



JT File Format Reference

Version 10.0

Rev-C

© 2017 Siemens Product Lifecycle Management Software Inc. All rights reserved. Siemens, JT, and Parasolid are registered trademarks or trademarks of Siemens Product Lifecycle Management Software Inc. in the United States and/or other countries. OpenGL is a registered trademark or trademark of SGI. All other trademarks are the property of their respective owners.

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 10.0 document was written by the JT Format and DirectModel team's developers themselves: Michael Carter, Jianbing Huang, Sashank Ganti, J

Contents

Page

1	Siemens JT Data Format Reference Intellectual Property License Terms	2
2	Scope	3
2.1	What's New in This Revision	4
3	Terms, definitions and abbreviated terms	5
3.1	Terms and definitions	5
3.2	Abbreviated terms	8
4	Notational conventions.....	10
4.1	Diagrams and field descriptions	10
4.2	Data Types	13
4.3	Empty field	17
5	File Format	18
5.1	File Structure	18
5.1.1	File Header	18
5.1.2	TOC Segment	20
5.1.3	Data Segment.....	21
5.2	Data Segments.....	26
6	LSG Segment	28
6.1	Graph Elements	28
6.1.1	Node Elements.....	29
6.1.2	Attribute Elements.....	49
6.2	Property Atom Elements.....	83
6.2.1	Base Property Atom Element	83
6.2.2	String Property Atom Element	84
6.2.3	Integer Property Atom Element	84
6.2.4	Floating Point Property Atom Element	85
6.2.5	JT Object Reference Property Atom Element.....	86
6.2.6	Date Property Atom Element.....	86
6.2.7	Late Loaded Property Atom Element	88
6.2.8	Vector4f Property Atom Element	89
6.3	Property Table	90
6.3.1	Element Property Table	91
7	Shape LOD Segment	92
7.1	Shape LOD Element	92
7.1.1	Tri-Strip Set Shape LOD Element	92
7.1.2	Polyline Set Shape LOD Element.....	93
7.1.3	Point Set Shape LOD Element	93
7.1.4	Polygon Set LOD Element	94
7.1.5	Null Shape LOD Element	107
7.2	Primitive Set Shape Element	107
7.2.1	Lossless Compressed Primitive Set Data.....	109
7.2.2	Lossy Quantized Primitive Set Data	111
8	Precise Geometry Segment.....	115
8.1	JT B-Rep Element (deprecated)	115
8.2	XT B-Rep Element	115
8.3	JT ULP Segment	115
9	JT LWPA Segment.....	115
9.1	JT LWPA Element.....	116
9.1.1	Analytic Surface Geometry.....	117
10	Wireframe Segment.....	120
10.1	Wireframe Rep Element	120
10.1.1	Wireframe MCS Curves Geometric Data	122
10.1.2	Wireframe Rep CAD Tag Data	122

11	Meta Data Segment	123
11.1	Property Proxy Meta Data Element.....	123
11.1.1	Date Property Value	125
11.2	PMI Manager Meta Data Element	126
11.2.1	PMI Design Group Entities	128
11.2.2	PMI Associations	130
11.2.3	PMI User Attributes	133
11.2.4	PMI String Table	133
11.2.5	PMI Model Views.....	134
11.2.6	Generic PMI Entities.....	139
11.2.7	PMI CAD Tag Data	149
11.2.8	PMI Polygon Data	150
11.2.9	PMI Properties	153
11.2.10	PMI Model View Sort Orders.....	154
12	Data Compression and Encoding	154
12.1	Common Compression Data Collection Formats	155
12.1.1	Int32 Compressed Data Packet.....	155
12.1.2	Int64 Compressed Data Packet.....	161
12.1.3	Compressed Vertex Coordinate Array	164
12.1.4	Compressed Vertex Normal Array	165
12.1.5	Compressed Vertex Texture Coordinate Array	167
12.1.6	Compressed Vertex Colour Array.....	168
12.1.7	Compressed Vertex Flag Array.....	170
12.1.8	Compressed Auxiliary Fields Array.....	170
12.1.9	Point Quantizer Data	174
12.1.10	Texture Quantizer Data	175
12.1.11	Colour Quantizer Data.....	175
12.1.12	Uniform Quantizer Data	177
12.1.13	Compressed Entity List for Non-Trivial Knot Vector	177
12.1.14	Compressed Control Point Weights Data	180
12.1.15	Compressed Curve Data.....	181
12.1.16	Compressed CAD Tag Data.....	185
12.2	Encoding Algorithms	186
12.2.1	Uniform Data Quantization	186
12.2.2	Bitlength CODEC	186
12.2.3	Arithmetic CODEC	188
12.2.4	Deering Normal CODEC.....	193
12.3	LZMA compression	195
13	Common Data Conventions and Constructs	196
13.1	Late-Loading Data	196
13.2	TOC Segment Location.....	196
13.3	Bit Fields.....	196
13.4	Empty Field	196
13.5	Local version numbers	196
13.5.1	Version numbers	196
13.6	Hash Value	197
13.7	Scene graph construction	197
13.8	Metadata Conventions	198
13.8.1	Property Key Naming Conventions	198
13.8.2	PMI Properties	199
13.8.3	CAD Properties	199
13.8.4	Tessellation Properties	201
13.8.5	Miscellaneous Properties	202
13.8.6	The SUBNODE property and Reference Sets	203
13.9	LSG Attribute Accumulation Semantics	207
13.10	LSG Part Structure	208
13.11	Range LOD Node Alternative Rep Selection	208
13.12	B-Rep Face Group Associations	208
13.13	Watermark Image.....	209

13.14	State Flags	210
	Annex A Object Type Identifiers	211
	Annex B Coding Algorithms – An Implementation	215
	Annex C Hashing – An Implementation	239
	Annex D Polygon Mesh Topology Coder	242
	Annex E (deprecated) JT B-Rep Segment	261
E.1.1	Topological Entity Counts	263
E.1.2	Geometric Entity Counts	264
E.1.3	Topology Data.....	265
E.1.4	Geometric Data	274
E.1.5	Topological Entity Tag Counters	283
E.1.6	B-Rep CAD Tag Data	284
	Annex F XT B- Rep data segment.....	285
F.1	XT B-Rep Element	285
F.1.1	XT B-Rep Data.....	286
F.1.2	Integer Attribute Data.....	286
F.1.3	MultiXT B-Rep Segment.....	287
F.2	XT B-Rep Data Segment Description	289
F.2.1	Logical Layout	289
F.2.2	Physical Layout	293
F.2.3	Model Structure	294
F.2.4	Schema Definition	300
F.2.5	Node Types	357
F.2.6	Node Classes	358
F.2.7	System Attribute Definitions	359
F.3	XT Moniker Attributes	365
F.3.1	Moniker IDs	366
	Annex G JT ULP Segment	370
G.1	JT ULP Element	370
G.1.1	Topology Data.....	372
G.1.2	Geometric Data	389
G.1.3	Material Attribute Element Properties	413
G.1.4	Information Recovery.....	414
	Annex H (deprecated) PMI Data Segment.....	419
	Annex I Procedural Geometry – Evaluation and Approximation.....	421
I.1	Introduction & Scope	421
I.2	Notation	421
I.3	Pseudocode	421
I.4	Intersection Curve	421
I.4.1	Intersection Curve Basics	421
I.4.2	Populating Chart Points.....	423
I.4.3	Computing a Point & Tangent on an Intersection Curve	429
I.4.4	Approximating an Intersection Curve	431
I.5	Rolling-Ball Blend Surface	440
I.5.1	Computing a Point on a Blend Surface.....	440
I.5.2	Approximating a Blend Surface	445
I.6	Blend Surface Questions and Answers	450
I.7	Annex Bibliography.....	453
	Annex J PMI Properties	454
	Bibliography.....	562

