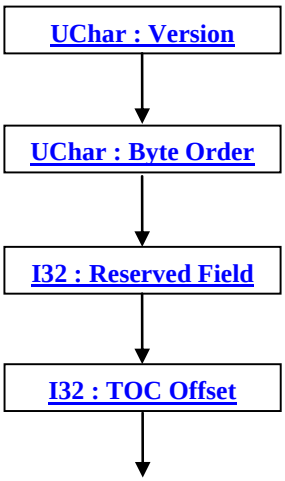
Figure 1: JT File Structure



7.1.1 File Header

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

Figure 2: File Header data collection



UChar : Version

An 80-character version string defining the version of the file format used to write this file. The Version string has the following format:

Version ***M.n Comment***

Where ***M*** is replaced by the major version number, ***n*** is replaced by the minor version number, and ***Comment*** provides other unspecified reserved information. The string with the following format is commonly used as ***Comment*** to indicate the DM library version that was used to write this JT file:

DM ***Maj.Min.Qrm.Irm***

Where ***Maj***, ***Min***, ***Qrm***, and ***Irm*** are replaced by the major, minor, QRM, and IRM numbers respectively.

The version string is padded with spaces to a length of 75 ASCII characters and then the final five characters must 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] = ' '
```

These final 5 characters (shown above and referred to as ASCII/binary translation detection bytes) can be used by JT file readers to validate that the JT files has not been corrupted by ASCII mode FTP transfers. For a JT Version 9.5 file written by DM library version 7.3.4.0 this string will look as follows:

³9HUVLRQ 9.5 JT DM 7.3.4.0 \n\r\Q ³

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 : Reserved Field

Must have the value 0.

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 must be accessible from the root LSG Data Segment.

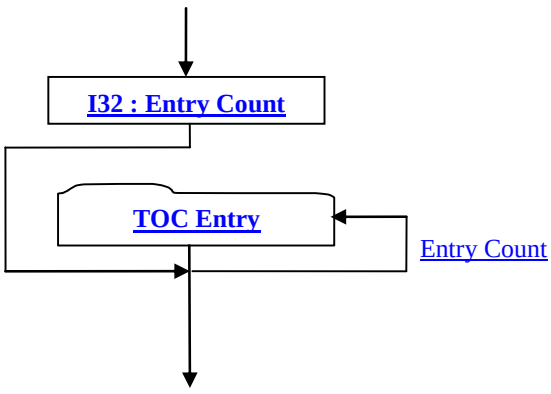
GUID: Reserved Field

Reserved Field is a data field reserved for future JT format expansion

7.1.2 TOC Segment

The TOC Segment contains information identifying and locating all individually addressable Data Segments within the file. A TOC Segment is always required to exist somewhere within a JT file. The actual location of the TOC Segment within the file is specified by the File Header sHJPHQW¶V ³72& 2IIVHW´ ILHOG 7KH 72& 6HJPHQW FRQWDLQV RQH 72& (QWU\ IRU HDFK individually addressable Data Segment in the file.

Figure 3: TOC Segment data collection



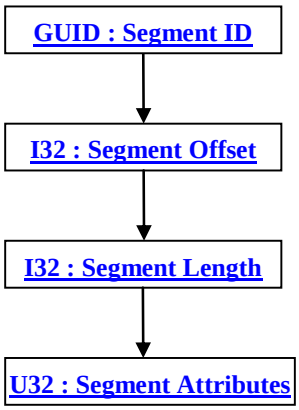
I32 : Entry Count

Entry Count is the number of entries in the TOC.

7.1.2.1 TOC Entry

Each TOC Entry represents a Data Segment within the JT File. The essential function of a TOC Entry is to map a Segment ID to an absolute byte offset within the file.

Figure 4: 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 is a collection of segment information encoded within a single U32 using the following bit allocation.

Bits 0 - 23	Reserved for future use.
Bits 24 - 31	Segment type. Complete list of Segment types can be found in Table 3: Segment Types.

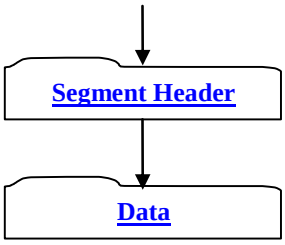
7.1.3 Data Segment

All data stored in a JT file must be defined within a Data Segment. Data Segments are 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 within the segment header. Also Table 3: 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.

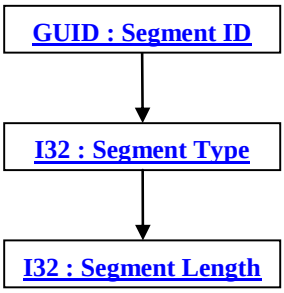
Figure 5: Data Segment data collection



7.1.3.1 Segment Header

Segment Header contains information that determines how the remainder of the Segment is interpreted by the loader.

Figure 6: 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, D 6HJPHQW 7\SH RI ³ ´ GHQRWHV WKDW WKH VHJPHQW FRQWDLQV /RJLFDO 6FHQH *UDSK PDWHULDO ³ ´ GHQRWHV FRQWHQWV RI D B-Rep, etc.

The complete list of segment types is as follows. The column labeled "ZLIB Applied?" denotes whether ZLIB compression is conditionally applied to the entirety of the segment's Data payload.

Table 3: Segment Types

Type	Data Contents	ZLIB Applied?
1	Logical Scene Graph	Yes
2	JT B-Rep	Yes
3	PMI Data	Yes
4	Meta Data	Yes
6	Shape	No
7	Shape LOD0	No
8	Shape LOD1	No
9	Shape LOD2	No
10	Shape LOD3	No
11	Shape LOD4	No
12	Shape LOD5	No
13	Shape LOD6	No
14	Shape LOD7	No
15	Shape LOD8	No

Type	Data Contents	ZLIB Applied?
16	Shape LOD9	No
17	XT B-Rep	Yes
18	Wireframe Representation	Yes
20	ULP	Yes
24	LWPA	Yes

Note: 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 (i.e. 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: The more generic Shape Segment type (i.e. 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 (i.e. is a child of) more than one LOD node,
- Shape has its own built-in LODs, and
- 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 (i.e. it is the size of the FRPSOHWH VHJPHQW DQG VKRXOG EH HTXDO WR WKH OHQJWK YDOXH VWRUHG ZLWK WKLV VHJPHQW¶V TOC Entry.

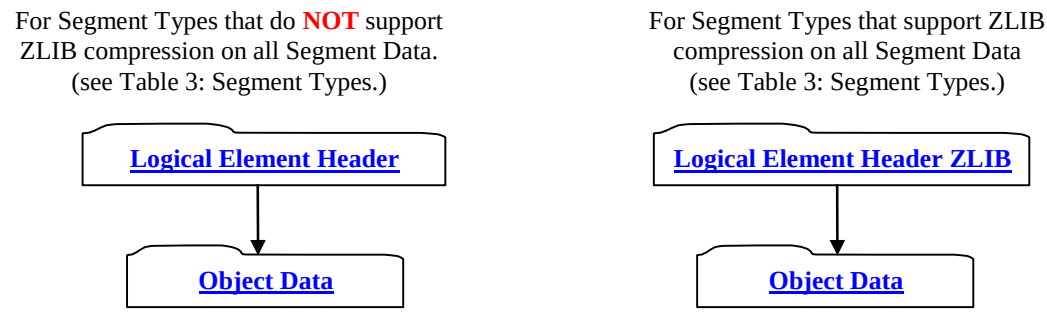
7.1.3.2 Data

The interpretation of the Data section depends on the Segment Type. See 7.2 Data Segments for complete description for all Data Segment that may be contained in a JT 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 (OHPHQW¶V VWRUDJH IRUPDW ,Q DGGLWLRQ DQRWKHU OHYHO of compression (i.e. 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 in Table 3: 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. An in-depth description of JT file compression/encoding techniques can be found in 8 Data Compression and Encoding.

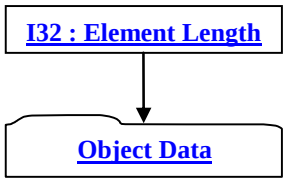
Figure 7: Data collection



7.1.3.2.1 Logical Element Header

Logical Element Header contains data defining the length in bytes of the Element along with the Element Header.

Figure 8: Logical Element Header data collection



Complete description for Logical Element Header can be found in 7.1.3.2.2 Element Header.

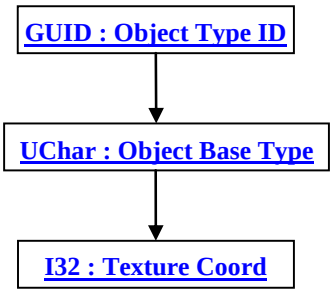
I32 : Element Length

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

7.1.3.2.2 Element Header

Element Header contains data describing the object type contained in the Element.

Figure 9: 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 a JT file can be found in Appendix A: Object Type Identifiers.

UChar : Object Base Type

Object Base Type identifies the base object type. This is useful when an unknown element type is encountered and thus the best the loader can do is to read the known Object Base Type data bytes (base type object data is always written first) and then skip (read pass) the bytes of unknown data using knowledge of number of bytes encompassing the Object Base Type data and the unknown types Length field. If the Object Base Type is unknown then the loader should simply skip (read pass) Element Length number of bytes.

Valid Object Base Types include the following:

Table 4: Object Base Types

Base Type	Description	Base Type's Data Format
255	Unknown Graph Node Object	none
0	Base Graph Node Object	7.2.1.1.1.1.1 Base Node Data
1	Group Graph Node Object	7.2.1.1.1.3.1Group Node Data
2	Shape Graph Node Object	7.2.1.1.1.10.1.1 Base Shape Data
3	Base Attribute Object	7.2.1.1.2.1.1 Base Attribute Data
4	Shape LOD	none

Base Type	Description	Base Type's Data Format
5	Base Property Object	7.2.1.2.1.1 Base Property Atom Data
6	JT Object Reference Object	7.2.1.2.5 JT Object Reference Property Atom Element without the Logical Element Header ZLIB data collection.
8	JT Late Loaded Property Object	0 Late Loaded Property Atom Element without the Logical Element Header ZLIB data collection.
9	JtBase (none)	none

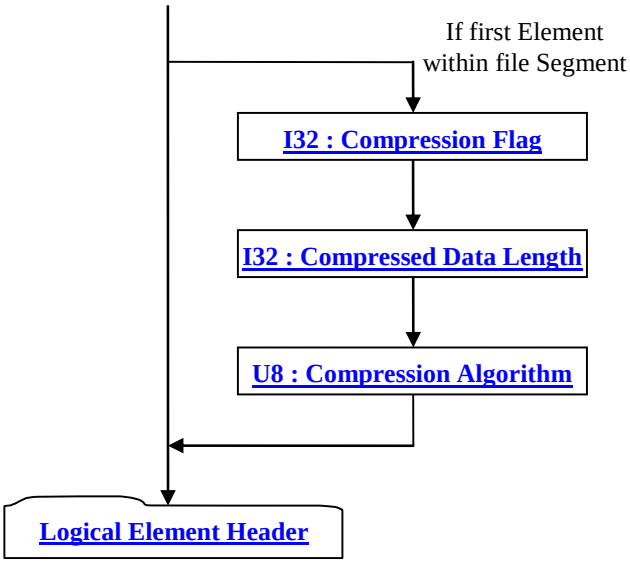
I32 : Object ID

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

7.1.3.2.3 Logical Element Header ZLIB

Logical Element Header ZLIB data collection 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 3: Segment Types for information on whether a particular Segment Type supports ZLIB compression on all data in the Segment.

Figure 10: Logical Element Header ZLIB data collection



Complete description for Logical Element Header can be found in 7.1.3.2.1 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 include the following:

= 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 include the following:

= 1	No compression
= 2	ZLIB compression

7.1.3.2.4 Object Data

The interpretation of the Object Data section depends upon the Object Type ID stored in the Logical Element Header (see 7.1.3.2.1 Logical Element Header).

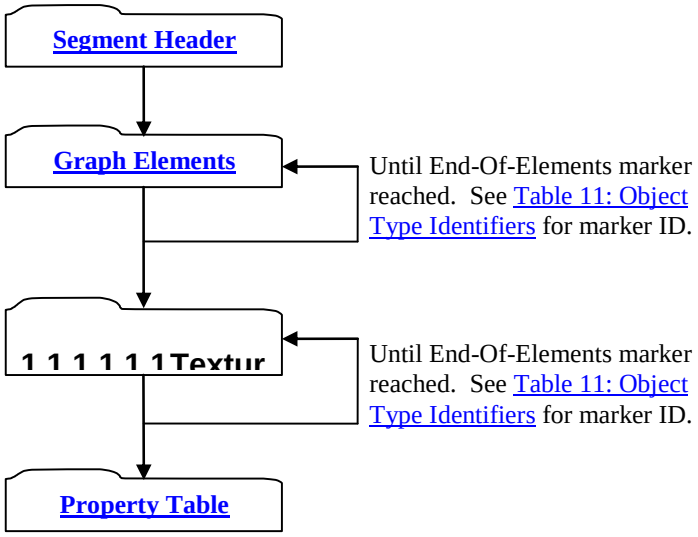
7.2 Data Segments

7.2.1 LSG Segment

LSG Segment contains a collection of objects (i.e. Elements) connected through directed references to form a directed acyclic graph structure (i.e. the LSG). The LSG is the graphical description of the model and contains graphics shapes and attributes UHSUHVHQWLQJ WKH PRGHO¶V SK\VFDO FRPSRQHQW SURSHUWLHV identifying arbitrary metadata (e.g. names, semantic roles) of those components, and a hierarchical strucWXUH H[SUHVVLQJ WKH FRPSRQHQW UHODWLRQVKLSV 7KH ³Directed´ QDWXUH RI WKH /6* references LPSOLHV WKDW WKHUH LV E\ GHIDXOW ³VWDWH DWWULEXWH´ LQKHULWDQFH IURP DQFHVWRU WR GHVFHQGDQW L H SUHGHFHVVRU WR successor). It is the responsibility of the loader to insure that the acyclic property of the resulting LSG is maintained.

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 11: LSG Segment data collection



Complete description for Segment Header can be found in 7.1.3.1Segment Header.

7.2.1.1 Graph Elements

Graph Elements form the backbone of the LSG directed acyclic graph structure DQG LQ GRLQJ VR VHUYH DV WKH -7 PRGHO¶V 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 PRGHO¶V SK\VLFDO FRPSRQents 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 (e.g. Attribute Elements) that is inherited down the LSG to all descendant nodes.

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

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

7.2.1.1.1 Node Elements

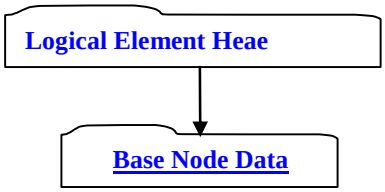
Node Elements represent the UHODWLRQVKLSV RI D PRGHO¶V FRPSRQHQWV 7KH PRGHO¶V Fomponent hierarchy is formed via certain types of Node Elements containing collections of references to other Node Elements who in turn may reference other collections of Node Elements. Node Elements are also the holders (either directly or indirectly) of geometric shape, properties, and other LQIRUPDWLRQ GHILQLQJ D PRGHO¶V FRPSRnents and representations.

7.2.1.1.1.1 Base Node Element

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

Base Node Element represents the simplest form of a node that can exist within the LSG. The Base Node Element has no implied LSG semantic behavior nor can it contain any children nodes.

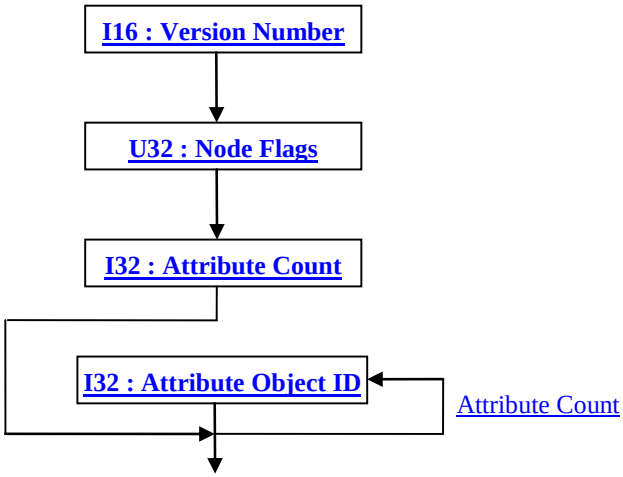
Figure 12: Base Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

7.2.1.1.1.1 Base Node Data

Figure 13: Base Node Data collection



I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ valid value for Base Node Data.

U32 : Node Flags

Node Flags 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 undocumented bits are reserved.

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

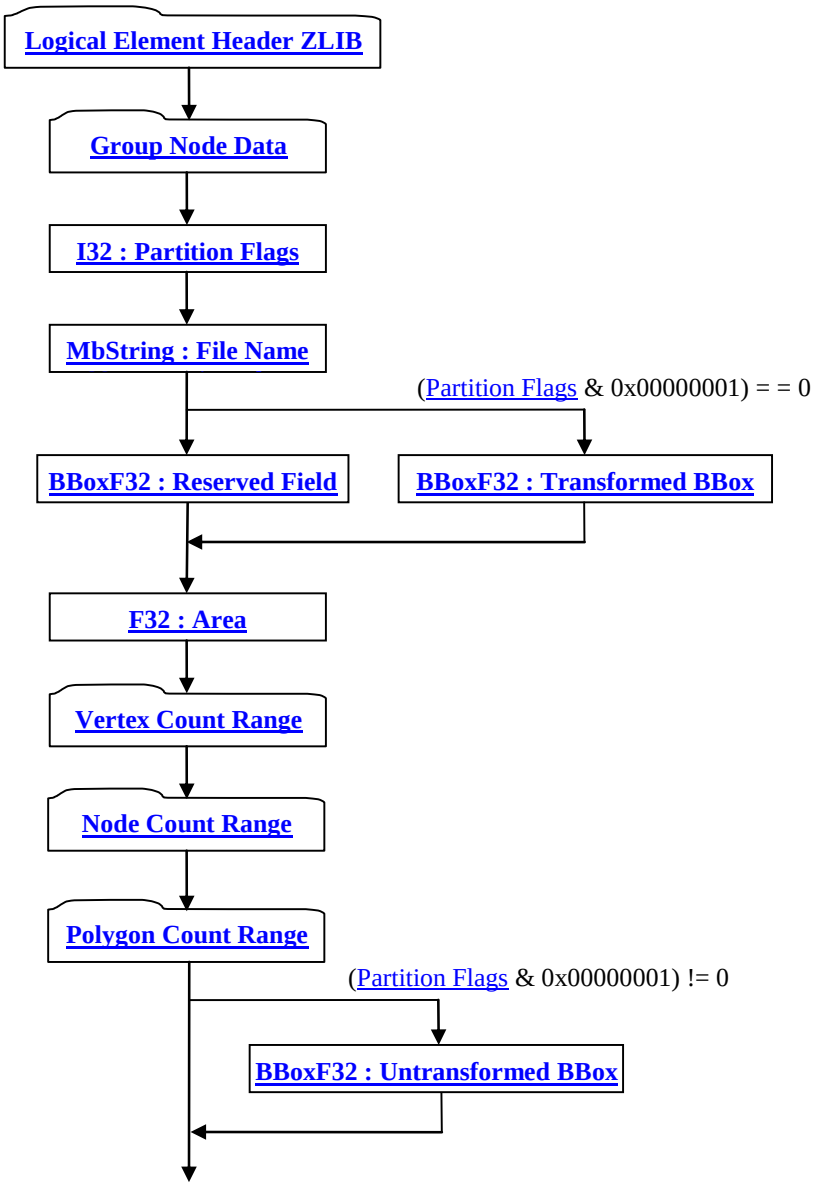
Attribute Object ID is the identifier for a referenced Attribute Object.

