INTERNATIONAL STANDARD

ISO 14306

First edition
2012-12-15

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

Systèmes d'automatisation industrielle et intégration — Spécification de format de fichier JT pour visualisation 3D

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Reference number
ISO 14306:2012(E)

© ISO 2012

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

COPYRIGHT PROTECTED DOCUMENT

© ISO 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

© ISO 2012 – All rights reserved

Contents

Page

Foreword xv

Introduction xvi

1 Scope 1

2 Terms, definitions and abbreviated terms 1
2.1 Terms and definitions 1
2.2 Abbreviated terms 4

3 Notational conventions 5

4 File Format 12
4.1 File Structure 12
4.1.1 File Header 12
4.1.2 TOC Segment 14
4.1.3 Data Segment 15
4.2 Data Segments 20

5 LSG Segment 21
5.1 Graph Elements 21
5.1.1 Node Elements 22
5.1.2 Attribute Elements 42
5.2 Property Atom Elements 85
5.2.1 Base Property Atom Element 85
5.2.2 String Property Atom Element 86
5.2.3 Integer Property Atom Element 86
5.2.4 Floating Point Property Atom Element 87
5.2.5 JT Object Reference Property Atom Element 88
5.2.6 Date Property Atom Element 88
5.2.7 Late Loaded Property Atom Element 90
5.2.8 Vector4f Property Atom Element 91
5.3 Property Table 92
5.3.1 Element Property Table 92

6 Shape LOD Segment 93
6.1 Shape LOD Element 93
6.1.1 Base Shape LOD Element 93
6.1.2 Vertex Shape LOD Element 94
6.1.3 Tri-Strip Set Shape LOD Element 108
6.1.4 Polyline Set Shape LOD Element 109
6.1.5 Point Set Shape LOD Element 110
6.1.6 Null Shape LOD Element 110
6.2 Primitive Set Shape Element 111
6.2.1 Lossless Compressed Primitive Set Data 113
6.2.2 Lossy Quantized Primitive Set Data 115

7 JT B-Rep Segment 118
7.1 JT B-Rep Element 119
7.1.1 Topological Entity Counts 121
7.1.2 Geometric Entity Counts 122
7.1.3 Topology Data 123
7.1.4 Geometric Data 131
7.1.5 Topological Entity Tag Counters 140
7.1.6 B-Rep CAD Tag Data 141

8 XT B-Rep Segment 141

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

8.1 XT B-Rep Element 141
8.1.1 XT B-Rep Data 142

9 Wireframe Segment 143
9.1 Wireframe Rep Element 143
9.1.1 Wireframe MCS Curves Geometric Data 145
9.1.2 Wireframe Rep CAD Tag Data 145

10 Meta Data Segment 146
10.1 Property Proxy Meta Data Element 146
10.1.1 Date Property Value 148
10.2 PMI Manager Meta Data Element 149
10.2.1 PMI Entities 152
10.2.2 PMI Associations 171
10.2.3 PMI User Attributes 173
10.2.4 PMI String Table 174
10.2.5 PMI Model Views 175
10.2.6 Generic PMI Entities 177
10.2.7 PMI CAD Tag Data 182
10.2.8 PMI Polygon Data 183

11 PMI Data Segment 186

12 JT ULP Segment 186
12.1 JT ULP Element 186
12.1.1 Topology Data 188
12.1.2 Geometric Data 204
12.1.3 Material Attribute Element Properties 226
12.1.4 Information Recovery 227
12.2 JT LWPA Segment 232
12.3 JT LWPA Element 232
12.3.1 Analytic Surface Geometry 233