Figures

Figure 1 — rectangle box diagram	10
Figure 2 — folder diagram	10
Figure 3 — rectangle box with lines at left and right sides diagram	11
Figure 4 — rectangle box with clipped right side corners	11
Figure 5 — compressed data packet diagram.....	11
Figure 6 — data type : field name diagram	12
Figure 7 — data filed dependency example.....	13
Figure 8 — loop construct example	13
Figure 9 — loop construct with iterations example	13
Figure 10 — JT File Structure.....	18
Figure 11 — File Header data collection	19
Figure 12 — TOC Segment data collection	20
Figure 13 — TOC Entry data collection.....	21
Figure 14 — Data Segment data collection.....	22
Figure 15 — Segment Header data collection	22
Figure 16 — Data collection	24
Figure 17 — Logical Element Header data collection.....	24
Figure 18 — Element Header data collection	24
Figure 19 — Logical Element Header Compressed data collection.....	25
Figure 20 — LSG Segment data collection	28
Figure 21 — Base Node Element data collection	29
Figure 22 — Base Node Data collection.....	29
Figure 23 — Partition Node Element data collection	31
Figure 24 — Vertex Count Range data collection	32
Figure 25 — Group Node Element data collection.....	33
Figure 26 — Group Node Data collection	34
Figure 27 — Instance Node Element data collection	35
Figure 28 — Part Node Element data collection.....	35
Figure 29 — Meta Data Node Element data collection	36
Figure 30 — Meta Data Node Data collection	36
Figure 31 — LOD Node Element data collection	37
Figure 32 — LOD Node Data collection.....	37
Figure 33 — Range LOD Node Element data collection	38
Figure 34 — Switch Node Element data collection	39
Figure 35 — Base Shape Node Element data collection	40
Figure 36 — Base Shape Data collection.....	40
Figure 37 —Vertex Count Range data collection	41
Figure 38 — Vertex Shape Node Element data collection.....	42
Figure 39 — Vertex Shape Data collection	43
Figure 40 — Polyline Set Shape Node Element data collection	44
Figure 41 — Point Set Shape Node Element data collection	45
Figure 42 — Polygon Set Shape Node Element data collection	46
Figure 43 — NULL Shape Node Element data collection	46
Figure 44 — Primitive Set Shape Node Element data collection.....	47
Figure 45 — Primitive Set Quantization Parameters data collection	48
Figure 46 — Base Attribute Data collection.....	49
Figure 47 — Material Attribute Element data collection	51
Figure 48 — Texture Image Attribute Element data collection	54
Figure 49 — Texture Vers-1 Data collection	55
Figure 50 — Texture Environment data collection.....	58
Figure 51 — Texture Coord Generation Parameters data collection	61
Figure 52 — Inline Texture Image Data collection	62
Figure 53 — Image Format Description data collection	63
Figure 54 — Draw Style Attribute Element data collection	65
Figure 55 — Light Set Attribute Element data collection	67
Figure 56 — Infinite Light Attribute Element data collection	68
Figure 57 — Base Light Data collection.....	69
Figure 58 — Point Light Attribute Element data collection.....	71
Figure 59 — Spread Angle value with respect to the light cone	72

Figure 60 — Attenuation Coefficients data collection	73
Figure 61 — Linestyle Attribute Element data collection	74
Figure 62 — Pointstyle Attribute Element data collection	75
Figure 63 — Geometric Transform Attribute Element data collection	76
Figure 64 — Texture Coordinate Generator Attribute Element data collection	78
Figure 65 — Mapping Plane Element data collection	79
Figure 66 — Mapping Cylinder Element data collection	80
Figure 67 — Mapping Sphere Element data collection	81
Figure 68 — Mapping TriPlanar Element data collection	82
Figure 69 — Base Property Atom Element data collection	83
Figure 70 — Base Property Atom Data collection	83
Figure 71 — String Property Atom Element data collection	84
Figure 72 — Integer Property Atom Element data collection	85
Figure 73 — Floating Point Property Atom Element data collection	85
Figure 74 — JT Object Reference Property Atom Element data collection	86
Figure 75 — Date Property Atom Element data collection	87
Figure 76 — Late Loaded Property Atom Element data collection	88
Figure 77 — Vector4f Property Atom Element data collection	89
Figure 78 — Property Table data collection	90
Figure 79 — Element Property Table data collection	91
Figure 80 — Shape LOD Segment data collection	92
Figure 81 — Tri-Strip Set Shape LOD Element data collection	92
Figure 82 — Polyline Set Shape LOD Element data collection	93
Figure 83 — Point Set Shape LOD Element data collection	94
Figure 84 — Polygon Set LOD Element data collection	94
Figure 85 — Vertex Shape LOD Data collection	95
Figure 86 — Base Shape LOD Data collection	97
Figure 87 — TopoMesh Compressed LOD Data collection	97
Figure 88 — TopoMesh LOD Data collection	98
Figure 89 — TopoMesh Compressed Rep Data data collection	99
Figure 90 — Quantization Parameters data collection	101
Figure 91 — TopoMesh Topologically Compressed LOD Data collection	102
Figure 92 — Topologically Compressed Rep Data Collection	103
Figure 93 — Topologically Compressed Vertex Records data collection	106
Figure 94 — Null Shape LOD Element data collection	107
Figure 95 — Primitive Set Shape Element data collection	108
Figure 96 — Lossless Compressed Primitive Set Data collection	109
Figure 97 — Lossy Quantized Primitive Set Data collection	111
Figure 98 — Compressed params1 data collection	113
Figure 99 — JT LWPA Segment data collection	116
Figure 100 — JT LWPA Element data collection	116
Figure 101 — Analytic Surface Geometry data collection	117
Figure 102 — Analytic Surface Creation	119
Figure 103 — Wireframe Segment data collection	120
Figure 104 — Wireframe Rep Element data collection	121
Figure 105 — Wireframe MCS Curves Geometric Data collection	122
Figure 106 — Wireframe Rep CAD Tag Data collection	123
Figure 107 — Meta Data Segment data collection	123
Figure 108 — Property Proxy Meta Data Element data collection	124
Figure 109 — Date Property Value data collection	125
Figure 110 — PMI Manager Meta Data Element data collection	127
Figure 111 — PMI Design Group Entities data collection	128
Figure 112 — Design Group Attribute data collection	129
Figure 113 — PMI Associations data collection	131
Figure 114 — PMI User Attributes data collection	133
Figure 115 — PMI String Table data collection	134
Figure 116 — PMI Model Views data collection	135
Figure 117 — PMI Property data collection	137
Figure 118 — Key PMI Property Atom data collection	138

Figure 119 —Generic PMI Entity data collection.....	139
Figure 120 — PMI 2D Data collection	142
Figure 121 — PMI Base Data collection	142
Figure 122 — 2D-Reference Frame data collection	143
Figure 123 — 2D Text Data collection	144
Figure 124 — Text Box data collection	145
Figure 125 — Constructing Text Polylines from data arrays	146
Figure 126 — Text Polyline Data collection	146
Figure 127 — Constructing Non-Text Polylines from packed 2D data arrays	147
Figure 128 — Non-Text Polyline Data collection.....	148
Figure 129 — PMI CAD Tag Data collection.....	149
Figure 130 — PMI Polygon Data	151
Figure 131 — PMI Model View Sort Orders data collection	154
Figure 132 — Int32 Compressed Data Packet data collection.....	156
Figure 133 — Int32 Probability Context	159
Figure 134 — Int32 Probability Context Table Entry data collection	160
Figure 135 — Int64 Compressed Data Packet data collection.....	161
Figure 136 — Int64 Probability Context data collection	163
Figure 137 — Int64 Probability Context Table Entry data collection	163
Figure 138 — Compressed Vertex Coordinate Array data collection	164
Figure 139 — Compressed Vertex Normal Array data collection.....	166
Figure 140 — Compressed Vertex Texture Coordinate Array data collection	167
Figure 141 — Compressed Vertex Colour Array data collection.....	169
Figure 142 — Compressed Vertex Flag Array data collection.....	170
Figure 143 — Compressed Auxiliary Fields Array data collection.....	171
Figure 144 — Point Quantizer Data collection.....	174
Figure 145 — Texture Quantizer Data collection.....	175
Figure 146 — Colour Quantizer Data collection	176
Figure 147 — Uniform Quantizer Data collection.....	177
Figure 148 — Compressed Entity List for Non-Trivial Knot Vector data collection	178
Figure 149 — Compressed Control Point Weights Data collection	180
Figure 150 — Compressed Curve Data collection	182
Figure 151 — Non-Trivial Knot Vector NURBS Curve Indices data collection.....	184
Figure 152 — NURBS Curve Control Point Weights data collection	184
Figure 153 — NURBS Curve Control Points data collection.....	184
Figure 154 — Compressed CAD Tag Data collection	185
Figure 155 — Sextant Coding on the Sphere	194
Figure 156 — Assembly node with SUBNODE	203
Figure 157 — Assembly node without SUBNODE	204
Figure 158 — Displaying Nodes that have SUBNODE properties	204
Figure 159 — CAD Component with Reference sets	205
Figure 160 — JT Format Convention for Modeling each Part in LSG	208
Figure 161 —data collection.....	261
Figure 162 — JT B-Rep Element data collection.....	262
Figure 163 — Topological Entity Counts data collection	263
Figure 164 — Geometric Entity Counts data collection	264
Figure 165 — Topology Data collection	265
Figure 166 — Regions Topology Data collection.....	266
Figure 167 — Shells Topology Data collection	267
Figure 168 — Trim Loop example in parameter Space - One Face with 2 Holes.....	268
Figure 169 — Faces Topology Data collection.....	268
Figure 170 — Loops Topology Data collection	270
Figure 171 — CoEdges Topology Data collection	271
Figure 172 — Edges Topology Data collection	272
Figure 173 — Vertices Topology Data collection	273
Figure 174 — Geometric Data collection	274
Figure 175 — Surfaces Geometric Data collection	275
Figure 176 — Non-Trivial Knot Vector NURBS Surface Indices data collection.....	276
Figure 177 — NURBS Surface Degree data collection	277
Figure 178 — NURBS Surface Control Point Counts data collection	277