7.2.1.1.1.2 Partition Node Element

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

A Partition Node represents an external JT file reference and provides a means to partition a model into multiple physical JT files (e.g. separate JT file per part in an assembly). : KHQ WKH UHIHUHQFHG -7 ILOH LV RSHQHG WKH 3DUWLWLRQ 1RGH¶V FKLOGUHQ DUH really the children of the LSG root node for the underlying JT file. Usage of Partition Nodes in LSG also aids in supporting -7 ILOH ORDGHU UHDGHU ³EHVW practice´ RI ODWH ORDGLQJ GDWD L H FDQ GHOD\ RSHQLQJ DQG ORDGLQJ WKH H[WHUQDO\ UHIHUHQFHG -7 ILOH until the data is needed).

Figure 14: Partition Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Group Node Data can be found in 7.2.1.1.1.3.1Group Node Data.

I32 : Partition Flags

Partition Flags 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 undocumented bits are reserved.

0x00000001	Untransformed bounding box is written.

MbString : File Name

)LOH 1DPH LV WKH UHODWLYH SDWK SRUWLRQ RI WKH 3DUWLWLRQ¶V ILOH 0RFDWLRQ. : KHUH ³UHODWLYH SDWK´ VKRXOG EH LQWHUSUHWHG WR PHDQ WKH string contains the file name along with any additional path information that locates the partition JT file relative to the location of the referencing JT file

BBoxF32 : Reserved Field

Reserved Field is a data field reserved for future JT format expansion

BBoxF32 : Transformed BBox

The Transformed BBox is an NCS 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 JT data to determine whether to load the data contained within the Partition node (i.e. 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 descendents. This value is stored in NCS coordinate space (i.e. values scaled by NCS scaling).

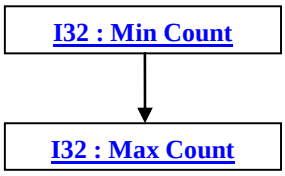
BBoxF32 : Untransformed BBox

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 JT data to determine whether to load the data contained within the Partition node (i.e. is any part of the bounding box within the view frustum).

7.2.1.1.1.2.1 Vertex Count Range

Vertex Count Range is the aggregate minimum and maximum vertex count for all descendants of the Partition Node. There is a minimum and maximum value to accommodate descendant 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 PLQLPXP YDOXH UHSUHVHQWV WKH OHDVW YHUWH[FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV 7KH Maximum YDOXH UHSUHVHQWV WKH JUHDWHVW YHUWH[FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV

Figure 15: Vertex Count Range data collection



I32 : Min Count

0LQ &RXQW LV WKH OHDVW YHUWH[FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV

I32 : Max Count

0D[&RXQW LV WKH PD[LPXP YHUWH[FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV

7.2.1.1.1.2.2 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 WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV 7KH PD[LPXP YDOXH UHSUHVHQWV WKH JUHDWHVW QRGH FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV

The data format for Node Count Range is the same as that described in 7.2.1.1.1.2.1Vertex Count Range.

7.2.1.1.1.2.3 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 descendant branches having LOD nodes, which encompass a range of count values within the branch, and to accommodate nodes that can themselves generate varying representations. 7KH PLQLPXP YDOXH UHSUHVHQWV WKH OHDVW SROJRQ FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV 7KH PD[LPXP YDOXH UHSUHVHQWV WKH JUHDWHVW SROJRQ FRXQW WKDW FDQ EH DFKLHYHG E\ WKH 3DUWLWLRQ 1RGH¶V GHVFHQGDQWV

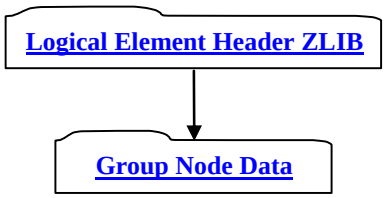
The data format for Polygon Count Range is the same as that described in 7.2.1.1.1.2.1Vertex Count Range.

7.2.1.1.1.3 Group Node Element

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

*URXS 1RGHV FRQWDLQ DQ RUGHUHG OLVW RI UHIHUHQFHV WR RWKHU QRGHV FDOOHG WKH JURXS¶V *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.

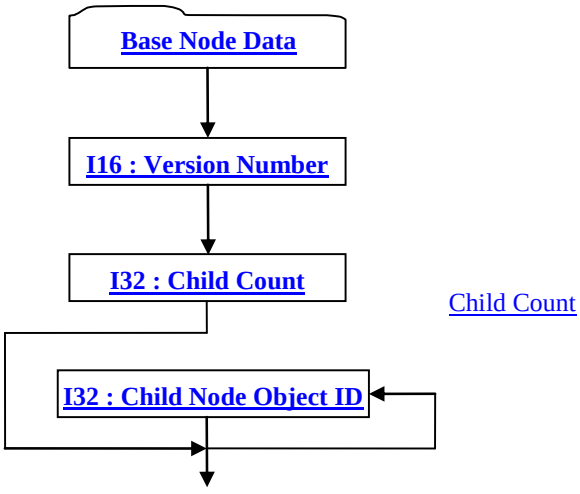
Figure 16: Group Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

7.2.1.1.1.3.1 Group Node Data

Figure 17: Group Node Data collection



Complete description for Base Node Data can be found in 7.2.1.1.1.1.1Base Node Data.

I16 : Version Number

Version Number is the version identifier for this QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQO\ YDOLG YDOXH IRU *URXS Node Data.

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.

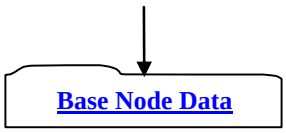
7.2.1.1.1.4 Instance Node Element

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

An Instance Node 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.

For example, a Group Node could use Instance Nodes to instance the same Shape Node several times, applying different material properties and matrix transformations to each instance. Note that this could also be done by using Group Nodes instead of Instance Nodes, but Instance Nodes require fewer resources.

Figure 18: Instance Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Node Data can be found in 7.2.1.1.1.1Base Node Data.

I16: Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDOXH IRU Instance Node Element.

I32 : Child Node Object ID

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

7.2.1.1.1.5 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 (e.g. B-Rep data, PMI data).

Figure 19: Part Node Element data collection

Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Meta Data Node Data can be found in 7.2.1.1.1.6.1Meta Data Node Data.

I16 : Version Number

Version Number is the version identifier for this node. 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQO\ YDOLG YDOXH IRU 3DUW nodes.

I32: Reserved Field

Reserved Fie

7.2.1.1.1.6.1 Meta Data Node Data

Figure 21: Meta Data Node Data collection

Complete description for Group Node Data can be found in 7.2.1.1.1.3.1Group Node Data.

I16 : Version Number

Version Number is the version identifier for this data. Version QXPEHU ³ [´LV FXUUHQW\ WKH RQ\ YDOLG YDOXH for Meta Data Node Data.

7.2.1.1.1.7LOD Node Element

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

An LOD Node 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.

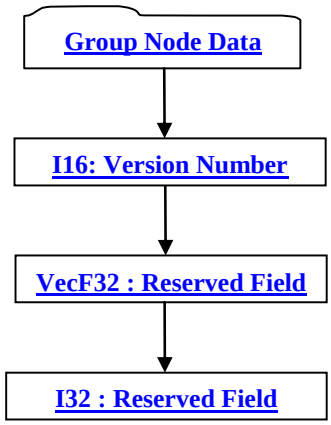
Each alternative representation could be a sub-assembly where the alternative representation is a group node with an assembly of children.

Figure 22: LOD Node Element data collection

Complete description for Log m

7.2.1.1.1.7.1 LOD Node Data

Figure 23: LOD Node Data collection



Complete description for Group Node Data can be found in 7.2.1.1.1.3.1Group Node Data.

I16: Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDOXH IRU /2' Node Data.

VecF32 : Reserved Field

Reserved Field is a vector data field reserved for future JT format expansion.

I32 : Reserved Field

Reserved Field is a data field reserved for future JT format expansion.

7.2.1.1.1.8 Range LOD Node Element

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

Range LOD Nodes hold a list of alternate representations and the ranges over which those representations are appropriate. Range Limits indicate the distance between a specified center 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 24: Range LOD Node Element data collection

Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for LOD Node Data can be found in 7.2.1.1.1.7.1 LOD Node Data

I16: Version Number

Version Number is the version LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQO\ YDOLG YDOXH IRU 5DQJH LOD Node Data.

VecF32 : Range Limits

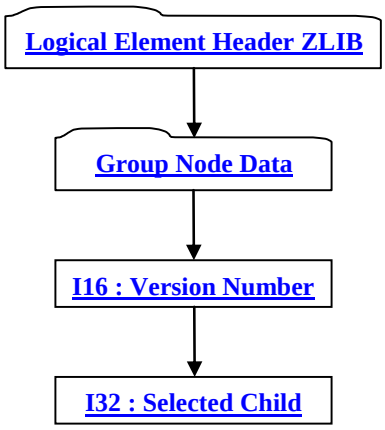
Range Limits indicate the WCS distance between a specified center point and the eye point, within which the corresponding alternate representation is appropriate. It is not required that the count of range limits is equivalent to the number of alternative representations. These valueV DUH FRQVLGHUHG ³VRIW YDOXHV´ in that loaders/viewers of JT data are free to throw these values away and compute new values based on their desired LOD selection semantics.

Best practices 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 center 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 : Center

Center specifies the X,Y,Z coordinates for the NCS center point upon which alternative representation selection eye distance computations are based. Typically this lo42 .00a29267 G(b)T2 0 Td (ce)Tj00017B6 Td [(y)18.0042(p01.0001 Td (m)Tj 0.00T2 j 0.000345

Figure 25: Switch Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Group Node Data can be found in 7.2.1.1.1.3.1Group Node Data.

I16 : Version Number

Version Number is the version identifier for WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQO\ YDOLG YDOXH IRU 6ZLWFK nodes.

I32 : Selected Child

Selected Child is the index for the selected child node. 9DOLG 6HOHFWHG &KLOG YDOXHV UHVLGH ZLWKLQ WKH IROORZLQJ UDQJH ³-1 < Selected &KLOG &RXQW´ Where ³- ´ LQGLFDWHV WKDW QR FKLOG LV WR EH VHOHFWHG DQG ³&KLOG &RXQW´ LV WKH GDWD ILHOG YDOXH from 7.2.1.1.1.3.1Group Node Data.

7.2.1.1.1.10 Shape Node Elements

Shape 1RGH (OHPHQWV DUH ³0HDI´ QRGHV ZLWKLQ WKH /6* VWUXFWXUH DQG FRQWDLQ RU UHIHUHQFH WKH JHRPHWULF VKDSH GHILQLWLRQ GDWD (e.g. 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 JT file, allows a -7 ILOH UHDGHU WR EH VWUXFWXUHG WR VXSSRUW WKH ³EHVW SUDFWLFH´ RI GHOD\LQJ WKH ORDGLQJ UHDGLQJ RI DVVRFLDWHG GDWD XQWLO LW LV actually needed. Complete descriptions for Late Loaded Property Atom Elements and Shape LOD Segments can be found in 0 Late Loaded Property Atom Element and 7.2.2 Shape LOD Segment respectively.

There are several types of Shape Node Elements which the JT format supports. The following sub-sections document the various Shape Node Element types.

7.2.1.1.1.10.1 Base Shape Node Element

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

Base Shape Node Element represents the simplest form of a shape node that can exist within the LSG.

Figure 26: Base Shape Node Element data collection

Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

7.2.1.1.1.10.1.1 Base Shape Data

Figure 27: Base Shape Data collection

Complete description for Base Node Data can be found in 7.2.1.1.1.1.1Base Node Data

I16: Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQO\ YDOLG YDOXH IRU %DVH Shape Data.

BBoxF32 : Reserved Field

Reserved Field is a data field reserved for future JT format expansion.

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 descendents. This value is stored in NCS coordinate space (i.e. values scaled by NCS 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 (JT File) size of the associated/referenced Shape LOD Element. A value of zero indicates that the in memory size is unknown. See 7.2.2.1Shape LOD Element for complete description of Shape LOD Elements. JT 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 specifies the qualitative compression level applied to the associated/referenced Shape LOD Element. See 7.2.2.1Shape LOD Element for 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 JT format by other data stored with both the Shape Node and the Shape LOD Element (e.g. 7.2.1.1.1.10.2.1.1Quantization Parameters), and thus it's not necessary to understand how to map this qualitative value to absolute compression values in order to uncompress/decode the data

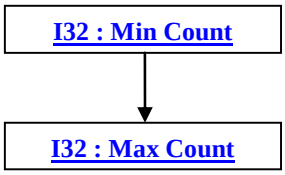
= 0.0	³Lossless´ compression used.
= 0.1	³Minimally LRVV\´ FRPSUHVVLRQ XVHG 7KLV VHWWLQJ JHQHUDOO\ UHVXOWV LQ PRGHVW compression ratios with little if any visual difference when compared to the same images rendered from ³/RVV0HVV´ compressed Shape LOD Element.
= 0.5	³Moderate LRVV\´ FRPSUHVVion used. The setting results in more data loss than ³0LQLPDO\ /RVV\´ DQG WKXV KLJKHU FRPSUHVVLRQ UDWLR LV REWDLQHG 6RPH YLVXDO GLIIHUHQFH ZLOO OLNH\ EH QRWLFHDEOH ZKHQ FRPSDUHG WR WKH VDPH LPDJHV UHQGHUHG IURP ³/RVV0HVV´ compressed Shape LOD Element.
= 1.0	³$JJUHVVLYH /RVV\´ FRPSUHVVLRQ XVHG :LWK WKLV VHWWLQJ DV PXFK GDWD DV SRVVLEOH ZLOO EH thrown away, resulting in highest compression ratio, while still maintaining a modestly useable representation of the underlying data. Visual differences may be evident when FRPSDUHG WR WKH VDPH LPDJHV UHQGHUHG IURP ³/RVV0HVV´ FRPSUHVVHG Shape LOD Element.

7.2.1.1.1.10.1.1.1 Vertex Count Range

Vertex Count Range 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 28: 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. $ YDOXH RI ³- ´ LQGLFDWHV maximum vertex count is unknown.

7.2.1.1.1.10.1.1.2 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 FDQ EH DFKLHYHG E\ WKH 6KDSH 1RGH¶V GHVFHQGDQWV 7KH PD[LPXP YDOXH UHSUHVHQWV WKH JUHDWHVW QRGH FRXQW WKDW FDQ EH DFKLHYHG E\ 6KDSH 1RGH¶V GHVFHQGDQWV For Shape Nodes the minimum and PD[LPXP FRXQW YDOXHV VKRXOG DOZD\V EH HTXDO WR ³ ´

The data format for Node Count Range is the same as that described in 7.2.1.1.1.10.1.1.1Vertex Count Range.

7.2.1.1.1.10.1.1.3 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.

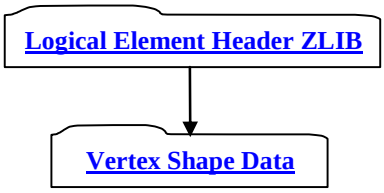
The data format for Polygon Count Range is the same as that described in 7.2.1.1.1.10.1.1.1Vertex Count Range.

7.2.1.1.1.10.2 Vertex Shape Node Element

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

Vertex Shape Node Element represents shapes defined by collections of vertices.

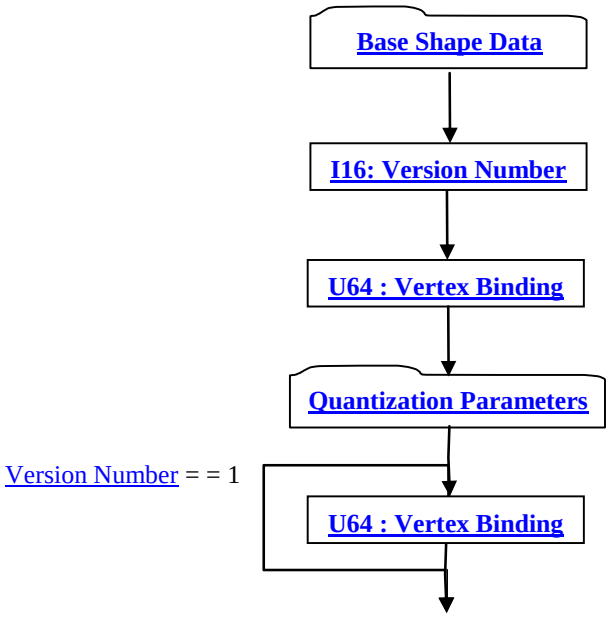
Figure 29: Vertex Shape Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

7.2.1.1.1.10.2.1 Vertex Shape Data

Figure 30: Vertex Shape Data collection



Complete description for Base Shape Data can be found in 7.2.1.1.1.10.1.1 Base Shape Data.

I16: Version Number

Version Number is the version identifier for this node. VHUVLRQ QXPEHU ³ [2´ LV currently the highest valid value for Vertex Shape Data.

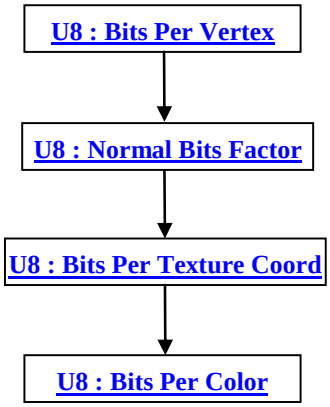
U64 : Vertex Binding

Vertex Bindings is a collection of normal, texture coordinate, and color binding information encoded within a single U64. All undocumented bits are reserved. For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

7.2.1.1.1.10.2.1.1 Quantization Parameters

Quantization Parameters specifies for each shape data type grouping (i.e. Vertex, Normal, Texture Coordinates, Color) the number of quantization bits used for given qualitative compression level. Although these Quantization Parameters values are saved in the associated/referenced Shape LOD Element, they are also saved here so that a JT File loader/reader does not have to load the Shape LOD Element in order to determine the Shape quantization level. See 7.2.2.1Shape LOD Element for complete description of Shape LOD Elements.

Figure 31: Quantization Parameters data collection



U8 : Bits Per Vertex

Bits Per Vertex specifies the number of quantization bits per vertex coordinate component. Value must be within range [0:24] inclusive.

U8 : Normal Bits Factor

Normal Bits Factor is a parameter used to calculate the number of quantization bits for normal vectors. Value must be within range [0:13] inclusive . The actual number of quantization bits per normal is computed using this factor and the following formula: ³%LWV3HU1RUPD0 = 6 + 2 * Normal %LWV)DFWRU´

U8 : Bits Per Texture Coord

Bits Per Texture Coord specifies the number of quantization bits per texture coordinate component. Value must be within range [0:24] inclusive.

U8 : Bits Per Color

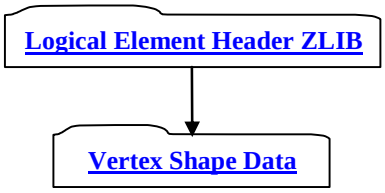
Bits Per Color specifies the number of quantization bits per color component. Value must be within range [0:24] inclusive.

7.2.1.1.1.10.3 Tri-Strip Set Shape Node Element

Object Type ID: 0x10dd1077, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A Tri-Strip Set Shape Node Element defines a collection of independent and unconnected triangle strips. Each strip constitutes one primitive of the set and is defined by one list of vertex coordinates.