13 Data Compression and Encoding 236
13.1 Common Compression Data Collection Formats 236
13.1.1 Int32 Compressed Data Packet 236
13.1.2 Int32 Compressed Data Packet Mk. 2 241
13.1.3 Float64 Compressed Data Packet 247
13.1.4 Compressed Vertex Coordinate Array 251
13.1.5 Compressed Vertex Normal Array 252
13.1.6 Compressed Vertex Texture Coordinate Array 254
13.1.7 Compressed Vertex Colour Array 256
13.1.8 Compressed Vertex Flag Array 258
13.1.9 Point Quantizer Data 259
13.1.10 Texture Quantizer Data 259
13.1.11 Colour Quantizer Data 260
13.1.12 Uniform Quantizer Data 262
13.1.13 Compressed Entity List for Non-Trivial Knot Vector 262
13.1.14 Compressed Control Point Weights Data 265
13.1.15 Compressed Curve Data 266
13.1.16 Compressed CAD Tag Data 269
13.2 Encoding Algorithms 272
13.2.1 Uniform Data Quantization 272
13.2.2 Bitlength CODEC 272
13.2.3 Arithmetic CODEC 273
13.2.4 Deering Normal CODEC 278
13.3 zlib compression 280

14 Best Practices 280
14.1 Late-Loading Data 280
14.2 TOC Segment Location 281
14.3 Bit Fields 281
14.4 Empty Field 281

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

 © ISO 2012 – All rights reserved

14.5 Local version numbers **281**
14.5.1 Version numbers **281**
14.6 Hash Value **283**
14.7 Scene graph construction **284**
14.8 Metadata Conventions **284**
14.8.1 CAD Properties **284**
14.8.2 PMI Properties **286**
14.8.3 Tessellation Properties **391**
14.8.4 Miscellaneous Properties **392**
14.9 LSG Attribute Accumulation Semantics **393**
14.10 LSG Part Structure **394**
14.11 Range LOD Node Alternative Rep Selection **394**
14.12 Brep Face Group Associations **394**

Annex A (informative) Object Type Identifiers **396**

Annex B (informative) Coding Algorithms – An Implementation **399**
B.1 Common classes **399**
B.1.1 CntxEntry class **399**
B.1.2 CntxEntryBase2 class **399**
B.1.3 ProbabilityContext class **400**
B.1.4 ProbContex2 class **401**
B.1.5 CodecDriver class **404**
B.1.6 CodecDriver2 class **407**
B.2 Bitlength decoding classes **410**
B.2.1 BitLengthCodec class **410**
B.2.2 BitLengthCodec2 class **415**
B.3 Arithmetic decoding classes **422**
B.3.1 ArithmeticProbabilityRange class **422**
B.3.2 ArithmeticCodec class **422**
B.3.3 ArithmeticCodec2 class **426**
B.4 Deering Normal decoding classes **430**
B.4.1 DeeringNormalLookupTable class **430**
B.4.2 DeeringNormalCodec class **432**

Annex C (informative) Hashing – An Implementation **435**

Annex D (informative) Polygon Mesh Topology Coder **438**
D.1 DualVFMesh **439**
D.2 Topology Decoder **445**
D.2.1 MeshCoderDriver class **445**
D.2.2 MeshCodec class **448**
D.2.3 MeshDecoder class **453**

Annex E (informative) **XT B- Rep data segment** **456**
E.1 Introduction to the XT B- Rep data segment **456**
E.2 Logical Layout **456**
E.2.1 Schema **457**
E.2.2 Embedded schemas **457**
E.2.3 Field types **458**
E.2.4 Variable-length nodes **459**
E.2.5 Unresolved indices **459**
E.3 Physical Layout **459**
E.3.1 Binary **459**
E.4 Model Structure **460**
E.4.1 Topology **460**
E.4.2 General points **461**
E.4.3 Entity definitions **461**
E.4.4 Entity matrix **465**
E.4.5 Representation of manifold bodies **465**
E.5 Schema Definition **467**
E.5.1 Underlying types **467**

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

E.5.2 Geometry 467
E.5.3 Topology 508
E.5.4 Associated Data 522
E.6 Node Types 535
E.7 Node Classes 537
E.8 System Attribute Definitions 538
E.8.1 Hatching Attributes 538
E.8.2 Density Attributes 539
E.8.3 Region 541
E.8.4 Colour 542
E.8.5 Reflectivity 542
E.8.6 Translucency 542
E.8.7 Name 543
E.8.8 Incremental faceting 543
E.8.9 Transparency 543
E.8.10 Non-mergeable edges 543
E.8.11 Group merge behaviour 544
E.8.12 Unicode name 544