Figure 179 — NURBS Surface Control Point Weights data collection	278
Figure 180 — NURBS Surface Control Points data collection.....	278
Figure 181 — NURBS Surface Knot Vectors data collection	278
Figure 182 — PCS Curves Geometric Data collection.....	279
Figure 183 — Trivial PCS Curves data collection	280
Figure 184 — Equality of corresponding curve end coordinates of opposite sides of the box	281
Figure 185 — MCS Curves Geometric Data collection	282
Figure 186 — Point Geometric Data collection	283
Figure 187 — Topological Entity Tag Counters data collection	283
Figure 188 — B-Rep CAD Tag Data collection	284
Figure 189 — XT B-Rep Element data collection	285
Figure 190 — Integer Attribute Data collection	287
Figure 191 — MultiXT B-Rep Element data collection.....	288
Figure 192 — Split a face	367
Figure 193 — Merge faces	368
Figure 194 — JT ULP Segment data collection	370
Figure 195 — JT ULP Element data collection	371
Figure 196 — Topology Data collection	372
Figure 197 — Topological Entity Counts data collection	373
Figure 198 — Combined Predictor Type data collection.....	374
Figure 199 — Regions Topology Data collection	375
Figure 200 — Shells Topology Data collection	376
Figure 201 — Faces Topology Data collection.....	377
Figure 202 — Loops Topology Data collection	380
Figure 203 — CoEdges Topology Data collection	382
Figure 204 — Sample Model with Randomly Assigned Edge Indices	383
Figure 205 — Sample Model with Sequentially Assigned Edge Indices.....	383
Figure 206 — Surface Domain Classification	385
Figure 207 — Edges Topology Data collection	387
Figure 208 — Geometric Data collection	389
Figure 209 — Geometric Entity Counts.....	390
Figure 210 — Degree Table data collection	391
Figure 211 — Recover Nurbs Degree	392
Figure 212 — Number of Control Points Table data collection	393
Figure 213 — Recover Number of Control Points.....	394
Figure 214 — Dimension Table data collection.....	395
Figure 215 — Recover Dimension	396
Figure 216 — 3D Unit Vector Table data collection	397
Figure 217 — Recover Dimension	398
Figure 218 — 2D Unit Vector Table data collection	399
Figure 219 — Recover 2D Unit Vector.....	399
Figure 220 — 3D MCS Point Table data collection.....	400
Figure 221 — Recover 3D MCS Points	402
Figure 222 — Knot Vector Table data collection.....	403
Figure 223 — Recover Knot Vectors	404
Figure 224 — 1D MCS Table data collection.....	406
Figure 225 — Recover 1D MCS Table.....	408
Figure 226 — PCS Value Table data collection	409
Figure 227 — Recover PCS Value Table	410
Figure 228 — Radian Table data collection	411
Figure 229 — Recover Radian Table	411
Figure 230 — Weight Table data collection	412
Figure 231 — Recover Weight Table	413
Figure 232 — Material Attribute Element Properties	414
Figure 233 — Information Recovery	415
Figure 234 — PCS Curve Recovery from Surface Domain	416
Figure 235 — MCS Curve Recovery	417
Figure 236 — MCS Curve Recovery from Surface Geometry	418

Tables

Table 1 — Symbols.....	10
Table 2 — Predictor Type	12
Table 3 — Basic Data Types.....	14
Table 4 — Composite Data Types.....	14
Table 5 — Segment attributes.....	21
Table 6 — Segment Types	22
Table 7 — Object Base Types	25
Table 8 — Compression flag values	26
Table 9 — Compression algorithm values	26
Table 10 — Node Flag values	30
Table 11 — Partition flag bits	32
Table 12 — Compression level values	42
Table 13 — Texture Coord Gen Type values	48
Table 14 — Version Number values	48
Table 15 — State Flag values	50
Table 16 — Material Attribute data field inhibit values	50
Table 17 — Material Attribute Version number value	52
Table 18 — Material Attribute Data Flag values	52
Table 19 — Texture Image Attribute data field inhibit values	53
Table 20 — Texture Image Version Number values	54
Table 21 — Texture Vers-1 Texture Type values.....	56
Table 22 — Texture Vers-1 Inline Image Storage Flag values	57
Table 23 — Texture Vers-1 Texture Environment Border Mode values.....	58
Table 24 — Texture Vers-1 Texture Environment Mipmap Magnification Filter values	59
Table 25 — Texture Vers-1 Texture Environment Mipmap Minification Filter values	59
Table 26 — Texture Vers-1 Texture Environment S-Dimen Wrap Mode values.....	59
Table 27 — Texture Vers-1 Texture Environment Blend Type values	60
Table 28 — Texture Vers-1 Texture Environment Internal Compression Level values	60
Table 29 — Texture Vers-1 Texture Coord Generation Gen Mode values.....	61
Table 30 — Texture Vers-1 Image Format Description Pixel Format values.....	63
Table 31 — Texture Vers-1 Image Format Description Pixel Data values	64
Table 32 — Texture Vers-1 Image Format Description Dimensionality values.....	64
Table 33 — Texture Vers-1 Image Format Description Shared Image Flag values	65
Table 34 — Draw Style Attribute Field Inhibit flag values	65
Table 35 — Draw Style Attribute Data Flag values.....	66
Table 36 — Light Set Attribute Version Number values.....	68
Table 37 — Base Light Data Coord System values	70
Table 38 — Base Light Data Shadow Caster Flag values	70
Table 39 — Point Light Attribute Version Number values	72
Table 40 — Point Light Attribute Spread Angle values	72
Table 41 — Linestyle Attribute Data Flag values	74
Table 42 — Pointstyle Attribute Data Flag values.....	75
Table 43 — Geometric Transform Attribute Stored Value Mask individual bit-flag values	77
Table 44 — Mapping Plane Matrix Coordinate System values	79
Table 45 — Mapping Cylinder Matrix Coordinate System values	80
Table 46 — Mapping Sphere Matrix Coordinate System values.....	81
Table 47 — Mapping TriPlanar Matrix Coordinate System values	82
Table 48 — Vertex Shape LOD Bindings values	95
Table 49 — Primitive Set Shape Version Number values.....	109
Table 50 — Primitive Set Shape Texture Coord Gen Type values	109
Table 51 — Lossless Compressed Primitive Set Data Field values.....	110
Table 52 — Primitive Set “params#” Data Fields Interpretation.....	110
Table 53 — Property Proxy Meta Data Property Value Type values	125
Table 54 — PMI Design Group Attribute Type values.....	129
Table 55 — PMI Associations Source Data values	131
Table 56 — PMI Associations Reason Code values	132
Table 57 — PMI Model Views Active Flag values	136
Table 58 — Common Property Keys and Their Value Encoding formats.....	137
Table 59 — PMI Property Atom Hidden Flag values	139

Table 60 — Generic PMI Entity Type values	140
Table 61 — Generic PMI User Flag values	141
Table 62 — PMI 2D Base Data Font values	144
Table 63 — PMI 2D Non-Text Polyline Type values	148
Table 64 — Int32 Probability Contexts CODEC Type values	157
Table 65 — TopoMesh Compressed Rep Data V2 Field Type values	172
Table 66 — Colour Quantizer values	175
Table 67 — Colour Quantizer HSV Flag values	176
Table 68 — Knot Type Exist Flag values.....	179
Table 69 — Compressed Curve Base Type values	182
Table 70 — NURB UV Curve entity dimensionality values.....	183
Table 71 — NURB XYZ Curve entity dimensionality values.....	183
Table 72 — Compressed CAD Tag Type values.....	186
Table 73 — Example assigned probability values	189
Table 74 — Example “probability line” values	190
Table 75 — Example input integer sequence values.....	190
Table 76 — Example integer number sequence values	191
Table 77 — CAD Properties	200
Table 78 — Tessellation Property values.....	201
Table 79 — Miscellaneous Property values	202
Table 80 — SUBNODE Property	204
Table 81 — Reference Set Properties.....	205
Table 82 — Properties related to the use of Reference Sets	206
Table 83 — REFSET_CURRENT property	207
Table 84 Texture watermark properties	209
Table 85 — JT B-Rep Shell Topology Anti-Hole Flag values	267
Table 86 — JT B-Rep Face Reverse Normal Flag values	269
Table 87 — JT B-Rep Loops Topology Data Anti-Hole Flag values	270
Table 88 — JT B-Rep MCS Curve Reversed Flag values	271
Table 89 — JT B-Rep Surface Base Type value	275
Table 90 — JT B-Rep NURBS Surface Control Point Dimensionality values.....	276
Table 91 — Trivial Domain Loops Exist Flag values	280
Table 92 — Trivial Box Loops Exist Flag values.....	281
Table 93 — Trivial Domain UV Curves Exist Flag values.....	281
Table 94 — Trivial UV Curve Para Domain Side Codes values.....	282
Table 95 — Moniker attribute use example.....	366
Table 96 — MONIKER/GUID_TABLE_ATTRIB and MONIKER/MONIKER_DATA_ATTRIB	368
Table 97 — MONIKER/BODY_ID_ATTRIB	369
Table 98 — JT ULP Shell Anti-Hole Flag values.....	376
Table 99 — JT ULP Flag Bit Array Look Index values	378
Table 100 — JT ULP Supported Surface Type values	378
Table 101 — JT ULP Supported Knot Type Values.....	379
Table 102 — JT ULP Face Reverse Normal Flag values.....	379
Table 103 — JT ULP Loops Topology Flag Bit Array values	380
Table 104 — JT ULP Loops Topology Reverse Normal Flag values.....	381
Table 105 — JT ULP Recover Edge Indices Flag Bit Array values.....	384
Table 106 — JT ULP Recover Edge Indices PCS curve type values.....	385
Table 107 — JT ULP PCS Curve Type values.....	385
Table 108 — JT ULP PCS Curve Type XYZ Reversed Flag values	385
Table 109 — JT ULP PCS Curve Type isUVInc Flag values	386
Table 110 — JT ULP Edges Topology Recover MCS Curve Indices Flag Bit Array values	388
Table 111 — JT ULP Edges Topology Recover MCS Curve Type values.....	388
Table 112 — Parameter Domain.....	407
Table 184 — PMI Properties	454

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's permission.