Figure 32: Tri-Strip Set Shape Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

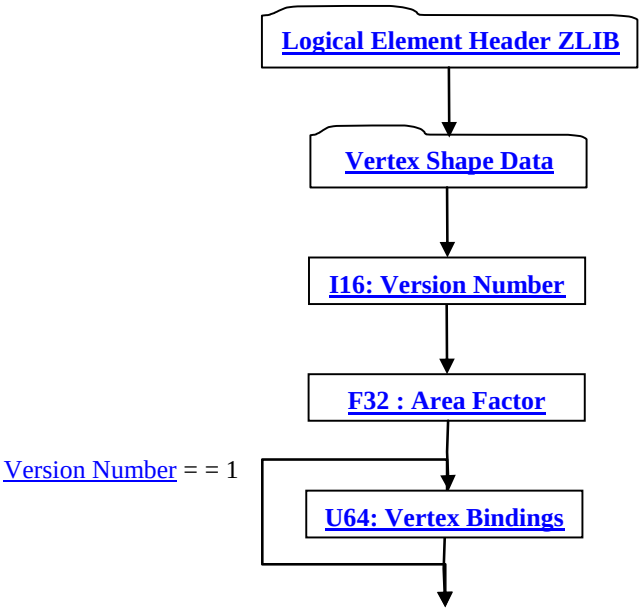
Complete description for Vertex Shape Data can be found in 7.2.1.1.1.10.2.1Vertex Shape Data.

7.2.1.1.1.10.4 Polyline Set Shape Node Element

Object Type ID: 0x10dd1046, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A Polyline Set Shape Node Element defines a collection of independent and unconnected polylines. Each polyline constitutes one primitive of the set and is defined by one list of vertex coordinates.

Figure 33: Polyline Set Shape Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Vertex Shape Data can be found in 7.2.1.1.1.10.2.1Vertex Shape Data.

I16: Version Number

Version Number is the version identifier for this node. 9HUVLRQ QXPEHU ³ [´ LV currently the highest valid value for Polyline Set Shape Data.

F32 : Area Factor

Area Factor specifies a multiplier factor applied to a Polyline Set computed surface area. In JT data viewer applications there may be LOD selection semantics that are based on screen coverage calculations. The so-FDOOHG ´VXUIDFH DUHD´ RI D SROLQH LV computed as if each line segment were a square. This Area Factor turns each edge into a narrow rectangle. Valid Area Factor values lie in the range (0,1].

U64: Vertex Bindings

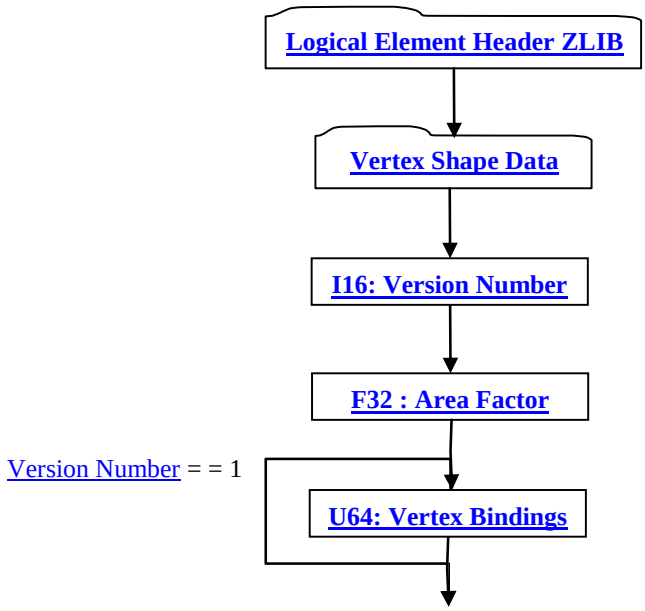
Vertex Bindings is a collection of normal, texture coordinate, and color binding information encoded within a single U64. All undocumented bits are reserved. For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

7.2.1.1.1.10.5 Point Set Shape Node Element

Object Type ID: 0x98134716, 0x0010, 0x0818, 0x19, 0x98, 0x08, 0x00, 0x09, 0x83, 0x5d, 0x5a

A Point Set Shape Node Element defines a collection of independent and unconnected points. Each point constitutes one primitive of the set and is defined by one vertex coordinate.

Figure 34: Point Set Shape Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Vertex Shape Data can be found in 7.2.1.1.1.10.2.1Vertex Shape Data.

I16: Version Number

Version Number is the version identifier for this node. 9HUVLRQ QXPEHU ³ [´ LV currently the highest valid value for Point Set Shape Data.

F32 : Area Factor

Area Factor specifies a multiplier factor applied to the Point Set computed surface area. In JT data viewer applications there may be LOD selection semantics that are based on screen coverage calculations. 7KH FRPSXWHG ³VXUIDFH DUHD´ RI D 3RLQW 6HW is equal to the larger (i.e. whichever is greater) of either WKH DUHD RI WKH 3RLQW 6HW¶V ERXQGLQJ ER[RU ³ ´ $UHD) actor scales the UHVXOW RI WKLV ³VXUIDFH DUHD´ FRPSXWDWLRQ

U64: Vertex Bindings

Vertex Bindings is a collection of normal, texture coordinate, and color binding information encoded within a single U64. All undocumented bits are reserved. For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

7.2.1.1.1.10.6 Polygon Set Shape Node Element

Object Type ID: 0x10dd1048, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A Polygon Set Shape Node Element defines a collection of independent and unconnected polygons. Each polygon constitutes one primitive of the set and is defined by one list of vertex coordinates.

Figure 35: Polygon Set Shape Node Element data collection

Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Vertex Shape Data can be found in 7.2.1.1.1.10.2.1Vertex Shape Data.

7.2.1.1.1.10.7 NULL Shape Node Element

Object Type ID: 0xd239e7b6, 0xdd77, 0x4289, 0xa0, 0x7d, 0xb0, 0xee, 0x79, 0xf7, 0x94, 0x94

A NULL Shape Node Element defines a shape which has no direct geometric primitive representation (i.e. it is empty/NULL). NULL Shape Node Elements are often used as ³proxy/SODFHKROGHU´ nodes within the serialized LSG when the actual Shape LOD data is run time generated (i.e. not persisted).

Figure 36: NULL Shape Node Element data collection

Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Shape Data can be found in 7.2.1.1.1.10.1.1 Base Shape Data.

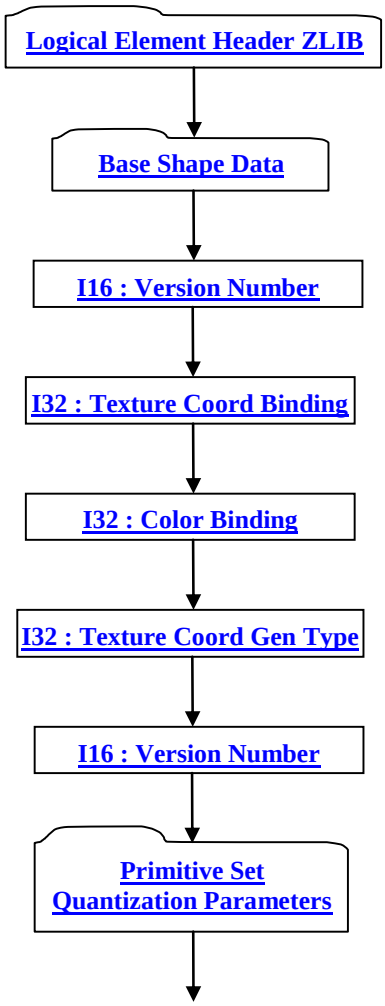
I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDOXH IRU NULL Shape Node Element.

7.2.1.1.1.10.8 Primitive Set Shape Node Element

Object Type ID: 0xe40373c1, 0x1ad9, 0x11d3, 0x9d, 0xaf, 0x0, 0xa0, 0xc9, 0xc7, 0xdd, 0xc2 A\ i

Figure 37: Primitive Set Shape Node Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Shape Data can be found in 7.2.1.1.1.10.1.1 Base Shape Data.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQO\ YDOLG YDOXH IRU Primitive Set Shape Node Element.

I32 : Texture Coord Binding

7H[WXUH &RRUG %LQGLQJ VSHFLILHV KRZ DW ZKDW JUDQXODULW\ WH[WXUH FRRUGLQDWH GDWD LV VXSSOLHG ³ERXQG´ IRU WKH VKDSH LQ WKH associated/referenced Shape LOD Element. Valid values are as follows:

= 0	None. Shape has no texture coordinate data.
= 1	Per Vertex. Shape has texture coordinates for every vertex.

I32 : Color Binding

&RORU %LQGLQJ VSHFLILHV KRZ DW ZKDW JUDQXODULW\ FRORU GDWD LV VXSSOLHG ³ERXQG´ IRU WKH VKDSH LQ WKH DVVRFLDWHG UHferenced Shape LOD Element. Valid values are the same as documented for Texture Coord Binding data field.

I16 : Version Number

Version Number is the version identifier for this element. The value of this Version Number indicates the format of data fields to follow.

= 0	Version 0 Format
= 1	Version 1 Format

I32 : Texture Coord Gen Type

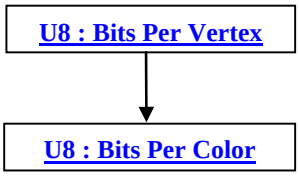
Texture Coord Gen Type specifies how texture coordinates are to be generated.

= 0	Single Tile « ,QGLFDWHV WKDW D VLQJOH FRS\ RI D WH[WXUH LPDJH ZLOO EH DSSOLHG WR VLJQLILFDQW primitive features (i.e. cube face, cylinder wall, end cap) no matter how eccentrically shaped.
= 1	,VRWURSLF « ,PSOLHV that multiple copies of a texture image may be mapped onto eccentric surfaces such that a mapped texel stays approximately square.

7.2.1.1.1.10.8.1 Primitive Set Quantization Parameters

Primitive Set Quantization Parameters specifies for the two shape data type grouping (i.e. Vertex, Color) the number of quantization bits used for given qualitative compression level. Although these Quantization Parameters values are saved in the associated/referenced Shape LOD Element, they are also saved here so that a JT File loader/reader does not have to load the Shape LOD Element in order to determine the Shape quantization level. See 7.2.2.1Shape LOD Element for complete description of Shape LOD Elements.

Figure 38: Primitive Set Quantization Parameters data collection



U8 : Bits Per Vertex

Bits Per Vertex specifies the number of quantization bits per vertex coordinate component. Value must be within range [0:24] inclusive.

U8 : Bits Per Color

Bits Per Color specifies the number of quantization bits per color component. Value must be within range [0:24] inclusive.

7.2.1.1.2 Attribute Elements

Attribute Elements (e.g. color, texture, material, lights, etc.) are placed in LSG as objects associated with nodes. Attribute Elements are not nodes themselves, but can be associated with any node.

For applications producing or consuming JT format data, it is important that the JT format semantics of how attributes are meant to be applied and accumulated down the LSG are followed. If not followed, then consistency between the applications in terms of 3D positioning and rendering of LSG model data will not be achieved.

To that end each attribute type defines its own application and accumulation semantics, but in general attributes at lower levels in the LSG take precedence and replace or accumulate with attributes set at higher levels. Nodes without associated

attributes inherit those of their parents. Attributes inherit only from their parents, thus D QRGH¶V DWWributes do not affect that QRGH¶V VLEOLQJV 7KH URRW RI D SDUWLWLRQ LQKHULWV WKH DWWULEXWHV LQ HIIHFW DW WKH UHIHUHQJ SDUWLWLRQ QRGH

$WWULEXWHV FDQ EH GHFODUHG ³ILQDO´ VHH 7.2.1.1.2.1.1Base Attribute Data), which terminates accumulation of that attribute type at that attribute and propagates the accumulated values there to all descendants of the associated node. Descendants can explicitly do a one-shot override RI ³ILQDO´ XVLQJ WKH DWWULEXWH ³IRUFH´ IODJ VHH 7.2.1.1.2.1.1Base Attribute Data), but do not by default. 1RWH WKDW ³IRUFH´ GRHV QRW WXUQ 2)) ³ILQDO´ ± it is simply a one-VKRW RYHUULGH RI ³ILQDO´ IRU WKH VSHFLILF DWWULEXWH PDUNHG DV ³IRUFLQJ´ An analogy IRU WKLV ³IRUFH´ DQG ³ILQDO´ LQWHUDFWLRQ LV WKDW ³ILQDO´ LV D EDFN-door in the attribute DFFXPXODWLRQ VHPDQWLFV DQG WKDW ³IRUFH´ LV D GRJJ\-door in the back-door!

7.2.1.1.2.1Common Attribute Data Containers

7.2.1.1.2.1.1 Base Attribute Data

Figure 39: Base Attribute Data collection

I16: Version Number

I16: Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOG YDOXH for Base Shape Data.

U8 : State Flags

State Flags is a collection of flags. The flags are combined using the binary OR operator and store various state information for Attribute Elements; such as indicating that the attributes accumulation is final. All undocumented bits are reserved.

0x01	Accumulation Final flag. 3URYLGHV D PHDQV WR WHUPLQDWH D SDUWLFXODU DWWULEXWH W\SH¶V DFFXPXODWLRQ DW DQ\ QRGH RI WKH /6* and thereby force all descendants to have that value of the attribute. = 0 ± Accumulation is to occur normally = 1 ± AFFXPXODWLRQ LV ³ILQDO´
0x02	Accumulation Force flag. Provides a way to assign nodes in LSG, attributes that must not be overridden by ancestors. = 0 ± $FFXPXODWLRQ RI WKLV DWWULEXWH REH\V DQFHVWRU¶V)LQDO IODJ VHWWLQJ = 1 ± Accumulation of this attribute is forced (overrides ancestor¶s Final flag setting)
0x04	Accumulation Ignore Flag. Provides a way to indicate that the attribute is to be ignored (not accumulated). = 0 ± Attribute is to be accumulated normally (subject to values of Force/Final flags) = 1 ± Attribute is to be ignored.
0x08	Attribute Persistable Flag.

	Provides a way to indicate that the attribute is to be persistable to a JT file. = 0 ± Attribute is to be non-persistable. = 1 ± Attribute is to be persistable.

U32 : Field Inhibit Flags

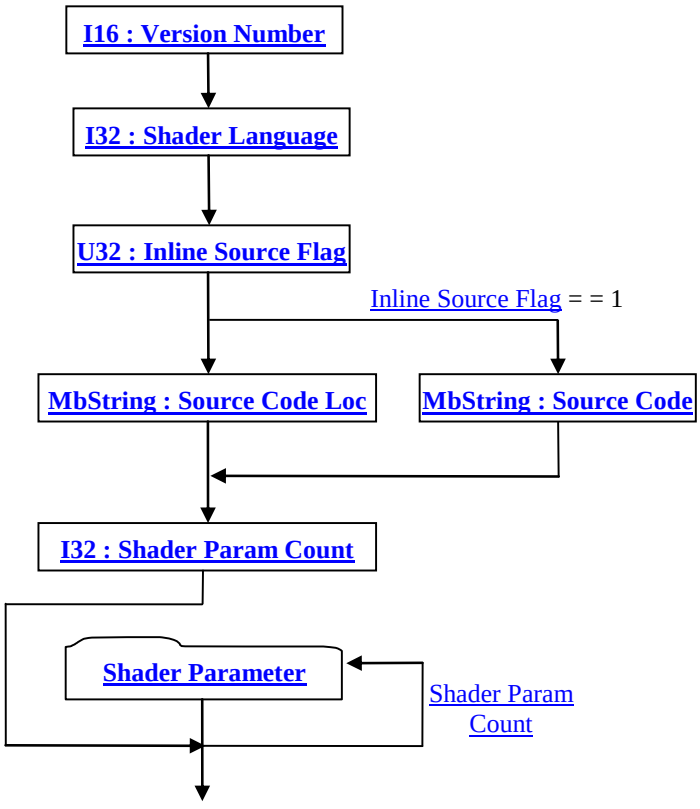
Field Inhibit Flags is a collection of flags. The flags are combined using the binary OR operator and store the per attribute value accumulation flag. Each value present in an Attribute Element is given a field number ranging from 0 to 31. If the field¶s corresponding bit in Inhibit Flags is set, then the field should not participate in attribute accumulation. All bits are reserved.

See each particular Attribute Element (e.g. Material Attribute Element) for a description of bit field assignments for each attribute value.

7.2.1.1.2.1.2 Base Shader Data

The JT v9 file format is able to represent vertex- and fragment shader programs in GLSL source code form together with parameter bindings for both. The shader source code can be specified inline directly in the JT file, or as a filename containing the shader source code.

Figure 40: Base Shader Data collection



I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV GDWD FRQHFWLRQ 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDXH.

I32 : Shader Language

Shader Language specifies the Shader program language. JT v9.5 only supports the GLSL Shading Language.

= 0	None
= 2	***/6/ 3*/ 6KDGLQJ /DQJXDJH´ DV** defined by the Architectural Review Board of OpenGL, the governing body of OpenGL [7]**.**

U32 : Inline Source Flag

,QOLQH 6RXUFH)ODJ VSHFLILHV ZKHWKHU WKH VKDGHU¶V 3VRXUFH FRGH´ LV VWRUHG ZLWKLQ WKLV -7 ILOH RU LQ VRPH RWKHU H[WHUQDOO referenced file. Valid values include the following:

= 0	Source code stored in an externally referenced file.
= 1	Source code stored within this JT file.

MbString : Source Code

6RXUFH &RGH LV WKH VKDGHU¶V VRXUFH FRGH LQ Shader Language programming language.

MbString : Source Code Loc

6RXUFH &RGH /RF VSHFLILHV WKH ILOH QDPH IRU WKH H[WHUQDO ILOH FRQWDLQLQJ WKH VKDGHU¶V VRXUFH FRGH

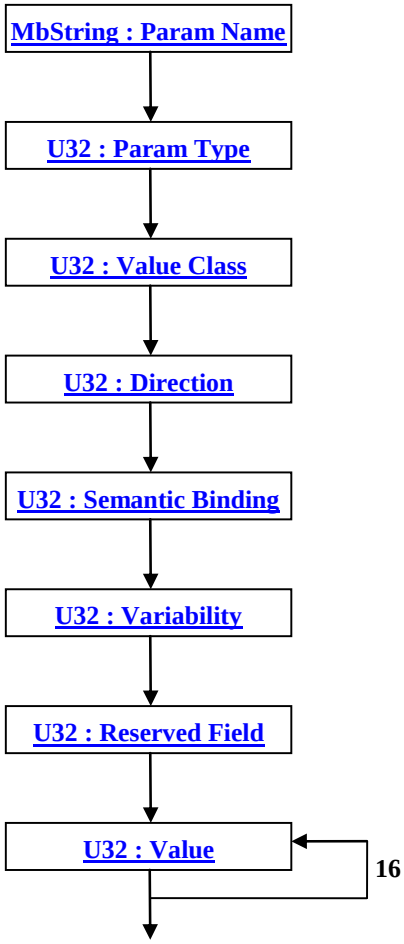
I32 : Shader Param Count

Shader Param Count specifies the number of shader parameters.

7.2.1.1.2.1.2.1 Shader Parameter

Shader Parameter data collection defines a Shader input and/or output parameter. A list of Shader Parameters represents the **UXQWLPH 0LQNDJH RI WKH VKDGHU SURJUDP LQWR WKH *38¶V GDWD VWUHDPV**

Figure 41: Shader Parameter data collection



MbString : Param Name

Param Name specifies the shader parameter name.

U32 : Param Type

Param Type specifies the shader parameter type. Valid types include the following:

= 0	Unknown
= 1	Boolean
= 2	Integer
= 3	Float
= 4	Vector of two Integer values.