Annex F (informative) XT data description – questions and answers 545
F.1 intersection curves 545
F.2 rolling ball blend surface 551
F.3 procedural geometry – general 555
F.4 bounded Geometry 555
F.5 geometry - general 556
F.6 face – surface connectivity 558

Bibliography 559

Figures

Figure 1 — rectangle box diagram 1
Figure 2 — folder diagram 1
Figure 3 — rectangle box with lines at left and right sides diagram 1
Figure 4 — rectangle box with clipped right side corners 1
Figure 5 — compressed data packet diagram 1
Figure 6 — data type : field name diagram 1
Figure 7 — data filed dependency example 1
Figure 8 — loop construct example 1
Figure 9 — loop construct with iterations example 1
Figure 10 — JT File Structure 12
Figure 11 — File Header data collection 13
Figure 12 — TOC Segment data collection 14
Figure 13 — TOC Entry data collection 15
Figure 14 — Data Segment data collection 16
Figure 15 — Segment Header data collection 16
Figure 16 — Data collection 18
Figure 17 — Logical Element Header data collection 18
Figure 18 — Element Header data collection 18
Figure 19 — Logical Element Header ZLIB data collection 19
Figure 20 — LSG Segment data collection 21
Figure 21 — Base Node Element data collection 22
Figure 22 — Base Node Data collection 22
Figure 23 — Partition Node Element data collection 24
Figure 24 — Vertex Count Range data collection 25
Figure 25 — Group Node Element data collection 26
Figure 26 — Group Node Data collection 26
Figure 27 — Instance Node Element data collection 27
Figure 28 — Part Node Element data collection 28
Figure 29 — Meta Data Node Element data collection 28
Figure 30 — Meta Data Node Data collection 29

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Figure 31 — LOD Node Element data collection 29
Figure 32 — LOD Node Data collection 30
Figure 33 — Range LOD Node Element data collection 31
Figure 34 — Switch Node Element data collection 32
Figure 35 — Base Shape Node Element data collection 33
Figure 36 — Base Shape Data collection 33
Figure 37 —Vertex Count Range data collection 35
Figure 38 — Vertex Shape Node Element data collection 36
Figure 39 — Vertex Shape Data collection 36
Figure 40 — Quantization Parameters data collection 37
Figure 41 — Tri-Strip Set Shape Node Element data collection 37
Figure 42 — Polyline Set Shape Node Element data collection 38
Figure 43 — Point Set Shape Node Element data collection 39
Figure 44 — Polygon Set Shape Node Element data collection 39
Figure 45 — NULL Shape Node Element data collection 40
Figure 46 — Primitive Set Shape Node Element data collection 41
Figure 47 — Primitive Set Quantization Parameters data collection 42
Figure 48 — Base Attribute Data collection 43
Figure 49 — Material Attribute Element data collection 45
Figure 50 — Texture Image Attribute Element data collection 48
Figure 51 — Texture Vers-1 Data collection 50
Figure 52 — Texture Environment data collection 52
Figure 53 — Texture Coord Generation Parameters data collection 55
Figure 54 — Inline Texture Image Data collection 56
Figure 55 — Image Format Description data collection 57
Figure 56 — Texture Vers-2 Data collection 60
Figure 57 — Texture Vers-3 Data collection 63
Figure 58 — Draw Style Attribute Element data collection 66
Figure 59 — Light Set Attribute Element data collection 67
Figure 60 — Infinite Light Attribute Element data collection 68
Figure 61 — Base Light Data collection 69
Figure 62 — Shadow Parameters data collection 70
Figure 63 — Point Light Attribute ElementPoint Light Attribute Element data collection 71
Figure 64 — Spread Angle value with respect to the light cone 1
Figure 65 — Attenuation Coefficients data collection 73
Figure 66 — Linestyle Attribute Element data collection 74
Figure 67 — Pointstyle Attribute Element data collection 75
Figure 68 — Geometric Transform Attribute Element data collection 76
Figure 69 — Shader Effects Attribute Element data collection 78
Figure 70 — Texture Coordinate Generator Attribute Element data collection 80
Figure 71 — Mapping Plane Element data collection 81
Figure 72 — Mapping Cylinder Element data collection 82
Figure 73 — Mapping Sphere Element data collection 83
Figure 74 — Mapping TriPlanar Element data collection 84
Figure 75 — Base Property Atom Element data collection 85
Figure 76 — Base Property Atom Data collection 85
Figure 77 — String Property Atom Element data collection 86
Figure 78 — Integer Property Atom Element data collection 87
Figure 79 — Floating Point Property Atom Element data collection 87
Figure 80 — JT Object Reference Property Atom Element data collection 88
Figure 81 — Date Property Atom Element data collection 89
Figure 82 — Late Loaded Property Atom Element data collection 90
Figure 83 — Vector4f Property Atom Element data collection 91
Figure 84 — Property Table data collection 92
Figure 85 — Element Property Table data collection 93
Figure 86 — Shape LOD Segment data collection 93
Figure 87 — Base Shape LOD Element data collection 94
Figure 88 — Base Shape LOD Data collection 94
Figure 89 — Vertex Shape LOD Element data collection 94
Figure 90 — Vertex Shape LOD Data collection 95