Siemens will enforce its copyright. Siemens's intention is to maintain the integrity of the JT Data Format standard, 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's copyrighted list of data structures and operators in the written specification to the extent 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's products that use the 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 10.0 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 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
- Triangle sets, Polygon sets, Point sets, Line sets and Implicit Primitive sets (cylinder, cone, sphere, etc...)
- Full array of visual attributes: Materials, Textures, Lights
- 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 frustum and occlusion culling and fixed-framerate display modes.

Layers, and Layer Filters.

Along with the pure syntactical definition of the JT Format, there is also series of conventions which although not required to have a reference compliant JT file, have become commonplace within JT format translators. These conventions have been documented in the “Best Practices” section of this JT format reference.

This JT format reference does not specifically address implementation of, nor define, a run-time architecture for viewing and/or processing JT data. This is because although the JT format is closely aligned with a run-time data representation for fast and efficient loading/unloading of data, no interaction behavior is defined within the format itself, either in the form of specific viewer controls, viewport information, animation behavior or other event-based interactivity. This exclusion of interaction behavior from the JT format makes the format more easily reusable for dissimilar application interoperability and also facilitates incremental update, without losing downstream authored data, as the original CAD asset revises.

2.1 What's New in This Revision

Revision A

Revision B

1. Corrections made in [Figure 104 — Wireframe Rep Element data collection](#), Lag1 is replaced by NULL in two places.
2. Added best practice sections [13.8.6 The SUBNODE property](#) and [13.8.6.1 Reference Sets and the Reference Set Property](#)

Revision C

1. Reformatting of best practices section, now called Common Data Conventions and Constructs. Additional reformatting and arrangement of content in Precise Geometry section, B-Rep descriptions now provided as separate Annexs
2. Remove remaining shader Cg shader references
3. Figure 233 - Int32 Compressed Data Packet data collection data box for Compressed Data Packet: OOB Values was not fully expanded
4. Bit length decoder code was not complete, additions added to sample Bitlength Decoder source.

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

3.1.1

Assembly

A related collection of model parts, represented in a JT format logical scene graph as a logical graph branch.

3.1.2

Attribute

Objects associated with nodes in a logical scene graph and specifying one of several appearances, positioning, or visual characteristics of a shape

3.1.3

Boundary representation

Solid model representation where the solid volume is specified by its surface boundary (both its geometric and topological boundaries)

3.1.4

Code text

Collection of data in encoded form

3.1.5

Coordinate system

A system which uses one or more numbers, or coordinates, to uniquely determine the position of a point or other geometric element

NOTE 1 If not otherwise specified in a data field's description, it is assumed that the data is defined in Local Coordinate System.

3.1.6

Directed acyclic graph

Graph that consists of a set of nodes and a set of edges that connect the nodes in a tree like structure

NOTE 1 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).

NOTE 2 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.

NOTE 3 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.

3.1.7

JT enabled application

Application which supports reading and/or writing reference compliant JT format files

3.1.8

Level of detail

LOD

Alternative graphical representation for some model component such as part

3.1.9

Local coordinate system

LCS

Coordinate system that is used to specify the raw data of the shape geometry with no transforms applied

3.1.10

Logical scene graph

LSG

Scene graph representing the logical organization of a model

NOTE A scene graph contains shapes and attributes representing the model's physical components, properties identifying arbitrary metadata (e.g. names, semantic roles) of those components, and a hierarchical structure expressing the component relationships.

3.1.11

Mipmap

Reduced resolution version of a texture map

NOTE 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.

3.1.12

Model

Representation, in JT format, of a physical or virtual product, part, assembly; or collections of such objects

3.1.13

Model coordinate system

MCS

Local coordinates transformed by any transforms specified as attributes at or above the node

3.1.14

Product and manufacturing information

PMI

Collection of information created on a 3D/2D CAD model to completely document the product with respect to design, manufacturing, inspection, etc.

NOTE This can include 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).

3.1.15

Property

Object associated with a logical scene graph node and identifying arbitrary application or enterprise specific information (meta-data) related to that node

3.1.16

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

3.1.17

Scene graph

Directed acyclic graph that arranges the logical and often (but not necessarily) spatial representation of a graphical scene

3.1.18

Streaming

Loading from disk based medium only the portions of data that are required by the user to perform the tasks at hand

NOTE 1 The motivation for streaming is to more efficiently manage system memory.

NOTE 2 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.

NOTE 3 The motivation for streaming is 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.

NOTE 4 The desired end result of streaming is to deliver only the JT data that the user needs, where the user needs it, when the user needs it.

3.1.19

Shape

Logical scene graph leaf node containing or referencing the geometric shape definition data (such as vertices, polygons and normals) of a model component

3.1.20

Texture channel

Texture unit plus the texture environment.

NOTE The JT format meaning for texture channel is the same as in OpenGL [1].

3.1.21

Texture object

Named cache that stores texture data, such as the image array, associated mipmaps, and associated texture parameter values: width, height, border width, internal format, resolution of components, minification and magnification filters, wrapping modes, border colour, and texture priority

NOTE The JT format meaning for texture object is the same as in OpenGL [1].

3.1.22

Texture unit

A hardware unit used to sample and filter a texture image.

NOTE The JT format meaning for texture unit is the same as in OpenGL [1].

3.1.23

View coordinate system

World coordinates transformed by a view matrix

3.1.24

World coordinate system

WCS

Node coordinates transformed by transforms inherited from a node's parent (i.e. the coordinate system at the root of the graph)

3.2 Abbreviated terms

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

Abs	Absolute Value
Bbox	Bounding Box
B-Rep	Boundary Representation
CAD	Computer Aided Design

CODEC	Coder-Decoder
GD&T	Geometric Dimensioning and Tolerancing
GUID	Globally Unique Identifier
HSV	Hue, Saturation, Value
LsbFirst	Least Significant Byte First
Max	Maximum
Min	Minimum
MsbFirst	Most Significant Byte First
N/A	Not Applicable
PCS	Parameter Coordinate Space
PLM	Product Lifecycle Management
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha
TOC	Table of Contents
VPCS	Viewpoint Coordinate System
URL	Uniform Resource Locator

4 Notational conventions

4.1 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:

Table 1 — Symbols

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

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

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

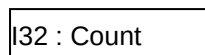


Figure 1 — rectangle box diagram

Diagram “folder” (i.e. logical data collections) symbols are simply titled with a collection name. In the example below the collection name is “Graph Elements.”

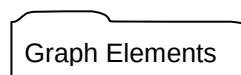


Figure 2 — folder diagram

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



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

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

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

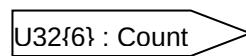


Figure 4 — rectangle box with clipped right side corners

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

The two supported compressed data packet types are:

Int32CDP – The Int32CDP (i.e. Int32 Compressed Data Packet) represents a third-generation format used to encode/compress a collection of data into a series of Int32 based symbols. This version of the Int32CDP supersedes the two similarly-named ones from the Version 9 JT Specification, and should not be confused with either of its predecessors. A complete description for Int32 Compressed Data Packet can be found in 12.1.1 Int32 Compressed Data Packet.

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

In the example below the Data Type is “VecU32”, Int32 Compressed Data Packet type is used, Lag1 Predictor Type is used, and Field Name is “First Shell Index.”

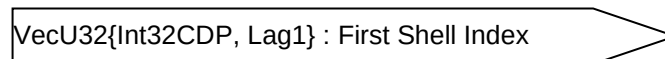


Figure 5 — compressed data packet diagram

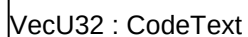
As mentioned above (with Predictor Type algorithm), the *primal* input data values are NOT always what is encoded/compressed. This is because the *primal* input data is first run through a Predictor Type algorithm, which produces an output array of residual values (i.e. difference from the predicted 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 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 Annex C.