= 5	Vector of three Integer values
= 6	Vector of four Integer values
= 7	Vector of two Float values
= 8	Vector of three Float values
= 9	Vector of four Float values
= 10	2 x 2 matrix of Float values
= 11	3 x 3 matrix of Float values
= 12	4 x 4 matrix of Float values

= 13	Texture Object/Unit number bound to current 1D texture sampler
= 14	Texture Object/Unit number bound to current 2D texture sampler
= 15	Texture Object/Unit number bound to current 3D texture sampler
= 16	Texture Object/Unit number bound to current rectangle map texture sampler
= 17	Texture Object/Unit number bound to current cube map texture sampler
= 18	Texture Object/Unit number bound to current 1D shadow map texture sampler
= 19	Texture Object/Unit number bound to current 2D shadow map texture sampler

U32 : Value Class

9DOXH &ODVV VSHFLILHV WKH VKDGHU SDUDPHWHU ³YDOXH FODVV´ 9DOLG YDOXHV LQFOXGH the following:

= 0	Unknown class
= 1	Immediate class.
= 2	Semantic class (i.e. Shader Parameter is implicitly tied/bound to a piece of OpenGL graphics system state (e.g. OpenGL ModelView matrix) or JT graphics system state (e.g. diffuse material color)). The actual graphics state that the parameter is bound to is indicated by value in Value data field.

U32 : Direction

Direction specifies whether the shader parameter is an input, output, or input/output parameter. Valid values include the following:

= 0	Unknown
= 1	Input parameter
= 2	Output parameter
= 3	Both an Input and an Output parameter.

U32 : Semantic Binding

6HPDQWLF %LQGLQJ VSHFLILHV WKH ³SHU YHUWH[LQSXW DQG RU RXWSXW´ RU WKH ³SHU IUDJPHQW LQSXW DQG RU RXWSXW´ WKLV VKDGHU parameter is associated with (i.e. bound to). Valid values, including their input/output applicability to vertex and fragment VKDGHUV DUH DV IROORZV QRWH WKDW 1 $ LQGLFDWHV µ1RW $SSOLFDEOH´

Value	Binding Description	Vertex Shader Applicability	Fragment Shader Applicability
= 0	Unknown		
= 1	None		
= 2	Position	Input/Output	Input
= 3	Normal	Input	N/A
= 4	Binormal	Input	N/A
= 5	Blend Indices	Input	N/A
= 6	Blend Weight	Input	N/A
= 7	Tangent	Input	N/A
= 8	Point Size	Input/Output	Input
= 10	Texture Coordinate 0	Input/Output	Input
= 11	Texture Coordinate 1	Input/Output	Input
= 12	Texture Coordinate 2	Input/Output	Input
= 13	Texture Coordinate 3	Input/Output	Input
= 14	Texture Coordinate 4	Input/Output	Input
= 15	Texture Coordinate 5	Input/Output	Input
= 16	Texture Coordinate 6	Input/Output	Input
= 17	Texture Coordinate 7	Input/Output	Input

Value	Binding Description	Vertex Shader Applicability	Fragment Shader Applicability
= 20	Fog Coordinate	Output	Input
= 21	Primary Color	Output	Input
= 22	Secondary Color	Output	Input
= 23	Primary Color	N/A	Output
= 24	Depth Value	N/A	Output

U32 : Variability

Variability specifies how often the value of the parameter is allowed to change. Valid values include the following:

= 0	Unknown
= 1	Constant (a parameter that takes on a single value and never changes)
= 2	Uniform (a parameter that may take on a different value each time the shader is invoked but remains the same for all vertices or fragments processed by the shader)
= 3	Varying (a parameter which may change with every vertex or fragment processed by the shader)

U32 : Reserved Field

Reserved Field is a data field reserved for future JT format expansion.

U32 : Value

Value specifies the shader parameter values treated as a U32 array of bytes. The maximum number of bytes required to store all possible Param Type and Value Class dependent values is 64 bytes and thus there are 16 U32 values stored. The interpretation of the Value data is Param Type and Value Class dependent as follows:

)RU ³,PPHGLDWH´ Value Class parameters (i.e. Value Class = = 1), the interpretation of the Value data is dependent upon the Param Type value.

)RU ³6HPDQWLF´ Value Class parameters, the Value data is to be interpreted as a single U32 with all the possible values documented in Appendix B: Semantic Value Class Shader Parameter Values.

7.2.1.1.2.2 Material Attribute Element

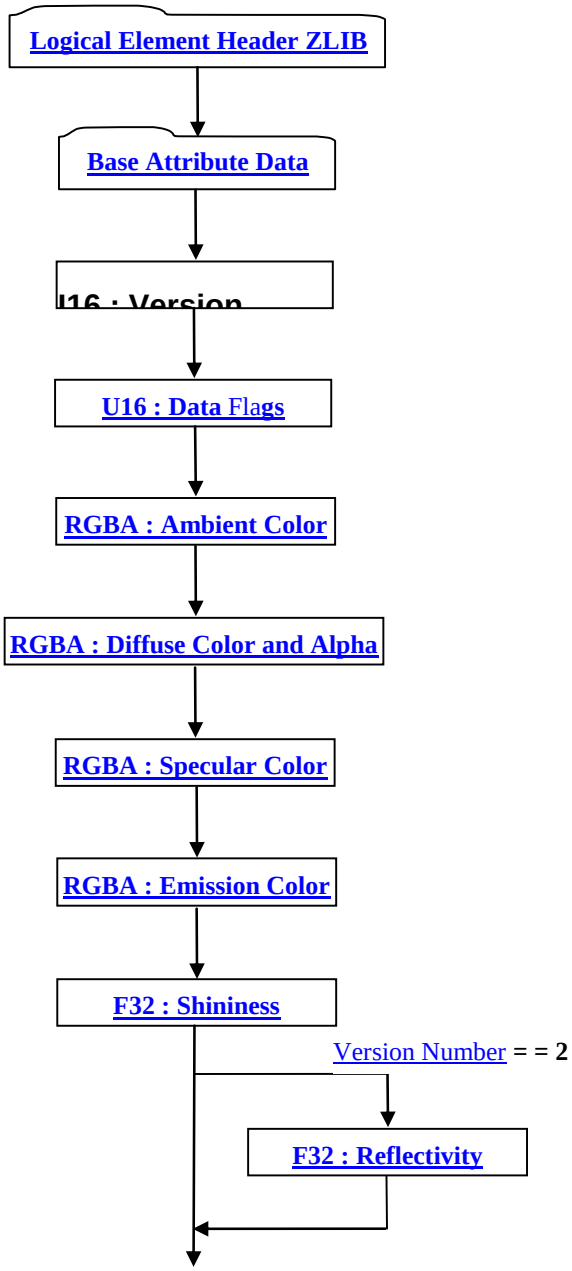
Object Type ID: 0x10dd1030, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Material Attribute Element defines the material properties of a object. JT format LSG traversal semantics state that material attributes accumulate down the LSG by replacement.

The Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments for the Material Attribute Element data fields, are as follows:

Field Inhibit Flag Bit	Data Field(s) Bit Applies To
0	Ambient Common RGB Value, Ambient Color
1	Diffuse Color and Alpha (Legacy)
2	Specular Common RGB Value, Specular Color
3	Emission Common RGB Value, Emission Color
4	Blending Flag, Source Blending Factor, Destination Blending Factor
5	Override Vertex Color Flag
6	Material Reflectivity
7	Diffuse Color
8	Diffuse Alpha

Figure 42: Material Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16 : Version Number

Version Number is the version identifier for this element. The value of this Version Number indicates the format of data fields to follow.

= 1	Version-1 Format

= 2	Version-2 Format

U16 : Data Flags

Data Flags is a collection of flags and factor data. The flags and factor data are combined using the binary OR operator. The flags store information to be used for interpreting how to read subsequent Material data fields. All undocumented bits are reserved.

0x0010	Blending Flag. Blending is a color combining operation in the graphics pipeline that happens just before writing a color to the framebuffer. If Blending is ON then incoming fragment RGBA color values are used (based on Source Blend Factor) and H[LVWLQJ IUDPHEXIIHU¶V 5*%$ color values are used (based on Destination Blend Factor) to blend between the incoming fragment RGBA and the current frame buffer RGBA to arrive at a new RGBA color to write into the framebuffer. If Blending is OFF then incoming fragment RGBA color is written directly into framebuffer unmodified (i.e. completely overriding existing framebuffer RGBA color). Additional information on how one might leverage the Blending Flag and Blending Factors to render an image can be found in the references listed in section 3 References and Additional Information. = 0 ± Blending OFF. = 1 ± Blending ON
0x0020	Override Vertex Colors Flag. If ON, then a shape¶s per vertex colors are to be overridden by the accumulated Material color. = 0 ± Override OFF = 1 ± Override ON
0x07C0	Source Blend Factor (stored in bits 6 ± 10 or in binary notation 0000011111000000). If Blending Flag enabled, this value indicates KRZ WKH LQFRPLQJ IUDJPHQW¶V L H WKH VRXUFH RGBA color values are to be used to blend ZLWK WKH FXUUHQW IUDPHEXIIHU¶V L H WKH GHVWLQDWLRQ RGBA color values. Additional information on the interpretation of the Blending Factor values and how one might leverage them to render an image can be found in reference [4] listed in section 3 References and Additional Information. = 0 ± Interpret same as OpenGL **GL_ZERO** Blending Factor = 1 ± Interpret same as OpenGL **GL_ONE** Blending Factor = 2 ± Interpret same as OpenGL **GL_DST_COLOR** Blending Factor = 3 ± Interpret same as OpenGL **GL_SRC_COLOR** Blending Factor = 4 ± Interpret same as OpenGL **GL_ONE_MINUS_DST_COLOR** Blending Factor = 5 ± Interpret same as OpenGL **GL_ONE_MINUS_SRC_COLOR** Blending Factor = 6 ± Interpret same as OpenGL **GL_SRC_ALPHA** Blending Factor = 7 ± Interpret same as OpenGL **GL_ONE_MINUS_SRC_ALPHA** Blending Factor = 8 ± Interpret same as OpenGL **GL_DST_ALPHA** Blending Factor = 9 ± Interpret same as OpenGL **GL_ONE_MINUS_DST_ALPHA** Blending Factor = 10 ± Interpret same as OpenGL **GL_SRC_ALPHA_SATURATE** Blending Factor
0xF800	Destination Blend Factor (stored in bits 11 ± 15 or in binary notation 1111100000000000).). If Blending Flag enabled, this value indicates KRZ WKH FXUUHQW IUDPHEXIIHU¶V WKH GHVWLQDWLRQ RGBA color values are to be XVHG WR EOHQG ZLWK WKH LQFRPLQJ IUDJPHQW¶V WKH VRXUFH 5*%$ color values. Additional information on the interpretation of the Blending Factor values and how one might leverage them to render an image can be found in reference [4] listed in section 3 References and Additional Information. = 0 ± Interpret same as OpenGL **GL_ZERO** Blending Factor = 1 ± Interpret same as OpenGL **GL_ONE** Blending Factor = 2 ± Interpret same as OpenGL **GL_DST_COLOR** Blending Factor

	= 3 ± Interpret same as OpenGL **GL_SRC_COLOR** Blending Factor = 4 ± Interpret same as OpenGL **GL_ONE_MINUS_DST_COLOR** Blending Factor = 5 ± Interpret same as OpenGL **GL_ONE_MINUS_SRC_COLOR** Blending Factor = 6 ± Interpret same as OpenGL **GL_SRC_ALPHA** Blending Factor = 7 ± Interpret same as OpenGL **GL_ONE_MINUS_SRC_ALPHA** Blending Factor = 8 ± Interpret same as OpenGL **GL_DST_ALPHA** Blending Factor = 9 ± Interpret same as OpenGL **GL_ONE_MINUS_DST_ALPHA** Blending Factor = 10 ± Interpret same as OpenGL **GL_SRC_ALPHA_SATURATE** Blending Factor

RGBA : Ambient Color

Ambient Color specifies the ambient red, green, blue, alpha color values of the material.

RGBA : Diffuse Color and Alpha

Diffuse Color and Alpha specify the diffuse red, green, blue color components, and alpha value of the material.

RGBA : Specular Color

Specular Color specifies the specular red, green, blue, alpha color values of the material.

RGBA : Emission Color

Emission Color specifies the emissive red, green, blue, alpha color values of the material.

F32 : Shininess

Shininess is the exponent associated with specular reflection and highlighting. Shininess controls the degree with which the specular highlight decays. Only values in the range [1,128] are valid.

F32 : Reflectivity

Reflectivity specifies the material reflectivity of the material. It represents the fraction of light reflected in the mirror direction by the material. Only values in the range [0.0, 1.0] are valid.

7.2.1.1.2.3Texture Image Attribute Element

Object Type ID: 0x10dd1073, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

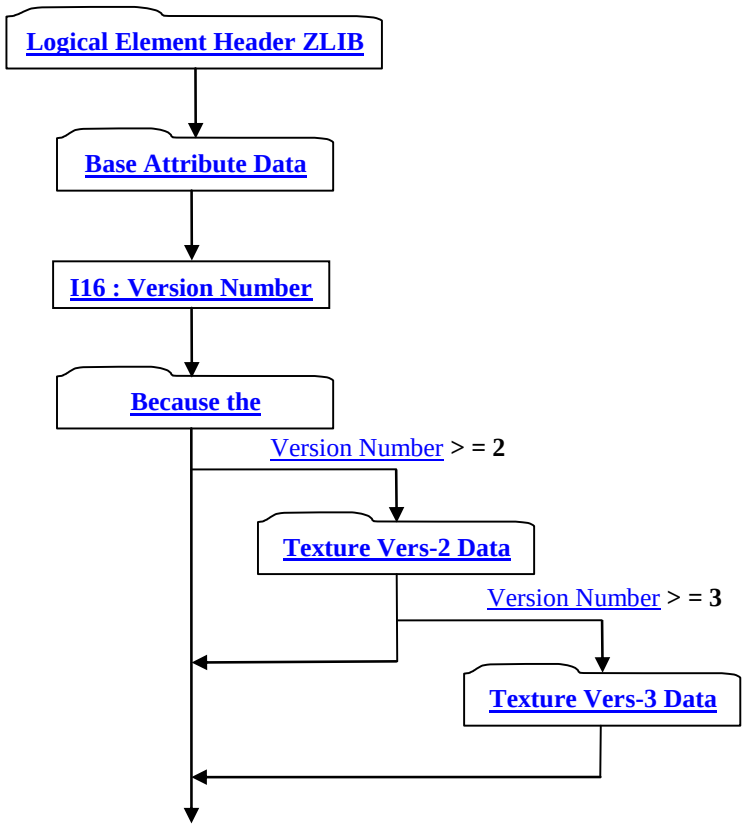
Texture Image Attribute Element defines a texture image and its mapping environment. JT format LSG traversal semantics state that texture image attributes accumulate down the LSG by replacement on a *per channel* basis. See below for more information on texture image channels.

Note that additional information on the interpretation of the various Texture Image Attribute Element data fields can be found in the OpenGL references listed in section 3 References and Additional Information.

The Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments for the Texture Image Attribute Element data fields, are as follows:

Field Inhibit Flag Bit	Data Field(s) Bit Applies To
0	I32 : Texture Type, Mipmap Image Texel Data, MbString : External Storage Name, Shared Image Flag
1	Border Mode, Border Color
2	Mipmap Minification Filter, Mipmap Magnification Filter
3	S-Dimen Wrap Mode, T-Dimen Wrap Mode, R-Dimen Wrap Mode
4	Blend Type, Blend Color
5	Texture Transform
6	Tex Coord Gen Mode, Tex Coord Reference Plane
8	Internal Compression Level

Figure 43: Texture Image Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1 Base Attribute Data.

Complete description for Texture Vers-1 Data can be found in 7.2.1.1.2.3.1 Texture Vers-1 Data.

Complete description for Texture Vers-2 Data can be found in 7.2.1.1.2.3.2 Texture Vers-2 Data.

Complete description for Texture Vers-3 Data can be found in 7.2.1.1.2.3.3 Texture Vers-3 Data.

I16 : Version Number

Version Number is the version identifier for this element. The value of this Version Number indicates the format of data fields to follow.

= 1	Version-1 Format
= 2	Version-2 Format
= 3	Version-3 Format

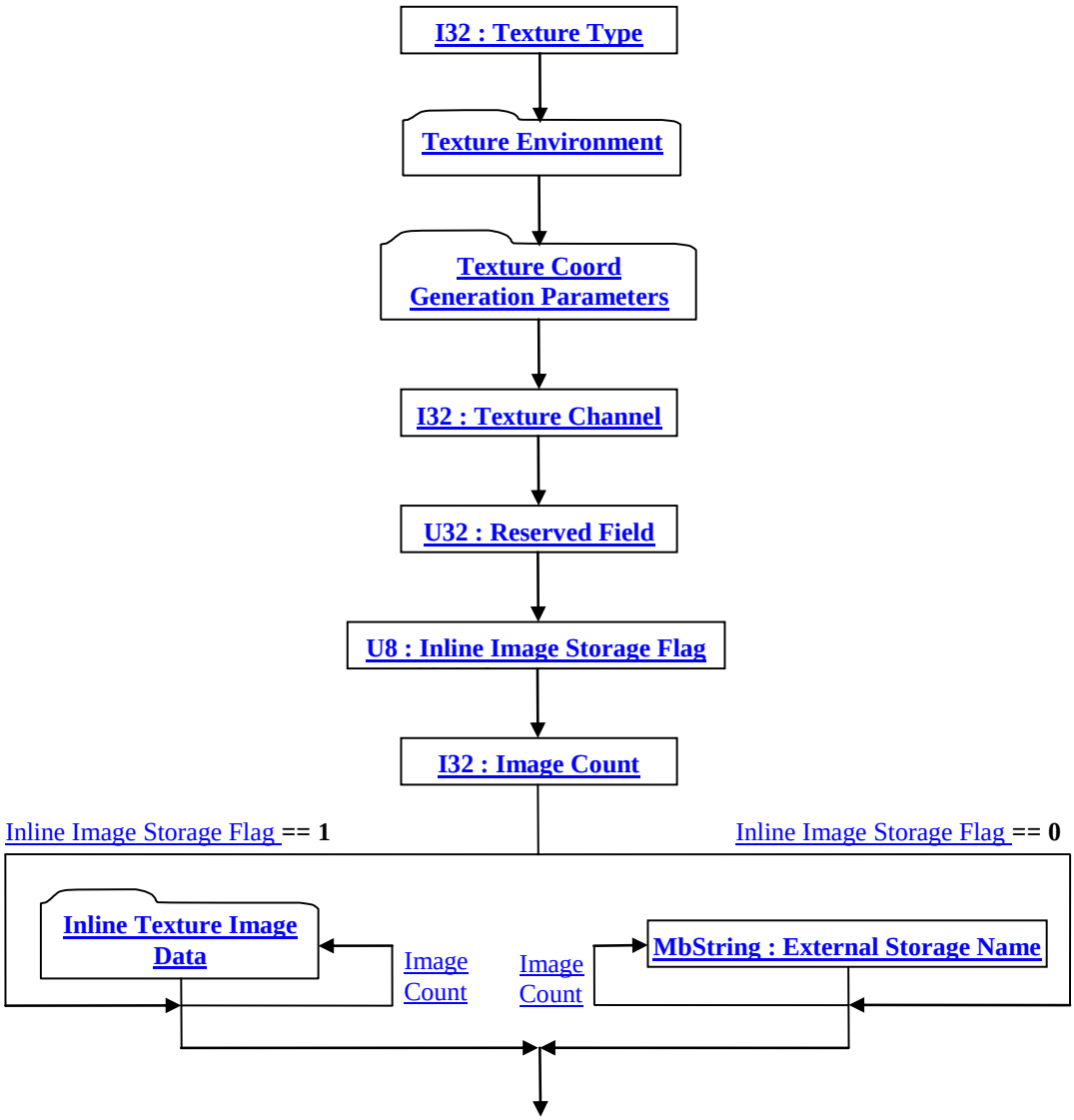
Because the 7.2.1.1.2.3Texture Image Attribute Element has undergone major upgrades during the lifetime of the JT v9 file format, the attribute has a complex version structure to be mindful of. Usually, when a data element in the JT file is versioned, it is for the purpose of merely adding a few pieces of new data onto the end of the existing data format. In this way, older viewers and readers of the JT file that do not yet know about higher local versions will naturally read the lower-numbered version blocks and ignore the higher-numbered ones they do not know how to read. This is sometimes the case with Texture Image Attribute Element, but sometimes not. Entirely new texture types with no analogous lower-level functionality have been added. In these cases, the most sensible thing for an older reader to do it to ignore the texture image entirely as if it were not even present in the JT file.