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Figure 91 — TopoMesh LOD Data collection 97
Figure 92 — TopoMesh LOD Data collection 97
Figure 93 — TopoMesh Topologically Compressed LOD Data collection 98
Figure 94 — Topologically Compressed Rep Data Collection 99
Figure 95 — Topologically Compressed Vertex Records data collection 102
Figure 96 — TopoMesh Compressed Rep Data V1 data collection 103
Figure 97 — TopoMesh Compressed Rep Data V2 data collection 106
Figure 98 — Tri-Strip Set Shape LOD Element data collection 109
Figure 99 — Polyline Set Shape LOD Element data collection 109
Figure 100 — Point Set Shape LOD Element data collection 110
Figure 101 — Null Shape LOD Element data collection 111
Figure 102 — Primitive Set Shape Element data collection 112
Figure 103 — Lossless Compressed Primitive Set Data collection 113
Figure 104 — Lossy Quantized Primitive Set Data collection 115
Figure 105 — Compressed params1 data collection 117
Figure 106 — JT B-Rep Segment data collection 119
Figure 107 — JT B-Rep Element data collection 120
Figure 108 — Topological Entity Counts data collection 121
Figure 109 — Geometric Entity Counts data collection 122
Figure 110 — Topology Data collection 123
Figure 111 — Regions Topology Data collection 124
Figure 112 — Shells Topology Data collection 124
Figure 113 — Trim Loop example in parameter Space - One Face with 2 Holes 126
Figure 114 — Faces Topology Data collection 126
Figure 115 — Loops Topology Data collection 127
Figure 116 — CoEdges Topology Data collection 128
Figure 117 — Edges Topology Data collection 129
Figure 118 — Vertices Topology Data collection 130
Figure 119 — Geometric Data collection 131
Figure 120 — Surfaces Geometric Data collection 132
Figure 121 — Non-Trivial Knot Vector NURBS Surface Indices data collection 133
Figure 122 — NURBS Surface Degree data collection 134
Figure 123 — NURBS Surface Control Point Counts data collection 134
Figure 124 — NURBS Surface Control Point Weights data collection 134
Figure 125 — NURBS Surface Control Points data collection 135
Figure 126 — NURBS Surface Knot Vectors data collection 135
Figure 127 — PCS Curves Geometric Data collection 136
Figure 128 — Trivial PCS Curves data collection 137
Figure 129 — Equality of corresponding curve end coordinates of opposite sides of the box 138
Figure 130 — MCS Curves Geometric Data collection 139
Figure 131 — Point Geometric Data collection 140
Figure 132 — Topological Entity Tag Counters data collection 140
Figure 133 — B-Rep CAD Tag Data collection 141
Figure 134 — XT B-Rep Element data collection 142
Figure 135 — Wireframe Segment data collection 143
Figure 136 — Wireframe Rep Element data collection 144
Figure 137 — Wireframe MCS Curves Geometric Data collection 145
Figure 138 — Wireframe Rep CAD Tag Data collection 145
Figure 139 — Meta Data Segment data collection 146
Figure 140 — Meta Data Segment data collection 147
Figure 141 — Date Property Value data collection 148
Figure 142 — PMI Manager Meta Data Element data collection 150
Figure 143 — PMI Entities data collection 152
Figure 144 — PMI Dimension Entities data collection 152
Figure 145 — PMI 2D Data collection 153
Figure 146 — PMI Base Data collection 153
Figure 147 — 2D-Reference Frame data collection 154
Figure 148 — 2D Text Data collection 155
Figure 149 — Text Box data collection 156
Figure 150 — Constructing Text Polylines from data arrays 157