Table 2 — Predictor Type

Predictor Type	Description
Lag1	Predicts as last value
Xor1	Predict as 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 “Combined:”. For example, “Combined:Lag1” means that predictor type “Lag1” is combined with additional preprocessing steps. Additional description about the processing steps is provided whenever such combined predictor is used.

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



VecU32 : CodeText

Figure 6 — data type : field name diagram

Note that for some JT file [Segment Types](#) there is LZMA compression also applied to all bytes of element data stored in the segment. This LZMA compression applied to all the segment’s data is not indicated in the diagrams through the use of “rectangle box with clipped right side corners”. Instead, one shall examine information stored with the first Element in the file segment to determine if LZMA compression is applied to all data in the segment. A complete description of the JT format data compression and encoding can be found in 5.1.3 Data Segment and 12 Data Compression and Encoding.

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

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

For folders (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.

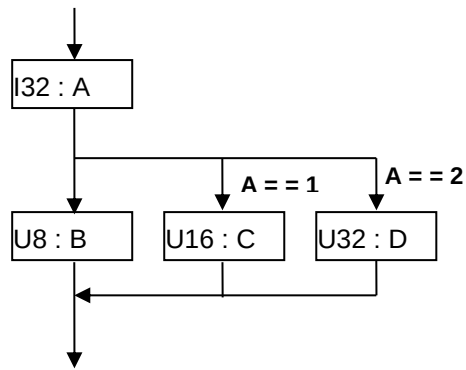


Figure 7 — data filed dependency example

In cases where the same data type repeats, a loop construct is used where the number of iterations appears next to the loop line. There are two forms of this loop construct. The first form is used when the number of iterations is not controlled by some previous read count value. Instead the number of iterations is either a hard coded count (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:

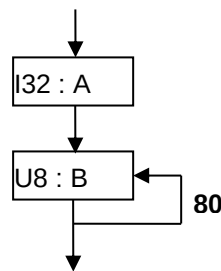


Figure 8 — loop construct example

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 form of the loop construct looks as follows (data field D is repeated C value times).

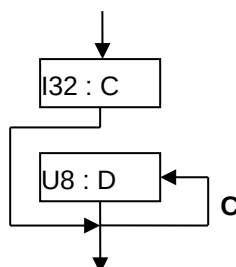


Figure 9 — loop construct with iterations example

4.2 Data Types

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

Table 3 — Basic Data Types lists the basic/standard data types which can occur in JT file.

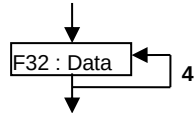
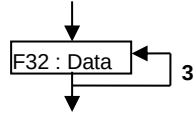
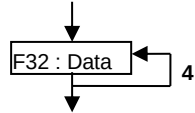
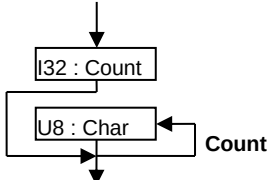
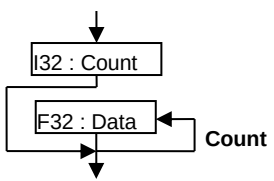
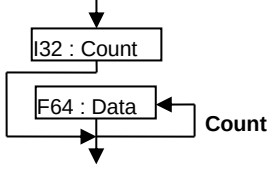
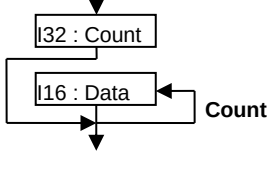
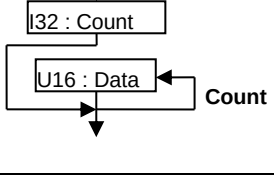
Table 3 — Basic Data Types

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

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

Table 4 — Composite Data Types

Type	Description	Symbolic Diagram
BBoxF32	The BBoxF32 type defines a bounding box using two CoordF32 types to store the XYZ coordinates for the bounding box minimum and maximum corner points.	
CoordF32	The CoordF32 type defines X, Y, Z coordinate values. So a CoordF32 is made up of three F32 base types.	
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.)	
MbString	The MbString type starts with an I32 that defines the number of characters (NumChar) the string contains. The number of bytes of character data is "2 * NumChar" (i.e. the strings are written out as multi-byte characters where each character is U16 size).	
Mx4F32	Defines a 4-by-4 matrix of F32 values for a total of 16 F32 values. The values are stored in row major order (right most subscript, column varies fastest), that is, the first 4 elements form the first row of the matrix.	
Mx4F64	Defines a 4-by-4 matrix of F64 values for a total of 16 F64 values. The values are stored in row major order (right most subscript, column varies fastest), that is, the first 4 elements form the first row of the matrix.	
PlaneF32	The PlaneF32 type defines a geometric Plane using the General Form of the plane equation ($Ax + By + Cz + D = 0$). The PlaneF32 type is made up of four F32 base types where the first three F32 define the plane unit normal vector (A, B, C) and the last F32 defines the negated perpendicular distance (D), along normal vector, from the origin to the plane.	

Quaternion	The Quaternion type defines a 3-dimensional orientation (no translation) in quaternion linear combination form ($a + bi + cj + dk$) where the four scalar values (a, b, c, d) are associated with the 4 dimensions of a quaternion (1 real dimension, and 3 imaginary dimensions). So the Quaternion type is made up of four F32 base types.	
RGB	The RGB type defines a colour composed of Red, Green, Blue components, each of which is a F32. So a RGB type is made up of three F32 base types. The Red, Green, Blue colour values typically range from 0.0 to 1.0.	
RGBA	The RGBA type defines a colour composed of Red, Green, Blue, Alpha components, each of which is a F32. So a RGBA type is made up of four F32 base types. The Red, Green, Blue colour values typically range from 0.0 to 1.0. The Alpha value ranges from 0.0 to 1.0 where 1.0 indicates completely opaque.	
String	The String type starts with an I32 that defines the number of characters (NumChar) the string contains. The number of bytes of character data is "NumChar" (i.e. the strings are written out as single-byte characters where each character is U8 size).	
VecF32	The VecF32 type defines a vector/array of F32 base type. The type starts with an I32 that defines the count of following F32 base type data. So a VecF32 is made up of one I32 followed by that number of F32. Note that it is valid for the I32 count number to be equal to "0", indicating no following F32.	
VecF64	The VecF64 type defines a vector/array of F64 base type. The type starts with an I32 that defines the count of following F64 base type data. So a VecF64 is made up of one I32 followed by that number of F64. Note that it is valid for the I32 count number to be equal to "0", indicating no following F64.	
VecI16	The VecI16 type defines a vector/array of I16 base type. The type starts with an I32 that defines the count of following I16 base type data. So a VecI16 is made up of one I32 followed by that number of I16. Note that it is valid for the I32 count number to be equal to "0", indicating no following I16.	
VecU16	The VecU16 type defines a vector/array of U16 base type. The type starts with an I32 that defines the count of following U16 base type data. So a VecU16 is made up of one I32 followed by that number of U16. Note that it is valid for the I32 count number to be equal to "0", indicating no following U16.	

VecI32	The VecI32 type defines a vector/array of I32 base type. The type starts with an I32 that defines the count of following I32 base type data. So a VecI32 is made up of one I32 followed by that number of I32. Note that it is valid for the I32 count number to be equal to "0", indicating no following I32.	
VecU32	The VecU32 type defines a vector/array of U32 base type. The type starts with an I32 that defines the count of following U32 base type data. So a VecU32 is made up of one I32 followed by that number of U32. Note that it is valid for the I32 count number to be equal to "0", indicating no following U32.	

4.3 Empty field

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

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

Refer to Empty Field guidelines in the Common Data Conventions section

5 File Format

All objects represented in the JT format are assigned an “object identifier” (e.g. see Base Node Data, or Base Attribute DataBase Attribute Data) and all references from one object to another object are represented in the JT format using the referenced object’s “object 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 out to disk. Where “pointer swizzling” refers to the process of converting references based on object identifiers into direct memory pointer references and “pointer unswizzling” is the reverse operation (i.e. replacing references based on memory pointers with object identifier references).

5.1 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 shall always exist to have a reference compliant JT file is the 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.

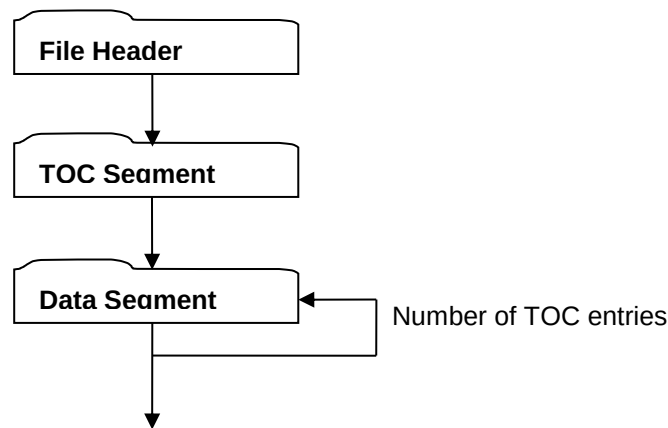


Figure 10 — JT File Structure

5.1.1 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:

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

0 – Least Significant byte first (LsbFirst)

1 – Most Significant byte first (MsbFirst)

I32: Empty Field

Refer to Common Data Conventions and Constructs Empty Field description.