In order to support this sensible fallback mechanism, the following two general rules are followed: 1) a given texture image is written at the lowest version level that completely captures its contents, and 2) lower-order Texture Vers Data blocks are written with a "stub" texture.

7.2.1.1.2.3.1 Texture Vers-1 Data

Texture Vers-1 Data format is stored in JT file if the Texture Image Element is a vanilla/basic texture image (i.e. if texture does not use any advanced features as described in 7.2.1.1.2.3.2Texture Vers-2 Data and 7.2.1.1.2.3.3Texture Vers-3 Data). However, advanced textures *also* write a Texture Vers-1 Data block because of the need to be backward-compatible with older readers that may not understand Vers-2 and Vers-3 data.

Figure 44: Texture Vers-1 Data collection



Complete details for Texture Environment can be found in 7.2.1.1.2.3.1.1Texture Environment.

Complete details for Texture Coord Generation Parameters can be found in 7.2.1.1.2.3.1.2Texture Coord Generation Parameters.

Complete details for Inline Texture Image Data can be found in 7.2.1.1.2.3.1.3Inline Texture Image Data.

I32 : Texture Type

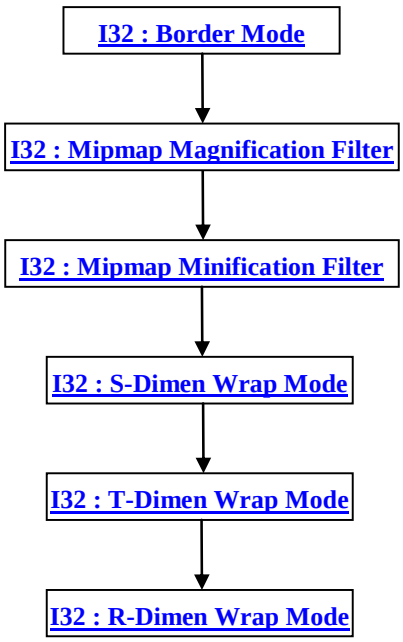
Texture Type specifies the type of texture.

= 0	None.
= 1	One-Dimensional. A one-dimensional texture has a height (T-Dimension) and depth (R-'LPHQVLRQ HTXDO WR ³ ´ DQG QR WRS RU ERWWRP ERUGHU
= 2	Two-Dimensional. A two-dimensional texture has a depth (R-Dimension) HTXDO WR ³ ´
= 3	Three-Dimensional. A three-dimensional texture can be thought of as layers of two-dimensional sub image rectangles arranged in a sequence.
= 4	Bump Map. A bump map texture is a texture where the image texel data (e.g. RGB color values) represents surface normal XYZ components.
= 5	Cube Map. A cube map texture is a texture cube centered at the origin and formed by a set of six two-dimensional texture images.
= 6	Depth Map. A depth map texture is a texture where the image texel data represents depth values.

I32 : Texture Channel

Texture Channel specifies the texture channel number for the Texture Image Element. For purposes of multi-texturing, the -7 FRQFHSW RI D WH[WXUH FKDQQHO FRUUHVSRQGV WR WKH 2SHQ*/ FRQFHSW RI D ³WH[WXUH XQLW´ 7KH 7H[WXUH &KDQQHO YDOXH PXVW EH between 0 and 31 inclusive. Best practices suggest that renderer of JT data ignore all but channel-0 if the̶ignore 0 Td (H)

Figure 45: Texture Environment data collection



I32 : Border Mode

Border Mode specifies the texture border mode.

= 0	No border.
= 1	Constant Border Color. Indicates that the texture has a constant border color whose value is defined in data field Border Color.
= 2	Explicit. Indicates that a border texel ring is present in the texture image definition.

I32 : Mipmap Magnification Filter

Mipmap Magnification Filter specifies the texture filtering method to apply when a single pixel on screen maps to a tiny portion of a texel.

= 0	None.
= 1	Nearest. Texel with coordinates nearest the center of the pixel is used.
= 2	Linear. A weighted linear average of the 2 x 2 array of texels nearest to the center of the pixel is used. For one-dimensional texture is average of 2 texels. For three dimensional texel is 2 x 2 x 2 array.

I32 : Mipmap Minification Filter

Mipmap Minification Filter specifies the texture filtering method to apply when a single pixel on screen maps to a large collection of texels.

= 0	None.
= 1	Nearest. Texel with coordinates nearest the center of the pixel is used.
= 2	Linear. A weighted linear average of the 2 x 2 array of texels nearest to the center of the pixel is used. For one-dimensional texture is average of 2 texels. For three-dimensional texture is 2 x 2 x 2 array.
= 3	Nearest in Mipmap. Within an individual mipmap, the texel with coordinates nearest the center of the pixel is used.
= 4	Linear in Mipmap. Within an individual mipmap, a weighted linear average of the 2 x 2 array of texels nearest to the center of the pixel is used. For one-dimensional texture is average of 2 texels. For three-dimensional texture is 2 x 2 x 2 array
= 5	Nearest between Mipmaps. Within each of the adjacent two mipmaps, selects the texel with coordinates nearest the center of the pixel and then interpolates linearly between these two selected mipmap values.
= 6	Linear between Mipmaps. Within each of the two adjacent mipmaps, computes value based on a weighted linear average of the 2 x 2 array of texels nearest to the center of the pixel and then interpolates linearly between these two computed mipmap values.

I32 : S-Dimen Wrap Mode

S-Dimen Wrap Mode specifies the mode for handling texture coordinates S-Dimension values outside the range [0, 1].

= 0	None.
= 1	Clamp. Any values greater than 1.0 are set to 1.0; any values less than 0.0 are set to 0.0
= 2	Repeat Integer parts of the texture coordinates are ignored (i.e. retains only the fractional component o texture coordinates greater than 1.0 and only one-minus the fractional component of values less than zero). Resulting in copies of the texture map tiling the surface
= 3	0LUURU 5HSHDW /LNH 5HSHDW H[FHSW WKH VXUIDFH WLOHV ³IOLS-IORS´ UHVXOWLQJ LQ DQ DOWHUQDWLQJ mirror pattern of surface tiles.
= 4	Clamp to Edge. Border is always ignored and instead texel at or near the edge is chosen for coordinates outside the range [0, 1]. Whether the exact nearest edge texel or some average of the nearest edge texels is used is dependent upon the mipmap filtering value.
= 5	Clamp to Border. Nearest border texel is chosen for coordinates outside the range [0, 1]. Whether the exact nearest border texel or some average of the nearest border texels is used is dependent upon the mipmap filtering value.

I32 : T-Dimen Wrap Mode

T-Dimen Wrap Mode specifies the mode for handling texture coordinates T-Dimension values outside the range [0, 1]. Same mode values as documented for S-Dimen Wrap Mode.

I32 : R-Dimen Wrap Mode

R-Dimen Wrap Mode specifies the mode for handling texture coordinates R-Dimension values outside the range [0, 1]. Same mode values as documented for S-Dimen Wrap Mode.

I32 : Blend Type

Blend Type contains information indicating how the values in the texture map are to be modulated/combined/blended with the original color of the surface or some other alternative color to compute the final color to be painted on the surface. Additional information on the interpretation of the Blend Type values and how one might leverage them to render an image can be found in reference [4] listed in section 3 References and Additional Information.

= 0	None.
= 1	Decal. Interpret same as OpenGL **GL_DECAL** environment mode.
= 2	Modulate. Interpret same as OpenGL **GL_MODULATE** environment mode.
= 3	Replace. Interpret same as OpenGL **GL_REPLACE** environment mode.
= 4	Blend. Interpret same as OpenGL **GL_BLEND** environment mode.
= 5	Add. Interpret same as OpenGL **GL_ADD** environment mode.
= 6	Combine. Interpret same as OpenGL **GL_COMBINE** environment mode.

I32 : Internal Compression Level

Internal Compression Level specifies a data compression hint/recommendation that a JT file loader is free to follow for internally (in memory) storing texel data. This setting does not affect how image texel data is actually stored in JT files or other externally referenced files.

= 0	None. No compression of texel data.
= 1	Conservative. Lossless compression of texel data.
= 2	Moderate. Texel components truncated to 8-bits each.
= 3	Aggressive. Texel components truncates to 4-bits each (or 5 bits for RGB images).

RGBA : Blend Color

%0HQG &R0RU VSHFLILHV WKH FR0RU WR EH XVHG IRU WKH ³%0HQG´ PRGH RI Blend Type operations.

RGBA : Border Color

Border Color specifies the constant border color WR XVH IRU ³&0DPS WR %RUGHU´ VW\0H ZUDS PRGHV ZKHQ WKH WH[WXUH LWVH0I GRHV not have a border.

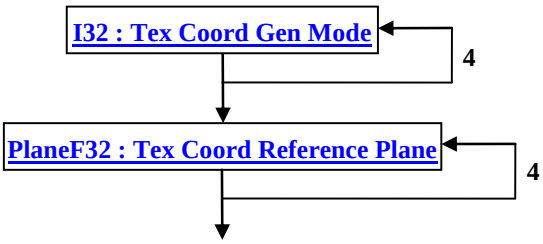
Mx4F32 : Texture Transform

Texture Transform defines the texture coordinate transformation matrix. A renderer of JT data would typically apply this transform to texture coordinates prior to applying the texture.

7.2.1.1.2.3.1.2 Texture Coord Generation Parameters

Texture Coord Generation Parameters contains information indicating if and how texture coordinate components should be automatically generated for each of the 4 components (S, T, R, Q) of a texture coordinate.

Figure 46: Texture Coord Generation Parameters data collection



I32 : Tex Coord Gen Mode

Tex Coord Gen Mode specifies the texture coordinate generation mode for each component (S, T, R, Q) of texture coordinate. There are four mode values stored, one for each component of texture coordinate. The mode values are stored in S, T, R, Q order.

= 0	None. No texture coordinates automatically generated.
= 1	Model Coordinate System Linear. Texture coordinates computed as a distance from a reference plane specified in model coordinates.
= 2	View Coordinate System Linear. Texture coordinates computed as a distance from a reference plane specified in view coordinates.
= 3	Sphere Map. Texture coordinates generated based on spherical environment mapping.
= 4	Reflection Map. Texture coordinates generated based on cubic environment mapping.
= 5	Normal Map. Texture coordinates computed/set by copying vertex normal in view coordinates to S, T, R.

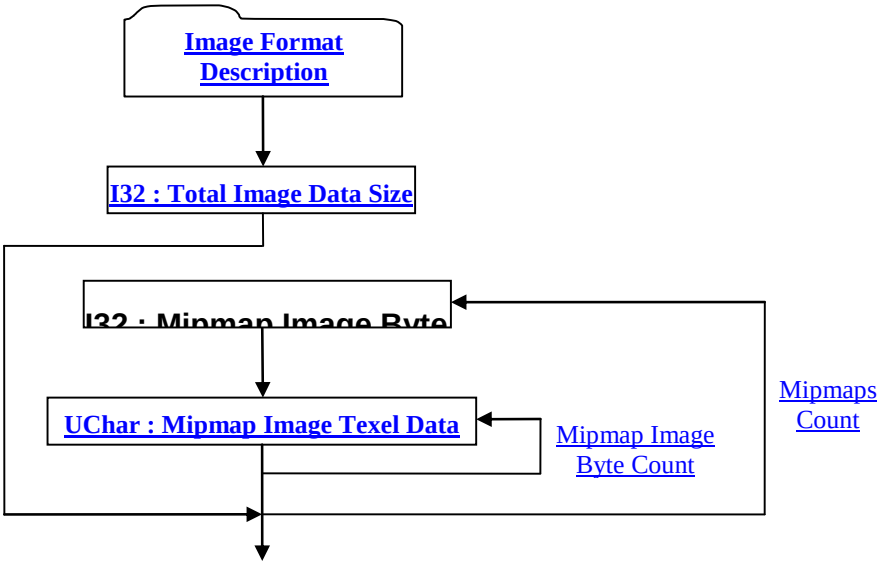
PlaneF32 : Tex Coord Reference Plane

5HIHUHQFH 3ODQH VSHFLILHV WKH UHIHUHQFH SODQH XVHG IRU ³0RGHO &RRUGLQDWH 6\VWHP /LQHDU´ DQG ³9LHZ &RRUGLQDWH 6\VWHP /LQHDU´ WH[WXUH FRRUGLQDWH JHQHUDWLRQ PRGHV 7KUHH DUH IRXU 5HIHUHQFH 3ODQHV VWRUHG RQH IRU HDFK FRPSRQHQW RI WH[WXUH coordinate. The RefHUHQFH 3ODQHV DUH VWRUHG LQ 6 7 5 4 RUGHU (YHQ LI D FRPSRQHQW ³7H[&RRUG *HQ 0RGH´ LV RQH WKDW does not require a reference plane, dummy reference planes are still stored in JT file.

7.2.1.1.2.3.1.3 Inline Texture Image Data

Inline Texture Image Data is a collection of data defining the texture format properties and image texel data for one texture image. Inline Texture Image Data is only present if data field Inline Image Storage Flag HTXDO ³ ´ ,I SUHVHQW WKHUH will be data field Image Count number of Inline Texture Image Data instances.

Figure 47: Inline Texture Image Data collection



Complete description for Image Format Description can be found in 7.2.1.1.2.3.1.3.1Image Format Description.

I32 : Total Image Data Size

Total Image Data Size specifies the total length, in bytes, of the on-disk representation for all mipmap images. This byte total does not include the I32 : Mipmap Image Byte Count

data field storage (4 bytes per) for each mipmap.

I32 : Mipmap Image Byte Count

Mipmap Image Byte Count specifies the length, in bytes, of the on-disk representation of the next mipmap image.

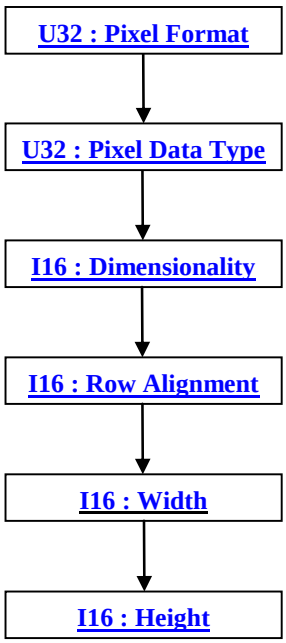
UChar : Mipmap Image Texel Data

Mipmap Image Texel Data is t[illegible] block of image data. The length of this field in bytes is specified by the value of data field Mipmap Image Byte Count.

7.2.1.1.2.3.1.3.1 Image Format Description

The Image Format Description is a collection of data defining the pixel format, data type, size, and other miscellaneous characteristics of the texel image data.

Figure 48: Image Format Description data collection



U32 : Pixel Format

Pixel format specifies the format of the texture image pixel data. Depending on the format, anywhere from one to four elements of data exists per texel.

= 0	No format specified. Texture mapping is not applied.
= 1	RGB: A red color component followed by green and blue color components
= 2	RGBA: A red color component followed by green, blue, and alpha color components
= 3	LUM: A single luminance component
= 4	LUMA: A luminance component followed by an alpha color component.
= 5	A single stencil index.
= 6	A single depth component
= 7	A single red color component
= 8	A single green color component
= 9	A single blue color component

= 10	A single alpha color component
= 11	A blue color component, followed by green and red color components
= 12	A blue color component, followed by green , red, and alpha color components
= 13	A depth component, followed by a stencil component

U32 : Pixel Data Type

Pixel Data Type specifies the data type used to store the per texel data. If the Pixel Format represents a multi component value (e.g. red, green, blue) then each value requires the Pixel Data Type number of bytes of storage (e.g. a Pixel Format 7\SHRI³ ´ZLWK3L[HO'DWD7\7\pe(\)1n001 U9148 T672 8848594 eZ55 T2440230 Tf(7)Tj0.0.000259148 j0.0

U8 : Shared Image Flag

Shared Image Flag is a flag indicating whether this texture image is shareable with other Texture Image Element attributes.

= 0	Image is not shareable with other Texture Image Elements.
= 1	Image is shareable with other Texture Image Elements.

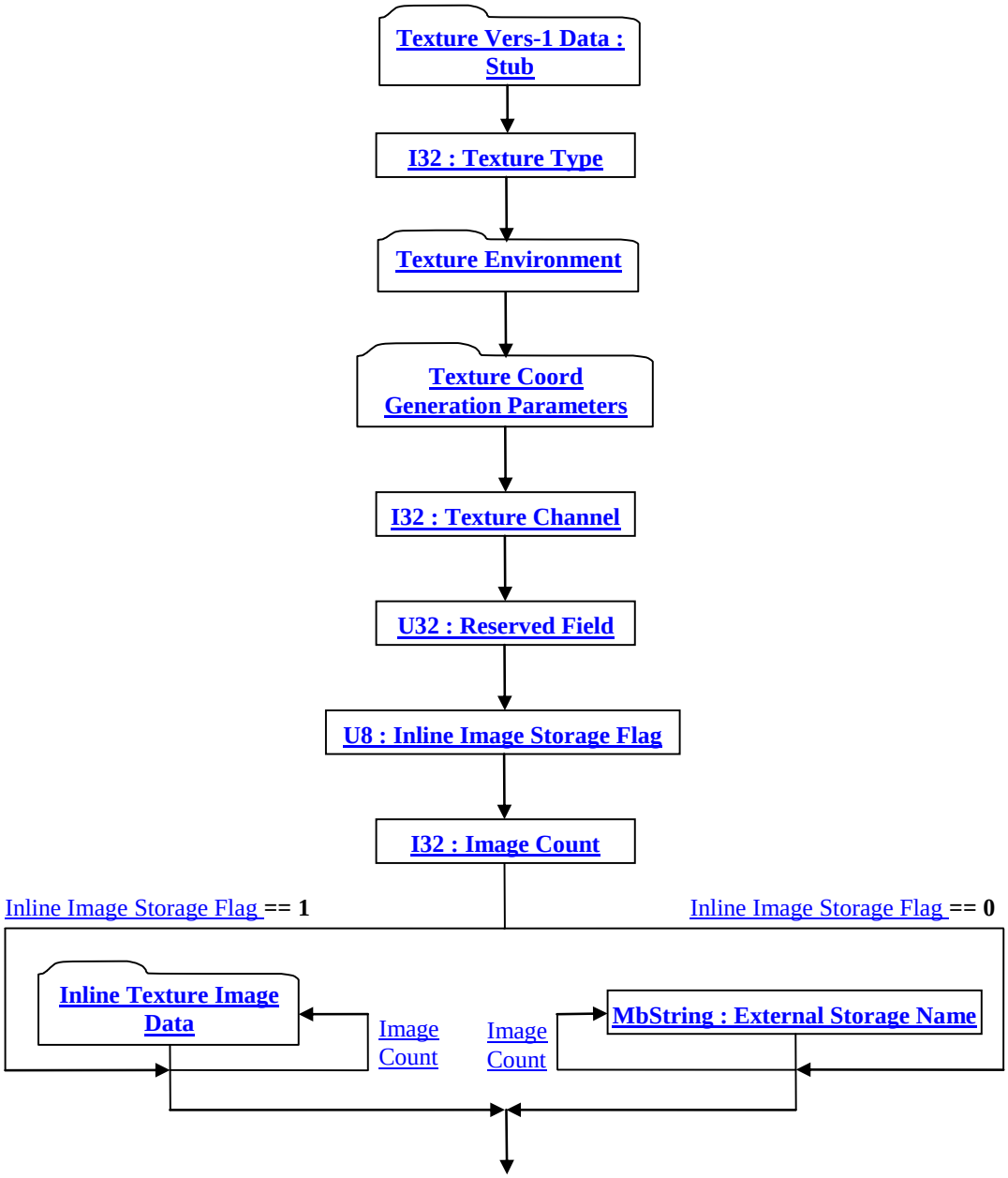
I16 : Mipmaps Count

Mipmaps Count specifies the number of mipmap images. A value of 1 indicates that no mipmaps are used. A value greater than 1 indicates that mipmaps are present all the way down to a 1-by-1 texel.