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Figure 151 — Text Polyline Data collection 157
Figure 152 — Constructing Non-Text Polylines from packed 2D data arrays 158
Figure 153 — Non-Text Polyline Data collection 159
Figure 154 — PMI Note Entities data collection 160
Figure 155 — PMI Datum Feature Symbol Entities data collection 161
Figure 156 — PMI Datum Target Entities data collection 161
Figure 157 — PMI Feature Control Frame Entities data collection 162
Figure 158 — PMI Line Weld Entities data collection 162
Figure 159 — PMI Spot Weld Entities data collection 163
Figure 160 — PMI 3D Data collection 164
Figure 161 — PMI Surface Finish Entities data collection 165
Figure 162 — PMI Measurement Point Entities data collection 166
Figure 163 — PMI Locator Entities data collection 167
Figure 164 — PMI Reference Geometry Entities data collection 167
Figure 165 — PMI Design Group Entities data collection 168
Figure 166 — Design Group Attribute data collection 169
Figure 167 — PMI Coordinate System Entities data collection 170
Figure 168 — PMI Associations data collection 171
Figure 169 — PMI User Attributes data collection 174
Figure 170 — PMI String Table data collection 174
Figure 171 — PMI Model Views data collection 175
Figure 172 — Generic PMI Entities data collection 177
Figure 173 — PMI Property data collection 179
Figure 174 — PMI Property Atom data collection 181
Figure 175 — PMI CAD Tag Data collection 182
Figure 176 — PMI Polygon Data 184
Figure 177 — JT ULP Segment data collection 186
Figure 178 — JT ULP Element data collection 187
Figure 179 — Topology Data collection 188
Figure 180 — Topological Entity Counts data collection 189
Figure 181 — Combined Predictor Type data collection 190
Figure 182 — Regions Topology Data collection 191
Figure 183 — Shells Topology Data collection 192
Figure 184 — Faces Topology Data collection 193
Figure 185 — Loops Topology Data collection 196
Figure 186 — CoEdges Topology Data collection 198
Figure 187 — Surface Domain Classification 199
Figure 188 — Edges Topology Data collection 201
Figure 189 — Vertices Topology Data collection 203
Figure 190 — Geometric Data collection 204
Figure 191 — Geometric Entity Counts 205
Figure 192 — Degree Table data collection 206
Figure 193 — Recover Nurbs Degree 207
Figure 194 — Number of Control Points Table data collection 208
Figure 195 — Recover Number of Control Points 209
Figure 196 — Dimension Table data collection 210
Figure 197 — Recover Dimension 211
Figure 198 — 3D Unit Vector Table data collection 212
Figure 199 — Recover Dimension 213
Figure 200 — 2D Unit Vector Table data collection 214
Figure 201 — Recover 2D Unit Vector 214
Figure 202 — 3D MCS Point Table data collection 215
Figure 203 — Recover 3D MCS Points 216
Figure 204 — Knot Vector Table data collection 217
Figure 205 — Recover Knot Vectors 218
Figure 206 — 1D MCS Table data collection 220
Figure 207 — Recover 1D MCS Table 221
Figure 208 — PCS Value Table data collection 222
Figure 209 — Recover PCS Value Table 223
Figure 210 — Radian Table data collection 223