U64: TOC Offset

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

GUID: LSG Segment ID

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

GUID: Empty Field

Refer to Common Data Conventions and Constructs Empty Field description.

5.1.2 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 segment's "TOC Offset" field. The TOC Segment contains one TOC Entry for each individually addressable Data Segment in the file.

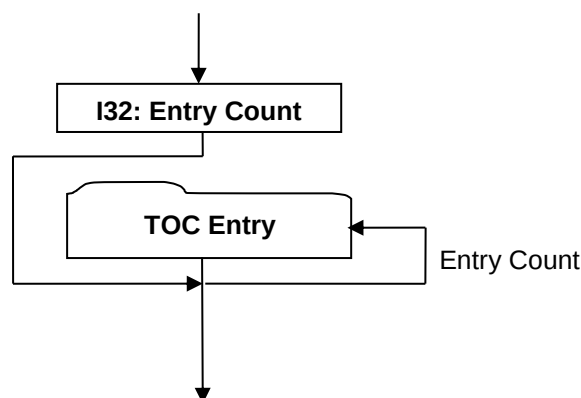


Figure 12 — TOC Segment data collection

I32: Entry Count

Entry Count is the number of entries in the TOC.

TOC Entry

Each TOC Entry 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.

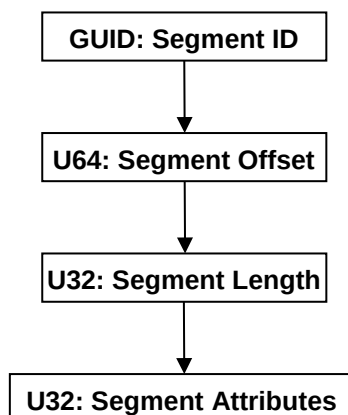


Figure 13 — TOC Entry data collection

GUID: Segment ID

Segment ID is the globally unique identifier for the segment.

U64: Segment Offset

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

U32: 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.

Table 5 — Segment attributes

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

5.1.3 Data Segment

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

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

All Data Segments have the same basic structure.

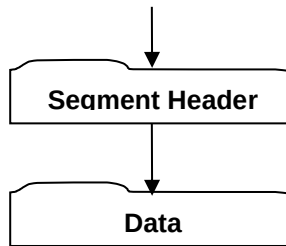


Figure 14 — Data Segment data collection

5.1.3.1 Segment Header

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

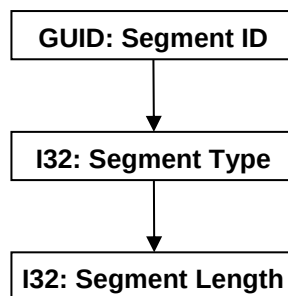


Figure 15 — Segment Header data collection

GUID: Segment ID

Global Unique Identifier for the segment.

I32: Segment Type

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

The complete list of segment and whether or not they support compression on all Data bytes in the payload is as follows:

Table 6 — Segment Types

Type	Data Contents	Compression
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

Type	Data Contents	Compression
13	Shape LOD6	No
14	Shape LOD7	No
15	Shape LOD8	No
16	Shape LOD9	No
17	XT B-Rep	Yes
18	Wireframe Representation	Yes
20	ULP	Yes
24	LWPA	Yes
30	MultiXT B-Rep	Yes
32	STEP B-rep	Yes

NOTE 1 Segment Types 7-16 all identify the contents as LOD Shape data, where the increasing type number is intended to convey some notion of how high an LOD the specific shape segment represents. The lower the type in this 7-16 range the more detailed the Shape LOD (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 2 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;
- no way to determine what LOD a Shape Segment represents.

132: 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 complete segment) and should be equal to the length value stored with this segment's TOC Entry.

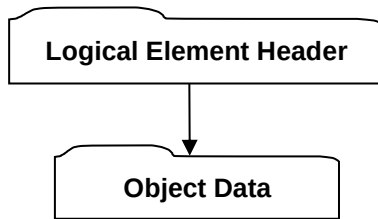
5.1.3.2 Data

The interpretation of the Data section depends on the Segment Type. See 5.2 Data Segments for complete description for all Data Segments 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 Element's storage format. In addition, another level of compression (i.e. LZMA compression) may be conditionally applied to all bytes of information stored for all Elements within a particular Segment. Not all Segment types support compression on all Segment data as indicated Table 8 — Segment Types If a particular file Segment is of the type which supports compression on all the Segment data, whether this compression is applied or not is indicated by data values stored in the Logical Element Header Compressed data collection of the first Element within the Segment. An in-depth description of JT file compression/encoding techniques can be found in 12 Data Compression and Encoding.

For Segment Types that do NOT support compression on all Segment Data.
(See Table 6 — Segment Types.)



For Segment Types that do support compression on all Segment Data.
(See Table 6 — Segment Types.)

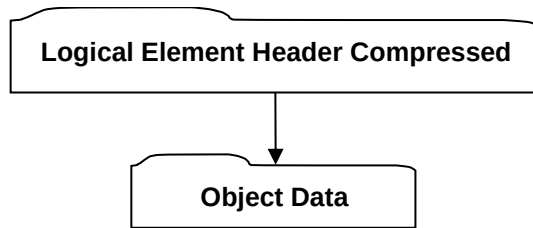


Figure 16 — Data collection

5.1.3.2.1 Logical Element Header

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

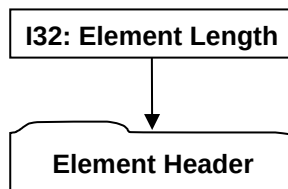


Figure 17 — Logical Element Header data collection

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

I32: Element Length

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

Element Header

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

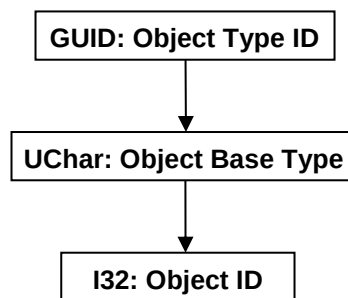


Figure 18 — Element Header data collection

GUID: Object Type ID

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

UChar: Object Base Type

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

Valid Object Base Types include the following:

Table 7 — Object Base Types

Base Type	Description	Base Type's Data Format
255	None	None
0	Base Graph Node Object	Base Node Data
1	Group Graph Node Object	Group Node Data
2	Shape Graph Node Object	Base Shape Data
3	Base Attribute Object	Base Attribute Data
4	Shape LOD	None
5	Base Property Object	6.2.1.1 Base Property Atom Data
6	JT Object Reference Object	JT Object Reference Property Atom Element without the Logical Element Header Compressed data collection.
8	JT Late Loaded Property Object	Late Loaded Property Atom Element without the Logical Element Header Compressed data collection.
9	JtBase (none)	None

I32: Object ID

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

5.1.3.2.2 Logical Element Header Compressed

Logical Element Header Compressed data collection is the format of Element Header data used by all Elements within Segment Types that support compression on all data in the Segment. See **Table 6 — Segment Types** for information on whether a particular Segment Type supports compression on all data in the Segment.

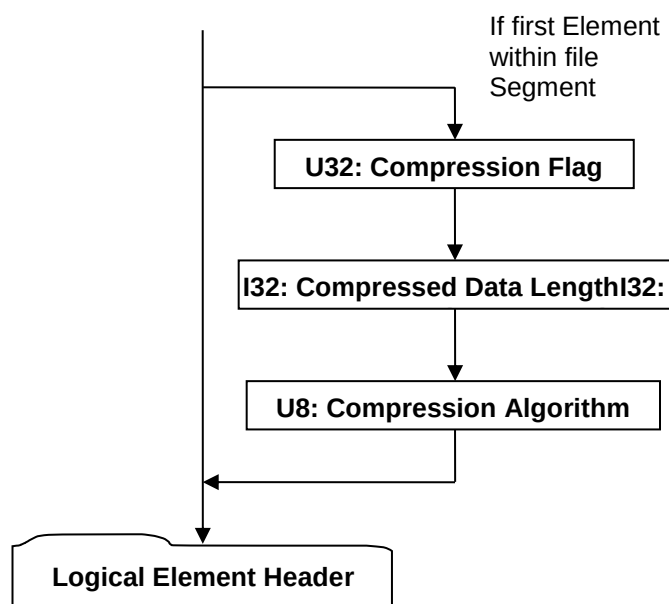


Figure 19 — Logical Element Header Compressed data collection

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

U32: Compression Flag

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

Table 8 — Compression flag values

= 3	LZMA compression is ON
!= 3	LZMA No Compression

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:

Table 9 — Compression algorithm values

= 1	No compression
= 3	LZMA compression

5.1.3.2.3 Object Data

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

5.2 Data Segments

A JT file may consists of the following segments of data:

- 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 representing the model's physical components, properties identifying arbitrary metadata (e.g. names, semantic roles) of those components, and a hierarchical structure expressing the component relationships.
- Shape LOD segment contains an element that defines the geometric shape definition data (e.g. vertices, polygons, normals, etc) for a particular shape level of detail or alternative representation.
- (deprecated)
JT B-Rep Segment contains an element that defines the precise geometric boundary representation data for a particular Part in JT B-Rep format.
- XT B-Rep Segment contains an element that defines the recommended precise geometric boundary representation data for a particular part in boundary representation format.
- Wireframe Segment contains an element that defines the precise 3D wireframe data for a particular part.