7.2.1.1.2.3.2 Texture Vers-2 Data

Texture Vers-2 Data collection supports texturing effects not representable in the Texture Vers-1 Data format (e.g. more precise texture types, automatic texture channel, etc.). Any Texture Image Attribute Element using the Texture Vers-2 Data IRUPDW ZLOO FRQWDLQ D ³GHJHQHUDWH´ Texture Vers-1 Data block, where Image Count GDWD ILHOG KDV D YDOXH RI ³ ´

Figure 49: Texture Vers-2 Data collection



Complete details for Texture Environment can be found in 7.2.1.1.2.3.1.1Texture Environment.

Complete details for Texture Coord Generation Parameters can be found in 7.2.1.1.2.3.1.2Texture Coord Generation Parameters.

Complete details for Inline Texture Image Data can be found in 7.2.1.1.2.3.1.3Inline Texture Image Data.

Texture Vers-1 Data : Stub

This is a dummy block written with its I32 : Texture Type field set to "None". This block is included so that older readers that do not understand Texture Vers-2 Data will read an "empty" texture.

I32 : Texture Type

Texture Type specifies the type of texture. 7KHUH LV D FRPSOHWH UHVWUXFWXULQJ DQG UHGHILQLWLRQ RI ZKDW D ³WH[WXUH W\SH´ LPSOLHV in Texture Vers-2 Data. It is a much stronger concept now, that not only describes generally what the texture image contains, but also defines precisely what the texture is being used for. ,Q WKH IROORZLQJ OLVW ³LPDJH´ UHIHUV WR DQ LPDJH WH[WXUH ³SUH-OLW´ LQGLFDWHV WKDW WKH LPDJH WH[WXUH LV WR EH DSSOLHG EHIRUH OLJKWLQJ ZKHQ UHQGHULQJ WKH REMHFW WR ZKLFK LW LV DSSOLHG DQG ³SRVW-OLW´ indicates that the image texture is to be applied after lighting. A gloss map is a pre-lit texture that applies itself to the specular material component of lighting instead of the diffuse component. A light map is an environment texture (texture at infinity surrounding the whole model) that serves as a source of illumination during shading calculations.

Texture Type	Description	Explicit Channel	Auto Channel
= 0	None.	N/A	N/A
= 1	One-Dimensional post-lit image texture.	Yes	No
= 2	Two-Dimensional post-lit image texture.	Yes	No
= 3	Three-Dimensional post-lit image texture.	Yes	No
= 4	Two-Dimensional 3-component tangent-space normal map.	No	Yes
= 5	Cube post-lit image texture.	Yes	No
= 7	Cube pre-lit image texture.	Yes	No
= 8	One-Dimensional pre-lit image texture.	Yes	No
= 9	Two-Dimensional pre-lit image texture.	Yes	No
= 10	Three-Dimensional pre-lit image texture.	Yes	No
= 11	Cube environment map.	No	Yes
= 12	One-Dimensional gloss map (specular) texture.	No	Yes
= 13	Two-Dimensional gloss map (specular) texture.	No	Yes
= 14	Three-Dimensional gloss map (specular) texture.	No	Yes
= 15	Cube gloss map (specular) texture.	No	Yes
= 16	Two-Dimensional 1-component bumpmap.	No	Yes
= 17	Two-Dimensional 3-component world-space normal map.	No	Yes
= 18	Two-Dimensional sphere environment map.	No	Yes
= 19	Two-Dimensional latitude/longitude environment map.	No	Yes
= 20	Two-Dimensional spherical diffuse light map.	No	Yes
= 21	Cube diffuse light map.	No	Yes
= 22	Two-Dimensional latitude/longitude diffuse light map.	No	Yes
= 23	Two-Dimensional spherical specular light map.	No	Yes
= 24	Cube specular light map.	No	Yes
= 25	Two-Dimensional latitude/longitude specular light map.	No	Yes

I32 : Texture Channel

Texture Channel specifies the texture channel number for the Texture Image Element. For purposes of multi-texturing, the JT FRQFHSW RI D WH[WXUH FKDQQHO FRUUHVSRQGV WR WKH 2SHQ*/ FRQFHSW RI D ³WH[WXUH XQLW´ 7KH 7H[WXUH &KDQQHO YDOXH PXVW EH between -1 and 31 inclusive. The value -1 is accepted to denote a texture whose channel number is to be automatically assigned. This assignment will never displace another texture with an explicit texture channel assignment from its slot. Best practices suggest that renderer of JT data ignore all but channel-0 if the renderer does not support multi-textured geometry. $OVR IRU SXUSRVHV RI EHQGLQJ DQ\ UHQGHUHU RI -7 GDWD VKRXOG HQVXUH WKDW KLJKHU QXPEHUHG WH[WXUH FKDQQHOV ³EHQGR RYHU´ lower numbered ones.

Pre- and post-lit image textures must specify an explicit texture channel. All other texture types must specify -1 for their texture channel.

U32 : Reserved Field

Reserved Field is a data field reserved for future JT format expansion.

U8 : Inline Image Storage Flag

Inline Image Storage Flag is a flag that indicates whether the texture image is stored within the JT File (i.e. inline) or in some other external file.

= 0	Texture image stored in an external file.
= 1	Texture image stored inline in this JT file.

I32 : Image Count

Image Count specifies the QXPEHU RI WH[WXUH LPDJHV $ ³&XEH 0DS´ I32 : Texture Type must have six images while all other Texture Types may only have one image.

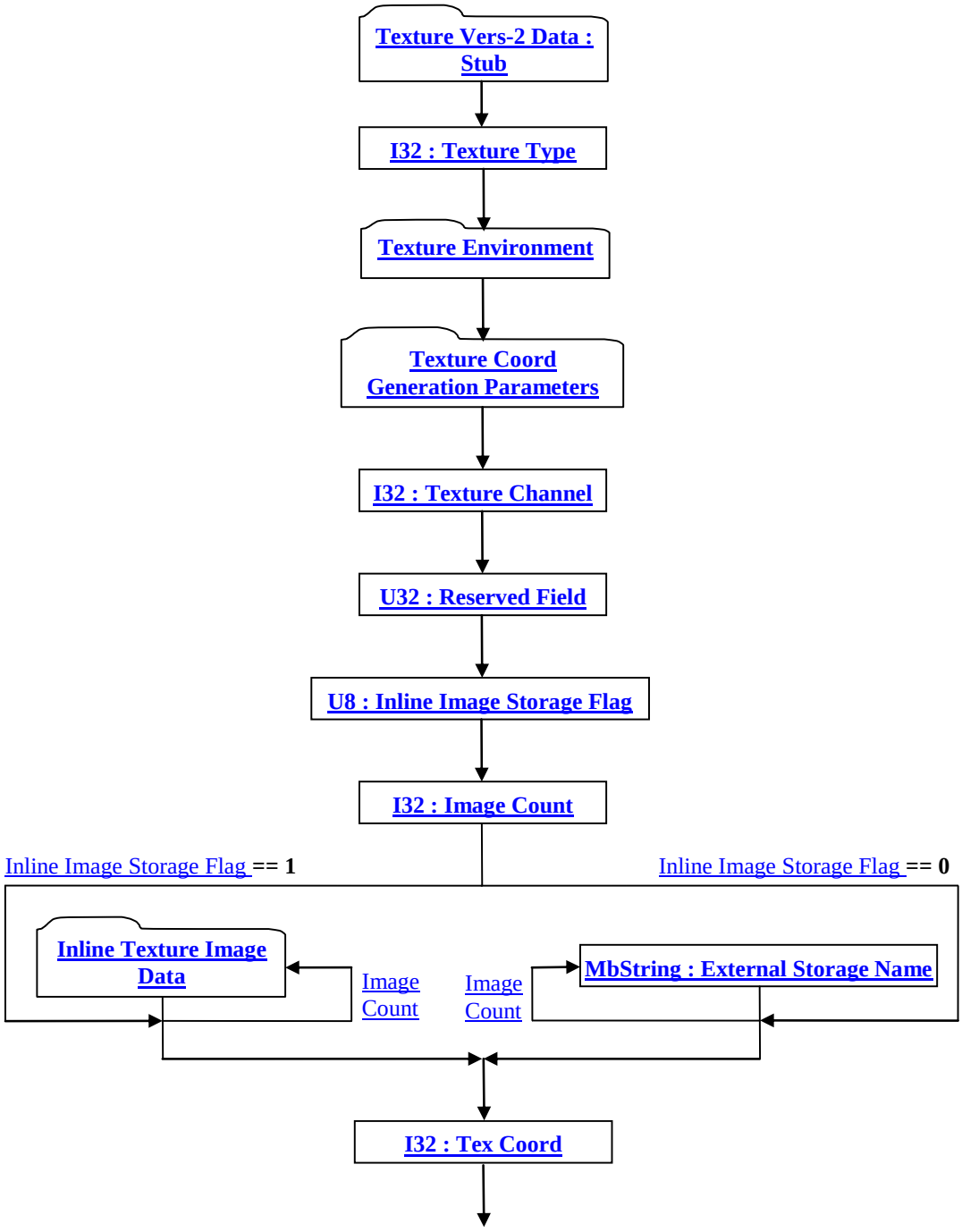
MbString : External Storage Name

External Storage Name is a string identifying the name of an external texture image storage. External Storage Name is only present if data field Inline Image Storage Flag equals 0. If present, there will be data field Image Count number of External Storage Name instances. This External Storage Name string is a relative path based name for the texture image file. Where ³UHODWLYH SDWK´ VKRXOG EH LQWHUSUHWHG WR PHDQ WKH VWULQJ FRQWDLQV WKH ILOH QDPH DORQJ ZLWK DQ\ DGGLWLRQDO SDWK LQIRUPDWLRQ WKat locates the texture image file relative to the location of the referencing JT file.

7.2.1.1.2.3.3 Texture Vers-3 Data

Texture Vers-3 Data collection supports texturing effects not representable in the Texture Vers-1 Data format or the Texture Vers-2 Data format (e.g. texture coordinate channel, separator texture type, and texture channel greater than 31). Any Texture Image Attribute Element using the Texture Vers-3 Data IRUPDW ZLOO FRQWDLQ D ³GHJHQHUDWH´ Texture Vers-1 Data block, and a ³degenerate´ Texture Vers-2 Data block, where Image Count data field has a value of 0 and the Texture Type will be set to None.

Figure 50: Texture Vers-3 Data collection



Complete details for Texture Environment can be found in 7.2.1.1.2.3.1.1Texture Environment.

Complete details for Texture Coord Generation Parameters can be found in 7.2.1.1.2.3.1.2Texture Coord Generation Parameters.

Complete details for Inline Texture Image Data can be found in 7.2.1.1.2.3.1.3Inline Texture Image Data.

Texture Vers-2 Data : Stub

This is a dummy block written with its I32 : Texture Type field set to "None". This block is included so that older readers that do not understand Texture Vers-3 Data will read an "empty" texture.

I32 : Texture Type

Texture Type specifies the type of texture. A new texture type, separator texture, is defined in Texture Vers-3 Data to support resetting the texture accumulation state mid-graph. Shadow maps and prefiltered light maps, however, are a general exception to this rule. In the following list ³LPDJH´ UHIHUV WR DQ LPDJH WH[WXUH ³SUH-OLW´ LQGLFDWHV WKDW WKH LPDJH WH[WXUH LV WR EH DSSOLHG EHIRUH OLJKWLQJ ZKHQ UHQGHULQJ WKH REMHFW WR ZKLFK LW LV DSSOLHG DQG ³SRVW-OLW´ LQGLFDWHV WKDW WKH LPDJH WH[WXUH LV WR EH DSSOLHG after lighting. A gloss map is a pre-lit texture that applies itself to the specular material component of lighting instead of the diffuse component. A light map is an environment texture (texture at infinity surrounding the whole model) that serves as a source of illumination during shading calculations.

Texture Type	Description	Explicit Channel	Auto Channel
= 0	None.	N/A	N/A
= 1	One-Dimensional post-lit image texture.	Yes	No
= 2	Two-Dimensional post-lit image texture.	Yes	No
= 3	Three-Dimensional post-lit image texture.	Yes	No
= 4	Two-Dimensional 3-component tangent-space normal map.	No	Yes
= 5	Cube post-lit image texture.	Yes	No
= 7	Cube pre-lit image texture.	Yes	No
= 8	One-Dimensional pre-lit image texture.	Yes	No
= 9	Two-Dimensional pre-lit image texture.	Yes	No
= 10	Three-Dimensional pre-lit image texture.	Yes	No
= 11	Cube environment map.	No	Yes
= 12	One-Dimensional gloss map (specular) texture.	No	Yes
= 13	Two-Dimensional gloss map (specular) texture.	No	Yes
= 14	Three-Dimensional gloss map (specular) texture.	No	Yes
= 15	Cube gloss map (specular) texture.	No	Yes
= 16	Two-Dimensional 1-component bumpmap.	No	Yes
= 17	Two-Dimensional 3-component world-space normal map.	No	Yes
= 18	Two-Dimensional sphere environment map.	No	Yes
= 19	Two-Dimensional latitude/longitude environment map.	No	Yes
= 20	Two-Dimensional spherical diffuse light map.	No	Yes
= 21	Cube diffuse light map.	No	Yes
= 22	Two-Dimensional latitude/longitude diffuse light map.	No	Yes
= 23	Two-Dimensional spherical specular light map.	No	Yes
= 24	Cube specular light map.	No	Yes
= 25	Two-Dimensional latitude/longitude specular light map.	No	Yes
=26	Resets texture state except shadow map and light maps.	N/A	N/A

I32 : Texture Channel

Texture Channel specifies the texture channel number for the Texture Image Element. For purposes of multi-texturing, the JT FRQFHSW RI D WH[WXUH FKDQQHO FRUUHVSRQGV WR WKH 2SHQ*/ FRQFHSW RI D ³WH[WXUH XQLW´ 7KH 7H[WXUH &KDQQHO YDOXH PXVW EH between -1 and 2,147,483,647 inclusive. The value -1 is accepted to denote a texture whose channel number is to be automatically assigned. This assignment will never displace another texture with an explicit texture channel assignment from its slot. Best practices suggest that renderer of JT data ignore all but channel-0 if the renderer does not support multi-textured geometry. Also for purposes of blending, any renderer of JT data should ensure that higher numbered texture channels ³EOHQG RYHU´ ORZHU QXPEHUHG RQHV

Pre- and post-lit image textures must specify an explicit texture channel. All other texture types must specify -1 for their texture channel.

U32 : Reserved Field

Reserved Field is a data field reserved for future JT format expansion.

U8 : Inline Image Storage Flag

Inline Image Storage Flag is a flag that indicates whether the texture image is stored within the JT File (i.e. inline) or in some other external file.

= 0	Texture image stored in an external file.
= 1	Texture image stored inline in this JT file.

I32 : Image Count

Image Count specifies tKH QXPEHU RI WH[WXUH LPDJHV $ ³&XEH 0DS´ I32 : Texture Type must have six images while all other Texture Types should only have one image.

MbString : External Storage Name

External Storage Name is a string identifying the name of an external texture image storage. External Storage Name is only present if data field Inline Image Storage Flag HTXDOV ³ ´ ,I SUHVHQW WKHUH ZLOO EH GDWD ILHOG Image Count number of External Storage Name instances. This External Storage Name string is a relative path based name for the texture image file. Where ³UHODWLYH SDWK´ VKRXOG EH LQWHUSUHWHG WR PHDQ WKH VWULng contains the file name along with any additional path information that locates the texture image file relative to the location of the referencing JT file.

I32 : Tex Coord Channel

Tex Coord Channel specifies the channel number for texture coordinate generation. Value must be within range [-1, 2147483647] inclusive.

7.2.1.1.2.4 Draw Style Attribute Element

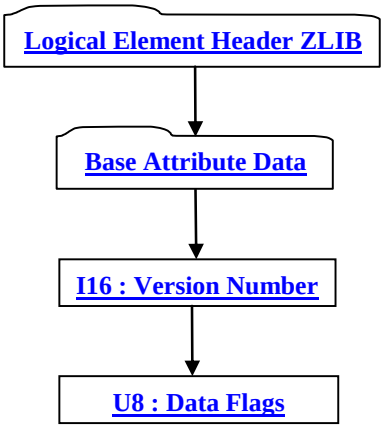
Object Type ID: 0x10dd1014, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Draw Style Attribute Element contains information defining various aspects of the graphics state/style that should be used for rendering associated geometry. JT format LSG traversal semantics state that draw style attributes accumulate down the LSG by replacement.

The Field Inhibit flag (see 7.2.1.1.2.1.1 Base Attribute Data) bit assignments for the Draw Style Attribute Element data fields, are as follows:

Field Inhibit Flag Bit	Data Field(s) Bit Applies To
0	Two Sided Lighting Flag
1	Back-face Culling Flag
2	Outlined Polygons Flag
3	Lighting Enabled Flag
4	Flat Shading Flag
5	Separate Specular Flag

Figure 51: Draw Style Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH only valid value for Draw Style Attribute Element.

U8 : Data Flags

Data Flags is a collection of flags. The flags are combined using the binary OR operator and store various state settings for Draw Style Attribute Elements. All undocumented bits are reserved.

0x01	Back-face Culling Flag. Indicates if back-facing polygons should be discarded (culled). = 0 ± Back-facing polygons not culled. = 1 ± Back-facing polygons culled.
0x02	Two Sided Lighting Flag. Indicates if two sided lighting should be enabled to insure that polygons are illuminated on both sides. = 0 ± Disable two sided lighting. = 1 ± Enable two sided lighting.
0x04	Outlined Polygons Flag. Indicates if polygons should be draw as ³ZLUHIUDPHs´ L H QRW ILOOHG = 0 ± Polygons drawn as filled. = 1 ± 2Q0\ SR0\JRQ¶V RXWOLQH GUDZQ
0x08	Lighting Enabled Flag. Indicates if lighting should be enabled. If lighting disabled, then renderer should perform no calculations concerning normals, light sources, material properties, etc. = 0 ± Disable lighting. = 1 ± Enable lighting.
0x10	Flat Shading Flag. Indicates if the geometry should be rendered with single color (flat shading) or with many different color (smooth/Gouraud) shading. = 0 ± Disable flat shading (i.e. use smooth/Gouraud shading). = 1 ± Enable flat shading.
0x20	Separate Specular Flag. Indicates if the application of the specular color should be delayed until after texturing. If

	no texture mapping then this flag setting is irrelevant. = 0 ± Apply specular color contribution before texture mapping. = 1 ± Apply specular color contribution after texture mapping.