- Meta Data Segments is used to store large collections of meta-data in separate addressable segments of the JT File. Storing meta-data in a separate addressable segment allows references (from within the JT file) to these segments to be constructed such that the meta-data can be late-loaded.
- JT ULP Segment contains an element that defines the semi-precise geometric boundary representation data for a particular part in JT ULP format.
- JT LWPA Segment contains an element that defines light weight precise analytic data for a particular part. More specifically LWPA contains the collection of analytic surfaces in the b-rep definition of the part.
- Multi XT B-Rep Segment contains an element that defines the precise geometric boundary representation data for one or more parts in boundary representation format.

The following segments are deprecated as of this specification description; they are included here for completeness but should be considered only for legacy purposes for future reference:

- JT B-Rep Segment containing an element that defines the precise geometric boundary representation data for a particular Part in JT B-Rep format;
- The PMI manager meta data element can sometimes also be represented in a PMI data segment. This can occur when a pre JT 8 version file is migrated to JT 9.5 version file. So from a parsing point of view a PMI data segment should be treated exactly the same as a meta data segment.

For completed information on all the segments of a JT file see the segment descriptions for each segment in their specific sections in this document.

6 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 representing the model's physical components, properties identifying arbitrary metadata (e.g. names, semantic roles) of those components, and a hierarchical structure expressing the component relationships. The "directed" nature of the LSG references implies that there is by default "state/attribute" inheritance from ancestor to descendant (i.e. predecessor to successor).

The first Graph Element in a LSG Segment should always be a Partition Node. The LSG Segment type supports compression on all element data, so all elements in LSG Segment use the Logical Element Header Compressed form of element header data. i.e this means the 3 compression related fields in the Logical Element Header Compressed appear in the first graph element only.

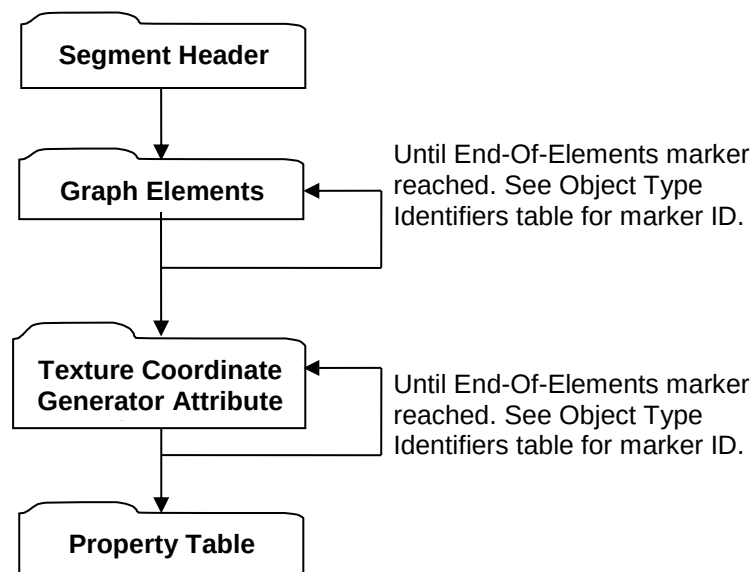


Figure 20 — LSG Segment data collection

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

6.1 Graph Elements

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

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

Attribute Elements represent graphical data (like appearance characteristics (e.g. colour), 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 behaviour. The following sub-sections describe each of the Node and Attribute Element types.

6.1.1 Node Elements

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

6.1.1.1 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 behaviour nor can it contain any children nodes.

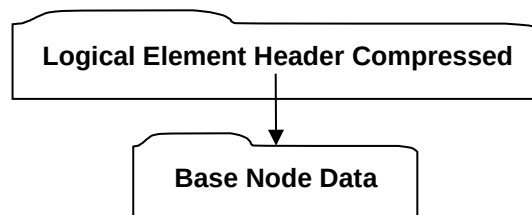


Figure 21 — Base Node Element data collection

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

6.1.1.1.1 Base Node Data

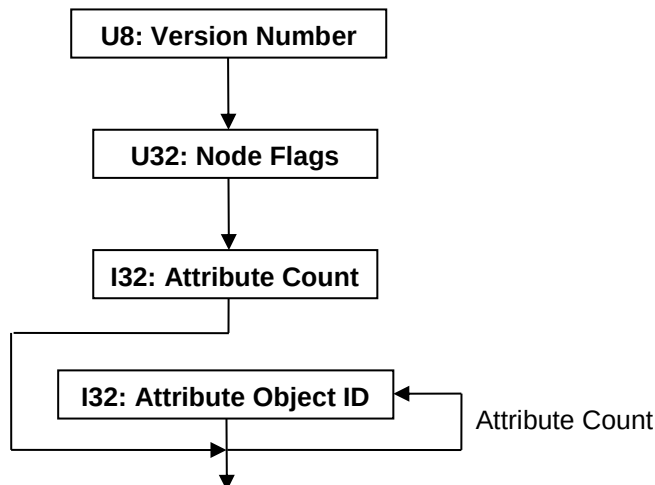


Figure 22 — Base Node Data collection

U8: Version Number

Version Number is the version identifier for this node. For information on local version numbers see Common Data Conventions and Constructs Local version numbers.

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 bits fields that are not defined as in use should be set to "0".

Table 10 — Node Flag values

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

I32: Attribute Count

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

I32: Attribute Object ID

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

6.1.1.2 Partition Node Element

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

A partition in a JT file must always be either the root or leaf node. A leaf 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).

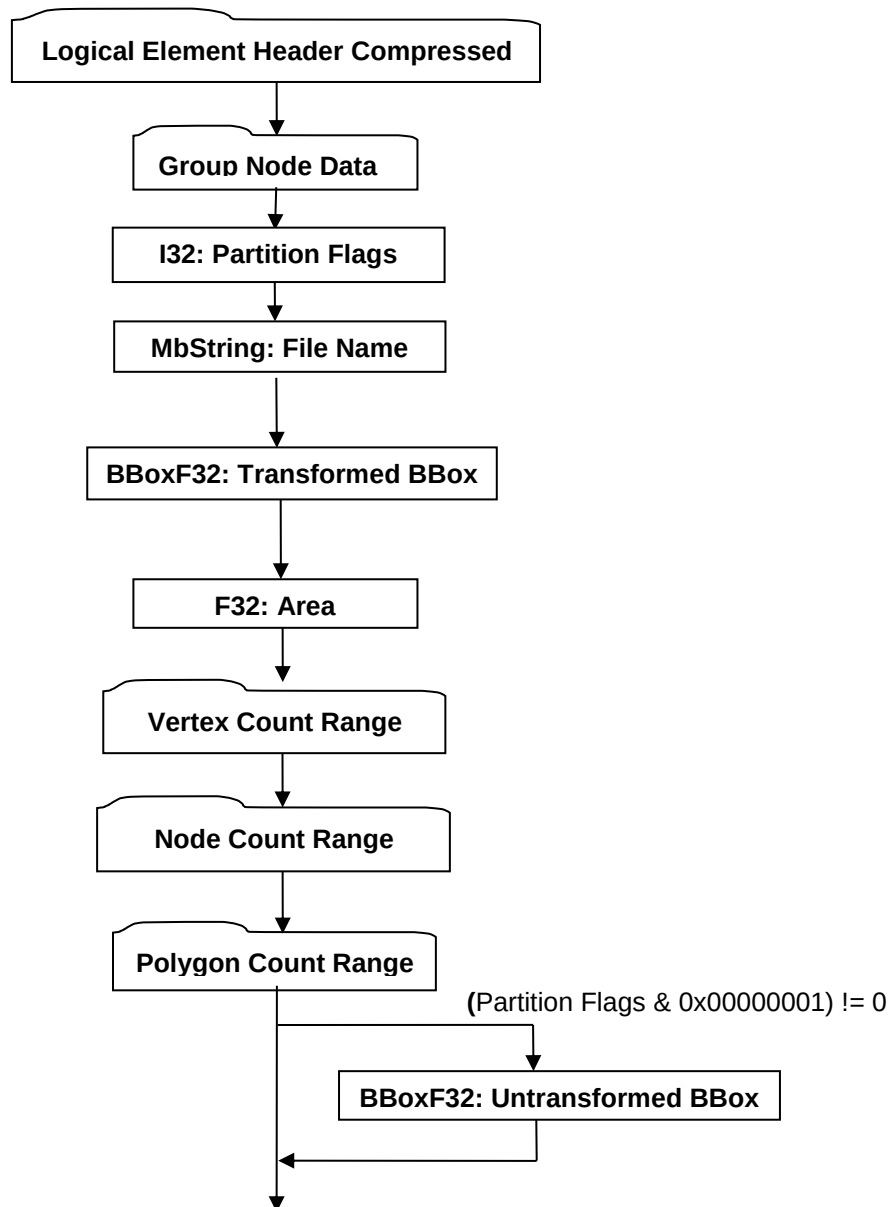


Figure 23 — Partition Node Element data collection

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

Complete description for Group Node Data can be found in Group 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 bits fields that are not defined as in use should be set to "0".