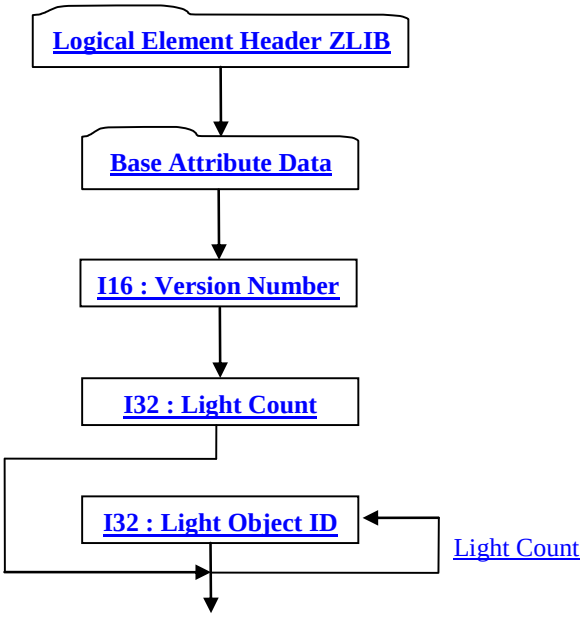
7.2.1.1.2.5 Light Set Attribute Element

Object Type ID: 0x10dd1096, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Light Set Attribute Element holds an unordered list of Lights. JT format LSG traversal semantics state that light set attributes accumulate down the LSG through addition of lights to an attribute list.

Light Set Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 52: Light Set Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQO\ YDOLG YDOXH IRU Light Set Attribute Element.

I32 : Light Count

Light Count specifies the number of lights in the Light Set.

I32 : Light Object ID

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

7.2.1.1.2.6 Infinite Light Attribute Element

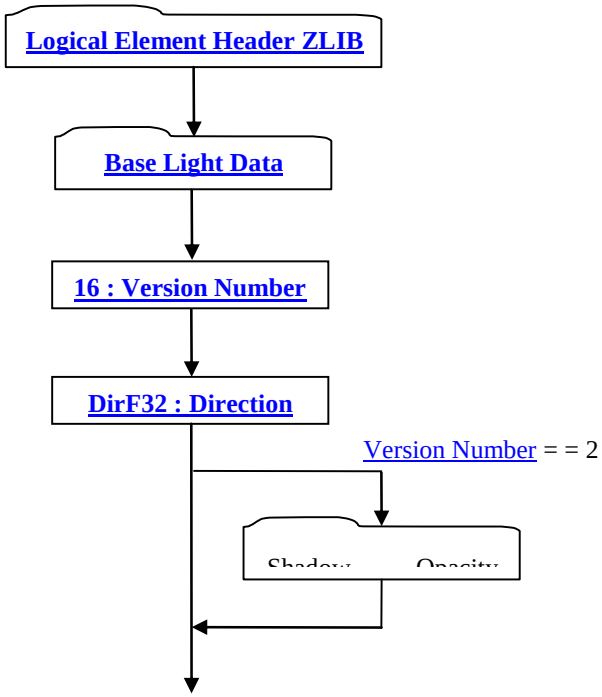
Object Type ID: 0x10dd1028, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Infinite Light Attribute Element specifies a light source emitting unattenuated light in a single direction from every point on an infinite plane. The infinite location indicates that the rays of light can be considered parallel by the time they reach an object.

JT format LSG traversal semantics state that infinite light attributes accumulate down the LSG through addition of lights to an attribute list.

Infinite Light Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 53: Infinite Light Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Light Data can be found in 7.2.1.1.2.6.1Base Light Data.

Complete description for Shadow Parameters can be found in 7.2.1.1.2.6.2 Shadow Parameters.

16 : Version Number

Version Number is the version identifier for this element. The value of this Version Number indicates the format of data fields to follow.

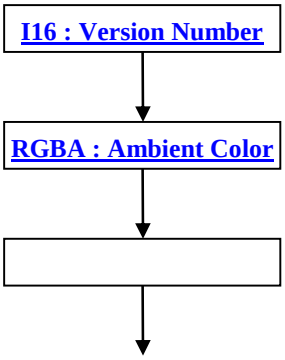
= 1	Version-1 Format
= 2	Version-2 Format

DirF32 : Direction

Direction specifies the direction the light is pointing in.

7.2.1.1.2.6.1 Base Light Data

Figure 54: Base Light Data collection



I16 : Version Number

9HUVLRQ QXPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQO\ YDOLG YDOXH IRU %DVH Light Data.

RGBA : Ambient Color

Ambient Color specifies the ambient red, green, blue, alpha color values of the light.

RGBA : Diffuse Color

Diffuse Color specifies the diffuse red, green, blue, alpha color values of the light.

RGBA : Specular Color

Specular Color specifies the specular red, green, blue, alpha color values of the light.

F32 : Brightness

%ULJKWQHVV VSHFLILHV WKH /LJKW EULJKWQHVV 7KH %ULJKWQHVV YDOXH PXVW EH JUHDWHU WKDQ RU HTXDO WR ³- ´

I32 : Coord System

Coord System specifies the coordinate space in which Light source is defined. Valid values include the following:

= 1	Viewpoint Coordinate System. Light source is to move together with the viewpoint
= 2	Model Coordinate System. Light source is affected by whatever model transforms that are current when the light source is encountered in LSG.
= 3	World Coordinate system. Light source is not affected by model transforms in the LSG.

U8 : Shadow Caster Flag

Shadow Caster Flag is a flag that indicates whether the light is a shadow caster or not.

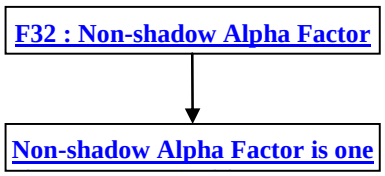
= 0	Light source is not a shadow caster.
= 1	Light source is a shadow caster.

F32 : Shadow Opacity

Shadow Opacity specifies the shadow opacity factor on Light source. Value must be within range [0.0, 1.0] inclusive. Shadow Opacity is intended to convey how dark a shadow cast by this light source are to be rendered. A value of 1.0 means that no light from this light source reaches a shadowed surface, resulting in a black shadow.

7.2.1.1.2.6.2 Shadow Parameters

Figure 55: Shadow Parameters data collection



F32 : Non-shadow Alpha Factor

Non-shadow Alpha Factor is one of a matched pair of fields intended to govern how a shadowing light source (one whose Shadow Caster Flag is set) casts "alpha light" into areas that it directly illuminates (i.e. are not in shadow). Those fragments directly lit by this light source will have their alpha values scaled by Non-shadow Alpha Factor. Non-shadow Alpha Factor value must lie on the range [0.0, 1.0] inclusive.

This field can be used to create "drop shadows" by setting its value to 0. The effect being that all geometry illuminated by the light source will be "burned away," leaving behind only those parts lying in shadow. Naturally, implementing this intended behavior implies extensive viewer support.

F32 : Shadow Alpha Factor

Shadow Alpha Factor is one of a matched pair of fields intended to govern how a shadowing light source (one whose Shadow Caster Flag is set) casts "alpha light" into areas that it does not illuminate (i.e. are in shadow). Those fragments in shadow from this light source will have their alpha values scaled by Shadow Alpha Factor. Shadow Alpha Factor value must lie on the range [0.0, 1.0] inclusive.

This field has the opposite effect of Non-shadow Alpha Factor. If set to a value of 0, for example, it will cause all geometry shadowed from the light source to be burned away, leaving behind only those parts directly illuminated by the light source. Naturally, implementing this intended behavior implies extensive viewer support.

7.2.1.1.2.7 Point Light Attribute Element

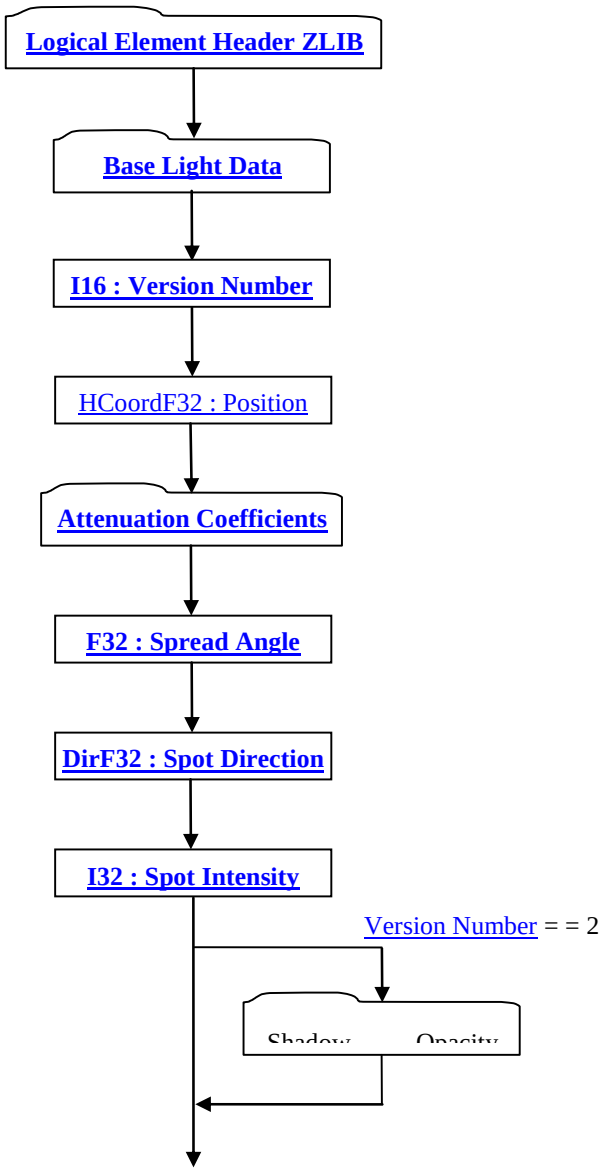
Object Type ID: 0x10dd1045, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Point Light Attribute Element specifies a light source emitting light from a specified position, along a specified direction, and with a specified spread angle

JT format LSG traversal semantics state that point light attributes accumulate down the LSG through addition of lights to an attribute list.

Point Light Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 56: Point Light Attribute ElementPoint Light Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Light Data can be found in 7.2.1.1.2.6.1 Base Light Data.

Complete description for Attenuation Coefficients can be found in 7.2.1.1.2.7.1Attenuation Coefficients.

Complete description for Shadow Parameters can be found in 7.2.1.1.2.6.2 Shadow Parameters.

I16 : Version Number

Version Number is the version identifier for this element. The value of this Version Number indicates the format of data fields to follow.

= 1	Version-1 Format
= 2	Version-2 Format

HCoordF32 : Position

Position specifies the light position in homogeneous coordinates.

F32 : Spread Angle

Spread Angle, as shown in Figure 57 below, specifies in degrees the half angle of the light cone. Valid Spread Angle values are clamped and interpreted as follows:

angle = = 180.0	Simple point light
0.0 >= angle <= 90.0	Spot Light

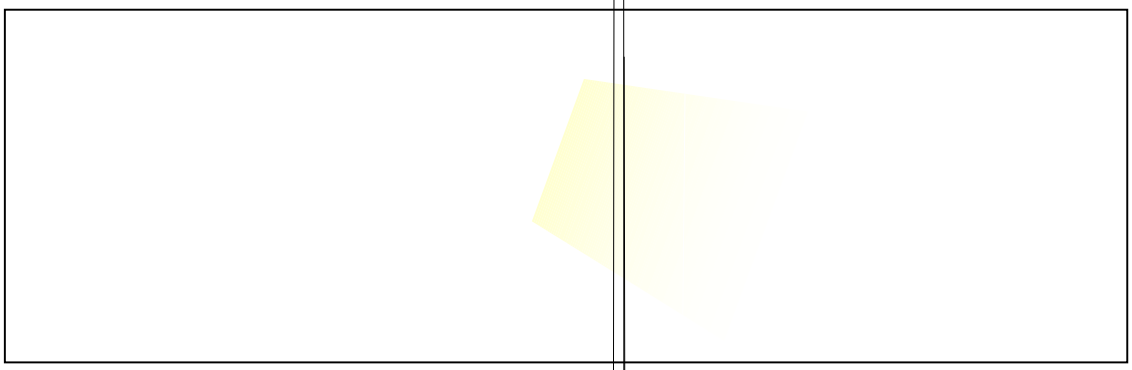


Figure 57: Spread Angle value with respect to the light cone

DirF32 : Spot Direction

Spot Direction specifies the direction the spot light is pointing in.

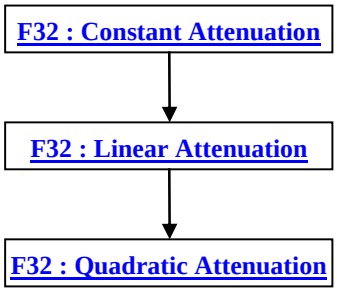
I32 : Spot Intensity

Spot Intensity specLILHV WKH LQWHQVLW\ GLVWULEXWLRQ RI WKH OLJKW ZLWKLQ WKH VSRW OLJKW FRQH 6SRW ,QWHQVLW\ LV UHD00\ D ³VSRW H[SRQHQW´ LQ D OLJKWLQJ HTXDWLRQ DQG LQGLFDWHV KRZ IRFXVHG WKH OLJKW LV DW WKH FHQWHU 7KH ODUJHU WKH YDOXH WKH PRUH IRFXVHd the light source. Only non-negative Spot intensity values are valid.

7.2.1.1.2.7.1 Attenuation Coefficients

Attenuation Coefficients data collection contains the coefficients for how light intensity decreases with distance.

Figure 58: Attenuation Coefficients data collection



F32 : Constant Attenuation

Constant Attenuation specifies the constant coefficient for how light intensity decreases with distance. Value must be greater WKDQ RU HTXDO WR ³ ´

F32 : Linear Attenuation

Linear Attenuation specifies the linear coefficient for how light intensity decreases with distance. Value must be greater than RU HTXDO WR ³ ´

F32 : Quadratic Attenuation

Quadratic Attenuation specifies the quadratic coefficient for how light intensity decreases with distance. Value must be JUHDWHU WKDQ RU HTXDO WR ³ ´

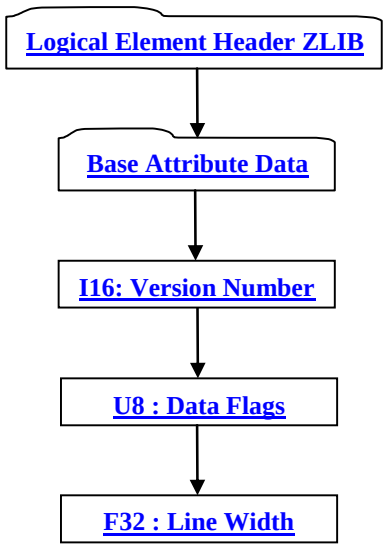
7.2.1.1.2.8 Linestyle Attribute Element

Object Type ID: 0x10dd10c4, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Linestyle Attribute Element contains information defining the graphical properties to be used for rendering polylines. JT format LSG traversal semantics state that Linestyle attributes accumulate down the LSG by replacement.

Linestyle Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 59: Linestyle Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16: Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQO\ YDOLG YDOXH IRU Linestyle Attribute Element.

U8 : Data Flags

Data Flags is a collection of flags and line type data. The flags and line type data are combined using the binary OR operator and store various polyline rendering attributes. All undocumented bits are reserved.

0x0F	Line Type (stored in bits 0 ± 3 or in binary notation 00001111) Line type specifies the polyline rendering stipple-pattern. = 0 - Solid = 1 ± Dash = 2 ± Dot = 3 ± Dash_Dot = 4 ± Dash_Dot_Dot = 5 ± Long_Dash = 6 ± Center_Dash = 7 ± Center_Dash_Dash
0x10	Antialiasing Flag (stored in bit 4 or in binary notation 00010000) Indicates if antialiasing should be applied as part of rendering polylines. = 0 ± Antialiasing disabled. = 1 ± Antialiasing enabled.

F32 : Line Width

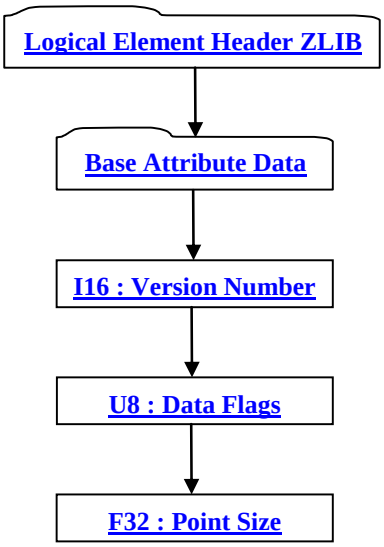
Line Width specifies the width in pixels that should be used for rendering polylines. The value of this field must be greater than 0.0.

7.2.1.1.2.9 Pointstyle Attribute Element

Object Type ID: 0x8d57c010, 0xe5cb, 0x11d4, 0x84, 0xe, 0x00, 0xa0, 0xd2, 0x18, 0x2f, 0x9d

Pointstyle Attribute Eleme213947 Tc 7.57305 0 Td (in)Tj 0.0s

Figure 60: Pointstyle Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16 : Version Number

Version Number is the version identifier for WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQO\ YDOLG YDOXH for Pointstyle Attribute Element.

U8 : Data Flags

Data Flags is a collection of flags and point type data. The flags and point type data are combined using the binary OR operator and store various point rendering attributes. All undocumented bits are reserved.

0x0F	Point Type (stored in bits 0 ± 3 or in binary notation 00001111) These bits are reserved for future expansion of the format to support Point Types.
0x10	Antialiasing Flag (stored in bit 4 or in binary notation 00010000) Indicates if antialiasing should be applied as part of rendering points. = 0 ± Antialiasing disabled. = 1 ± Antialiasing enabled.

F32 : Point Size

Point Size specifies the size in pixels that should be used for rendering points. The value must be greater than 0.0.

7.2.1.1.2.10 Geometric Transform Attribute Element

Object Type ID: 0x10dd1083, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Geometric Transform Attribute Element contains a 4x4 homogeneous transformation matrix that positions the associated /6* QRGH¶V FRRUGLQDWH V\VWHP UHODWLYH WR LWV SDUHQW /6* QRGH JT format LSG traversal semantics state that geometric transform attributes accumulate down the LSG through matrix multiplication as follows:

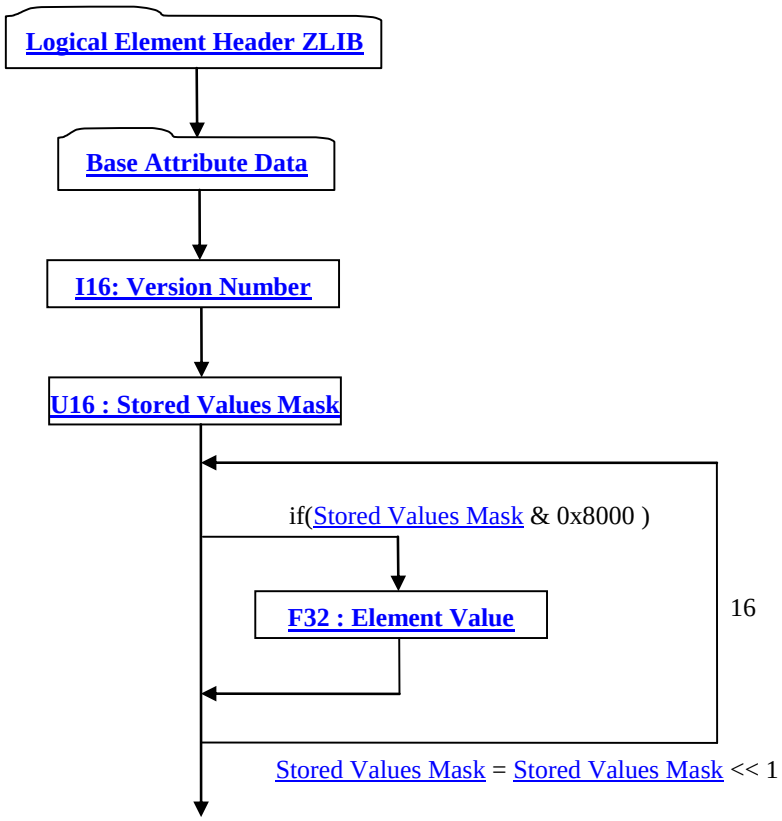
S¶ S$0

Where *p* is a point of the model, S¶ is the transformed point, *M* is the current modeling transformation matrix inherited from ancestor LSG nodes and previous Geometric Transform Attribute Element, and *A* is the transformation matrix of this Geometric Transform Attribute Element. The matrix is allowed to contain translation, rotation, and uniform- and non-

uniform scaling factors, including negative scales. It is not allowed to contain shearing or projective components, or scaling factors of zero (which would make the matrix singular).

Geometric Transform Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 61: Geometric Transform Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16: Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV QRGH 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDOXH IRU Geometric Transform Attribute Element.

U16 : Stored Values Mask

Stored Values mask is a 16-bit mask where each bit is a flag indicating whether the corresponding element in the matrix is different from the identity matrix. Only elements which are different from the identity matrix are actually stored. The bits are assigned to matrix elements as follows:

Bit15 Bit14 Bit13 Bit12

Bit11 Bit10 Bit9 Bit8

Bit7 Bit6 Bit5 Bit4

Bit3 Bit2 Bit1 Bit0

The individual bit-flag values are interpreted as follows:

= 0	Value not stored (matrix value same as corresponding element in identity matrix)
= 1	Value stored

F32 : Element Value

Element Value specifies a particular matrix element value.

7.2.1.1.2.11 Shader Effects Attribute Element

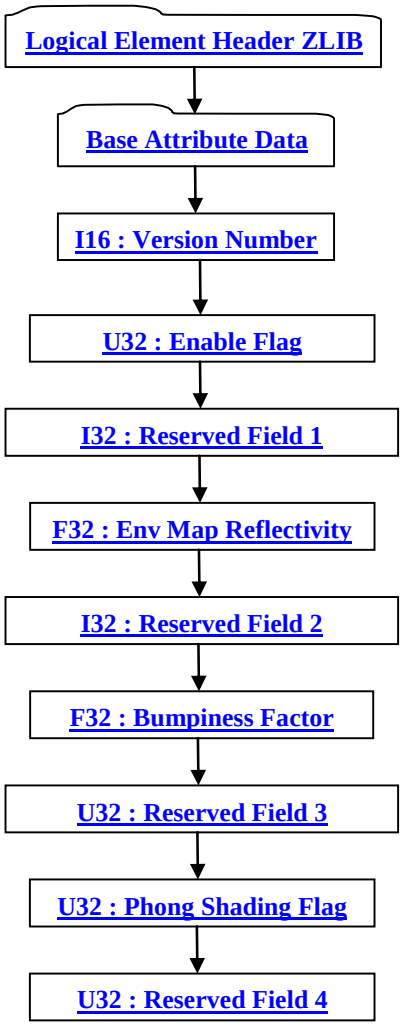
Object Type ID: 0xaa1b831d, 0x6e47, 0x4fee, 0xa8, 0x65, 0xcd, 0x7e, 0x1f, 0x2f, 0x39, 0xdb

Shader Effects Attribute Element FRQWDLQV LQIRUPDWLRQ VSHFLI\LQJ ³KLJK-0HYH0´ VKDGHU IXQFWLRQDOLW\ (e.g. Phong shading, bump mapping, etc.) that should be used for rendering the geometry this attribute element is associated with.

JT format LSG traversal semantics state that shader effects attributes accumulate down the LSG by replacement.

Shader Effects Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 62: Shader Effects Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDOXH

U32 : Enable Flag

Enable Flag specifies whether this Shader Effects Attribute is enabled. Valid values include the following:

= 0	Shader Effects Attribute disabled
= 1	Shader Effects Attribute enabled

I32 : Reserved Field 1

Reserved Field 1 is a data field reserved for future JT format expansion

F32 : Env Map Reflectivity

Env Map Reflectivity specifies the fraction of the environment to be reflected (1 minus this fraction will show through form the underlying texture channel). Valid value must be in the range [0:1] inclusive.

I32 : Reserved Field 2

Reserved Field 2 is a data field reserved for future JT format expansion

F32 : Bumpiness Factor

Bumpiness Factor specifies the GHJUHH RI ³EXPSLQHVV´ RU WKH UHODWLYH ³KHLJKW´ RI WKH EXPS PDS Larger values make the bumps appear deep and more severe. Negative values invert the sense of the bump map, making the surface appear engraved, rather than embossed. This value only has an effect with tangent space bump maps.; it has no effect on the appearance of object space bump maps.

U32 : Reserved Field 3

Reserved Field 3 is a data field reserved for future JT format expansion

U32 : Phong Shading Flag

Phong Shading Flag specifies whether Phong Shading (i.e. per fragment lighting) is enabled. Valid values include the following:

= 0	Phong Shading disabled
= 1	Phong Shading enabled

U32 : Reserved Field 4

Reserved Field 4 is a data field reserved for future JT format expansion

7.2.1.1.2.12 Vertex Shader Attribute Element

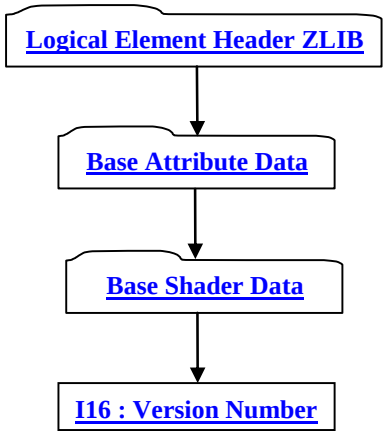
Object Type ID: 0x2798bcad, 0xe409, 0x47ad, 0xbd, 0x46, 0xb, 0x37, 0x1f, 0xd7, 0x5d, 0x61

Vertex Shader Attribute Element defines a per-vertex shader program in the GLSL shading language. A complete description of the GLSL shading language can be found in references listed within the 3 References and Additional Information section of this document.

JT format LSG traversal semantics state that vertex shader attributes accumulate down the LSG by replacement.

In general, a shader program is used to replace a portion of the otherwise fixed-function graphics pipeline with some user-defined functionality. Specifically a Vertex Shader program is a small user defined program to be run for each vertex that is sent to the GPU for processing. A Vertex shader can alter vertex positions and normals, generate texture coordinates, perform Gouraud per-vertex lighting, etc.

Figure 63: Vertex Shader Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW0 \ WKH RQ0 \ YD0LG YD0XH

7.2.1.1.2.13 Fragment Shader Attribute Element

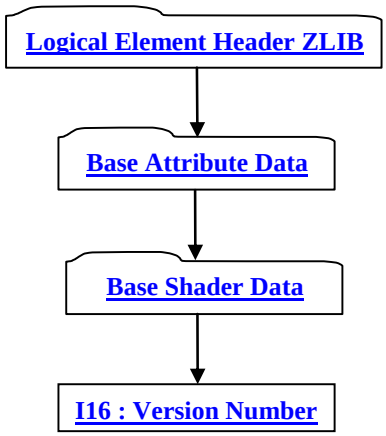
Object Type ID: 0xad8dccc2, 0x7a80, 0x456d, 0xb0, 0xd5, 0xdd, 0x3a, 0xb, 0x8d, 0x21, 0xe7

Fragment Shader Attribute Element defines a per-fragment shader program in the GLSL shading language. A complete description of the GLSL shading language can be found in references listed within the 3 References and Additional Information section of this document.

JT format LSG traversal semantics state that fragment shader attributes accumulate down the LSG by replacement; with the exception that if WKH QHZ IUDJPHQW VKDGHU DWWULEXWH¶V VKDGHU 0DQXDJH LV QRW WKH VDPH DV FXUUHQW IUDJPHQW VKDGHU DWWULEXWH¶V shader language, then new fragment shader attribute is simply ignored.

In general, a shader program is used to replace a portion of the otherwise fixed-function graphics pipeline with some user-defined functionality. Specifically a Fragment Shader program is a small user defined program to be run for each fragment JHQHUDWHG E\ WKH *38 KDUGZDUH¶V VFDQ-conversion logic. A fragment is a "proto-pixel" generated by triangle scan-conversion, but not let laid down into the frame buffer, where it will become an actual pixel. A Fragment Shader can support sophisticated effects like Phong shading, shadow mapping, bump mapping, reflection mapping, etc.

Figure 64: Fragment Shader Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

Complete description for Base Shader Data can be found in 7.2.1.1.2.1.2 Base Shader Data.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV H0HPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW0\ WKH RQ\ YD0LG YD0XH

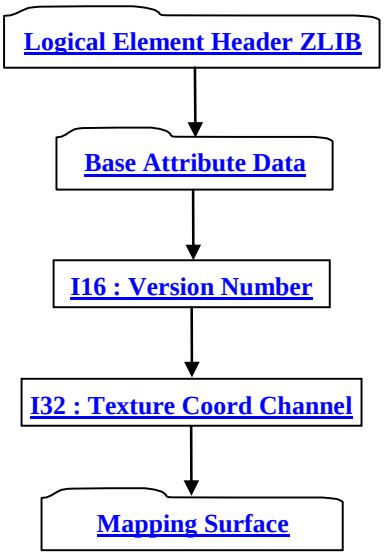
7.2.1.1.2.14 Texture Coordinate Generator Attribute Element

Object Type ID: 0xaa1b831d, 0x6e47, 0x4fee, 0xa8, 0x65, 0xcd, 0x7e, 0x1f, 0x2f, 0x39, 0xdc

Texture Coordinate Generator Attribute Element defines texture coordinate generation for texture mapping. Multiple texture FRRUGLQDWH JHQHUDWLRQ DW D JLYHQ QRGH LV VXSSRUWHG E\ ZD\ RI WKH ³WH[WXUH FRRUGLQDWH FKDQQH0´ FRQFHSW -7 IRUPDW /6* traversal semantics state that Texture Coordinate Generator attributes accumulate down the LSG by replacement on a per-channel basis.

Texture Coordinate Generator Attribute Element does not have any Field Inhibit flag (see 7.2.1.1.2.1.1Base Attribute Data) bit assignments.

Figure 65: Texture Coordinate Generator Attribute Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

Complete description for Base Attribute Data can be found in 7.2.1.1.2.1.1Base Attribute Data.

Complete description for Mapping Surface can be found in 7.2.1.1.2.14.1Mapping Surface.

I16 : Version Number

Version Number is the version identifier for thi V H0HPHQW 9HUVLRQ QXPEHU ³ [´LV FXUUHQW\ WKH RQ\ YD0LG YD0XH

I32 : Texture Coord Channel

Tex Coord Channel specifies the channel number for texture coordinate generation. Value must be within range [0, 2147483647] inclusive. This number is intended to match up with the I32 : Tex Coord Channel field on Texture Image Attribute Element in order to associate a specific Texture Coordinate Generator with a Specific Texture Image.

7.2.1.1.2.14.1 Mapping Surface

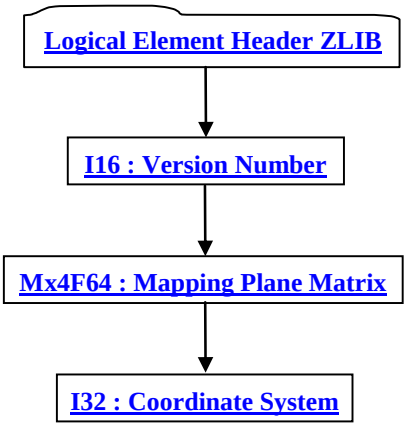
Mapping Surface defines the mapping surface for texture coordinate generation. Four kinds of mapping surfaces, Mapping Plane Element, Mapping Cylinder Element, Mapping Sphere Element, and Mapping TriPlanar Element, are defined to support texture coordinate generation.

7.2.1.1.2.14.1.1 Mapping Plane Element

Object Type ID: 0xa3cfb921, 0xbdeb, 0x48d7, 0xb3, 0x96, 0x8b, 0x8d, 0xe, 0xf4, 0x85, 0xa0

Mapping Plane Element defines the mapping plane for texture coordinate generation.

Figure 66: Mapping Plane Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQly valid value.

Mx4F64 : Mapping Plane Matrix

Mx4F64 : Mapping Plane Matrix specifies the transformation matrix and mapping parameters for the mapping plane. The transformation matrix defines the mapping coordinate system transformed from I32 : Coordinate System. The mapping parameters specifies the width and height of the mapping plane. The mapping plane is defined in the + xy-plane of the mapping coordinate system. In the mapping process, the geometry vertex coordinates in Model Coordinate System are transformed to the mapping coordinate system at first, and then the transformed vertex coordinates are mapped to texture coordinates as following:

s-coordinate = x-coordinate of the transformed vertex / the width of the mapping plane
t-coordinate = y-coordinate of the transformed vertex / the height of the mapping plane

I32 : Coordinate System

Coordinate system specifies the coordinate space in which mapping plane is defined. Valid values include the following

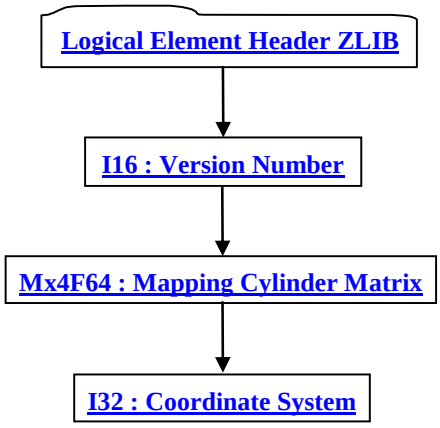
= 0	Undefined Coordinate System.
= 1	Viewpoint Coordinate System. Mapping plane is to move together with the viewpoint.
= 2	Model Coordinate System. Mapping plane is affected by whatever model transforms that are current when the mapping plane is encountered in LSG.
= 3	World Coordinate system. Mapping plane is not affected by model transforms in the LSG.

7.2.1.1.2.14.1.2 Mapping Cylinder Element

Object Type ID: 0x3e70739d, 0x8cb0, 0x41ef, 0x84, 0x5c, 0xa1, 0x98, 0xd4, 0x0, 0x3b, 0x3f

Mapping Cylinder Element defines the mapping cylinder for texture coordinate generation.

Figure 67: Mapping Cylinder Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQly valid value.

Mx4F64 : Mapping Cylinder Matrix

Mx4F64 : Mapping Cylinder Matrix specifies the transformation matrix and mapping parameters for the mapping cylinder. The transformation matrix defines the mapping coordinate system transformed from I32 : Coordinate System. The mapping parameters specifies the horizontal sweep angle and heigKW RI WKH PDSSLQJ F\OLQGHU 7KH PDSSLQJ F\OLQGHU¶V D[LV LV SDUDOOHO to the z-axis of the mapping coordinate system, and the horizontal sweep angle starts from the +x-axis in a counter clockwise direction. In the mapping process, the geometry vertex coordinates in Model Coordinate System are transformed to the mapping coordinate system at first, and then the transformed vertex coordinates are mapped to texture coordinates as following:

s-coordinate = the horizontal sweep angle of the vertex / the horizontal sweep angle of the mapping cylinder
t-coordinate = the z-coordinate of the vertex / height of the mapping cylinder

Mapping Cylinder Element implements the strategy to handle texture coordinates who cross the seam of the texture in the mapping process.

I32 : Coordinate System

Coordinate system specifies the coordinate space in which mapping cylinder is defined. Valid values include the following

= 0	Undefined Coordinate System.
= 1	Viewpoint Coordinate System. Mapping cylinder is to move together with the viewpoint.
= 2	Model Coordinate System. Mapping cylinder is affected by whatever model transforms that are current when the mapping cylinder is encountered in LSG.
= 3	World Coordinate system. Mapping cylinder is not affected by model transforms in the LSG.

7.2.1.1.2.14.1.3 Mapping Sphere Element

Object Type ID: 0x72475fd1, 0x2823, 0x4219, 0xa0, 0x6c, 0xd9, 0xe6, 0xe3, 0x9a, 0x45, 0xc1

Mapping Sphere Element defines the mapping sphere for texture coordinate generation.

Figure 68: Mapping Sphere Element data collection

Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

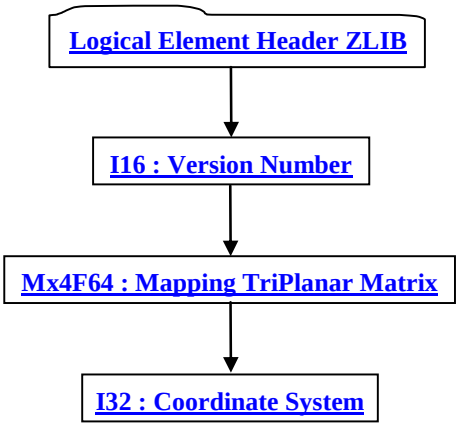
I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQW\ WKH RQ\ YDOLG YDOXH

Mx4F64 : Mapping Sphere Matrix

Mx4F64 : Mapping Sphere Matrix specifies the transformation matrix and mapping parameters of the mapping sphere. The transformation matrix defines the mapping coordinate system transformed from I32 : sfyfr

Figure 69: Mapping TriPlanar Element data collection



Complete description for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

I16 : Version Number

9HUVLRQ 1XPEHU LV WKH YHUVLRQ LGHQWLILHU IRU WKLV HOHPHQW 9HUVLRQ QXPEHU ³ [´ LV FXUUHQWO\ WKH RQly valid value.

Mx4F64 : Mapping TriPlanar Matrix

Mx4F64 : Mapping TriPlanar Matrix specifies the transformation matrix and mapping parameter for the mapping triplanar. The transformation matrix defines the mapping coordinate system transformed from I32 : Coordinate System. The mapping parameter specifies the planar length of the triplanar. The left bottom corner of the triplanar is located at the origin of the mapping coordinate system, and the three planes are in the + xy-plane, + yz-plane, and + xz-plane respectively. In the mapping process, the geometry vertex coordinates in Model Coordinate System are transformed to the mapping coordinate system at first, and then the transformed vertex coordinates are projected to the corresponding plane based on the maximum component of its normals, and at last the projected vertex coordinates are mapped to texture coordinates as following:

s-coordinate = the first-coordinate of the projected vertex / the planar length of the triplanar
t-coordinate = the second-coordinate of the projected vertex / the planar length of the triplanar

I32 : Coordinate System

Coordinate system specifies the coordinate space in which mapping triplanar surface is defined. Valid values include the following

= 0	Undefined Coordinate System.
= 1	Viewpoint Coordinate System. Mapping triplanar surface is to move together with the viewpoint.
= 2	Model Coordinate System. Mapping triplanar surface is affected by whatever model transforms that are current when the mapping triplanar surface is encountered in LSG.
= 3	World Coordinate system. Mapping triplanar surface is not affected by model transforms in the LSG.

7.2.1.2 Property Atom Elements

Property Atom Elements are meta-data objects associated with nodes or Attributes. Property Atom Elements are not nodes or attributes themselves, but can be associated with any node or Attribute to maintain arbitrary application- or enterprise information (meta-data) pertaining to that node or Attribute. Each Node Element or Attribute Element in an LSG may hold zero or more Property Atom Elements and this relationship information is stored within 7.2.1.3 Property Table section of a JT file.