Table 11 — Partition flag bits

0x00000001	Untransformed bounding box is written.

MbString: File Name

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

BBoxF32: Transformed BBox

The Transformed BBox is an MCS axis aligned bounding box and represents the transformed geometry extents for all geometry contained in the Partition Node. This bounding box information may be used by a renderer of 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 descendants. This value is stored in MCS coordinate space (i.e. values scaled by MCS scaling).

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 minimum value represents the least vertex count that can be achieved by the Partition Node's descendants. The maximum value represents the greatest vertex count that can be achieved by the Partition Node's descendants.

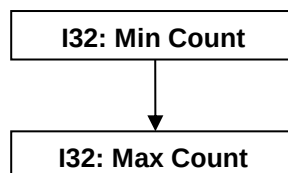


Figure 24 — Vertex Count Range data collection

I32: Min Count

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

I32: Max Count

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

Node Count Range

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

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

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. The minimum value represents the least polygon count that can be achieved by the Partition Node's descendants. The maximum value represents the greatest polygon count that can be achieved by the Partition Node's descendants.

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

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).

6.1.1.3 Group Node Element

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

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

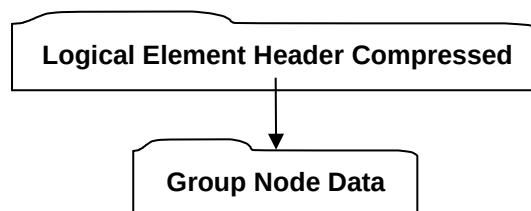


Figure 25 — Group Node Element data collection

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

6.1.1.3.1 Group Node Data

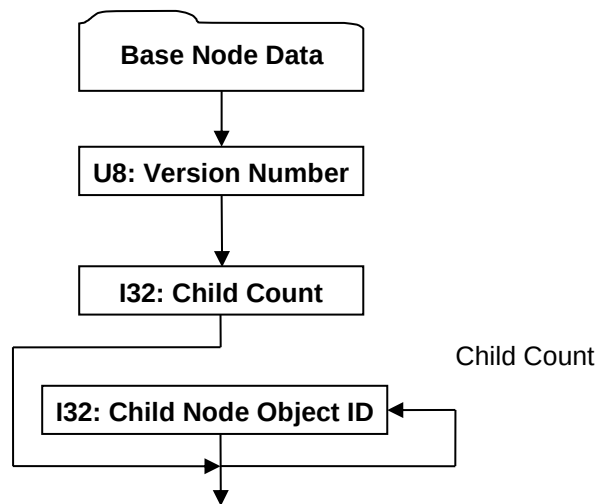


Figure 26 — Group Node Data collection

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

U8: Version Number

Version Number is the version identifier for this node. For information on local version numbers see Common Data Conventions and Constructs Local version numbers.

I32: Child Count

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

I32: Child Node Object ID

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

6.1.1.4 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.

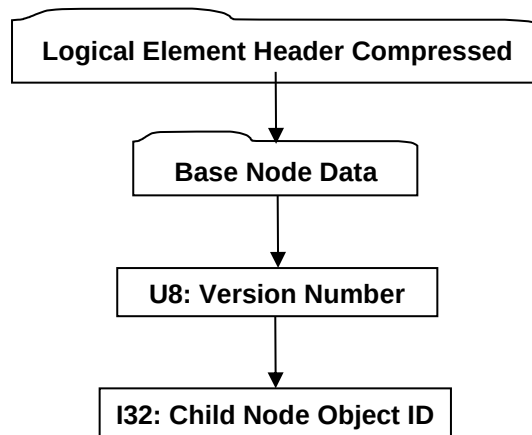


Figure 27 — Instance Node Element data collection

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

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

U8: Version Number

Version Number is the version identifier for this node. For information on local version numbers see Common Data Conventions and Constructs Local version numbers.

I32: Child Node Object ID

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

6.1.1.5 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).

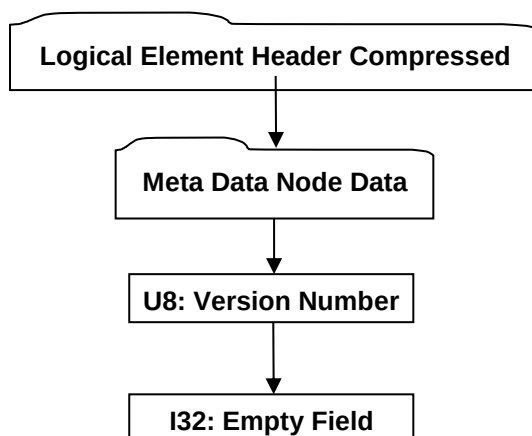


Figure 28 — Part Node Element data collection

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

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

U8: Version Number

Version Number is the version identifier for this node. For information on local version numbers see Common Data Conventions and Constructs Local version numbers.

I32: Empty Field

Refer to Common Data Conventions and Constructs Empty Field description.

6.1.1.6 Meta Data Node Element

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

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

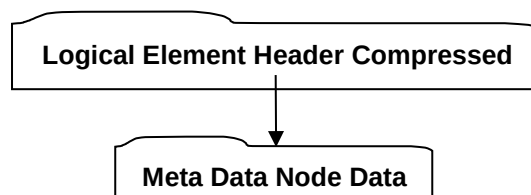


Figure 29 — Meta Data Node Element data collection

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

6.1.1.6.1 Meta Data Node Data

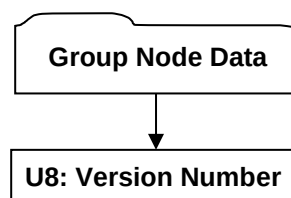


Figure 30 — Meta Data Node Data collection

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

U8: Version Number

Version Number is the version identifier for this data. For information on local version numbers see Common Data Conventions and Constructs Local version numbers.

6.1.1.7 LOD 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.

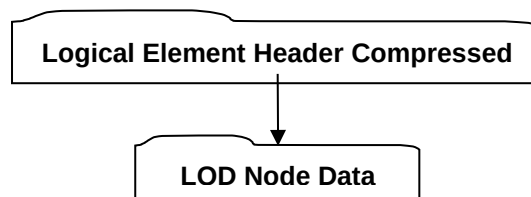


Figure 31 — LOD Node Element data collection

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

6.1.1.7.1 LOD Node Data

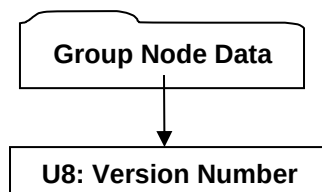


Figure 32 — LOD Node Data collection

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

U8: Version Number

Version Number is the version identifier for this node. For information on local version numbers see Common Data Conventions and Constructs Local version numbers.

6.1.1.8 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 centre point and the eye point, within which the corresponding alternate representation is appropriate. Traversers of LSG consult these range limit values when making an alternative representation selection.

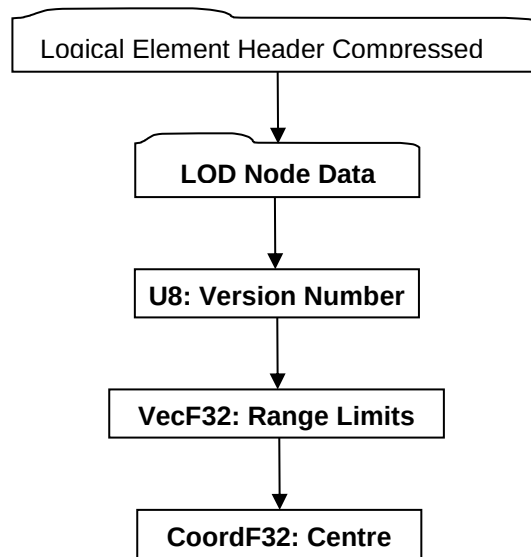


Figure 33 — Range LOD Node Element data collection

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

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

U8: Version Number

Version Number is the version identifier for this node. For information on local version numbers see Common Data Conventions and Constructs Local version numbers

VecF32: Range Limits

Range Limits indicate the WCS distance between a specified centre point and the eye point, within which the corresponding alternate representation is appropriate. It is not required that the count of range limits is equivalent to the number of alternative representations. These values are considered “soft values” in that loaders/viewers of 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 centre and the eye point is less than or equal to the first range limit (and when no range limits are specified). The second alternate representation is valid when the distance is greater than the first limit and less than or equal to the second limit, and so on. The last alternate representation is valid for all distances greater than the last specified limit.

CoordF32: Centre

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

6.1.1.9 Switch Node Element

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