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Figure 211 — Recover Radian Table ... 224
Figure 212 — Weight Table data collection ... 225
Figure 213 — Recover Weight Table ... 226
Figure 214 — Material Attribute Element Properties ... 227
Figure 215 — Material Attribute Element Properties ... 228
Figure 216 — PCS Curve Recovery from Surface Domain ... 229
Figure 217 — MCS Curve Recovery ... 230
Figure 218 — MCS Curve Recovery from Surface Geometry ... 231
Figure 219 — PCS Curve Recovery from MCS Curve and Surface Geometry ... 232
Figure 220 — JT LWPA Segment data collection ... 232
Figure 221 — JT LWPA Element data collection ... 233
Figure 222 — Analytic Surface Geometry data collection ... 234
Figure 223 — Analytic Surface Creation ... 235
Figure 224 — Int32 Compressed Data Packet data collection ... 237
Figure 225 — Int32 Probability Contexts data collection ... 239
Figure 226 — Int32 Probability Context Table Entry data collection ... 240
Figure 227 — Int32 Compressed Data Packet Mk. 2 data collection ... 243
Figure 228 — Int32 Probability Contexts Mk. 2 data collection ... 245
Figure 229 — Int32 Probability Context Table Entry Mk. 2 data collection ... 246
Figure 230 — Float64 Compressed Data Packet data collection ... 248
Figure 231 — Float64 Probability Contexts data collection ... 250
Figure 232 — Float64 Probability Context Table Entry data collection ... 250
Figure 233 — Compressed Vertex Coordinate Array data collection ... 251
Figure 234 — Compressed Vertex Normal Array data collection ... 253
Figure 235 — Compressed Vertex Texture Coordinate Array data collection ... 255
Figure 236 — Compressed Vertex Colour Array data collection ... 257
Figure 237 — Compressed Vertex Flag Array data collection ... 259
Figure 238 — Point Quantizer Data collection ... 259
Figure 239 — Texture Quantizer Data collection ... 260
Figure 240 — Colour Quantizer Data collection ... 261
Figure 241 — Uniform Quantizer Data collection ... 262
Figure 242 — Compressed Entity List for Non-Trivial Knot Vector data collection ... 263
Figure 243 — Compressed Control Point Weights Data collection ... 265
Figure 244 — Compressed Curve Data collection ... 267
Figure 245 — Non-Trivial Knot Vector NURBS Curve Indices data collection ... 269
Figure 246 — NURBS Curve Control Point Weights data collection ... 269
Figure 247 — NURBS Curve Control Points data collection ... 269
Figure 248 — Compressed CAD Tag Data collection ... 270
Figure 249 — Compressed CAD Tag Type-2 Data collection ... 271
Figure 250 — Sextant Coding on the Sphere ... 279
Figure 251 — JT Format Convention for Modeling each Part in LSG ... 394

Tables

Table 1 — Symbols ... 5
Table 2 — Predictor Type ... 7
Table 3 — Basic Data Types ... 9
Table 4 — Composite Data Types ... 10
Table 5 — Segment attributes ... 15
Table 6 — Segment Types ... 16
Table 7 — Object Base Types ... 19
Table 8 — Compression flag values ... 20
Table 9 — Compression algorithm values ... 20
Table 10 — Node Flag values ... 23
Table 11 — Partition flag bits ... 24
Table 12 — Compression level values ... 34
Table 13 — Texture Coord Binding values ... 41
Table 14 — Version Number values ... 42
Table 15 — Texture Coord Gen Type values ... 42
Table 16 — State Flag values ... 44
Table 17 — Material Attribute data field inihibit values ... 44

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 18 — Material Attribute Version number value 46
Table 19 — Material Attribute Data Flag values 46
Table 20 — Texture Image Attribute data field inihibit values 48
Table 21 — Texture Image Version Number values 49
Table 22 — Texture Vers-1 type values 51
Table 23 — Texture Vers-1 Inline Image Storage Flag vaules 51
Table 24 — Texture Vers-1 Texture Environment Border Mode values 52
Table 25 — Texture Vers-1 Texture Environment Mipmap Magnification Filter values 53
Table 26 — Texture Vers-1 Texture Environment Mipmap Minification Filter values 53
Table 27 — Texture Vers-1 Texture Environment S-Dimen Wrap Mode values 53
Table 28 — Texture Vers-1 Texture Environment Blend Type values 54
Table 29 — Texture Vers-1 Texture Environment Internal Compression Level values 54
Table 30 — Texture Vers-1 Texture Coord Generation Gen Mode values 55
Table 31 — Texture Vers-1 Image Format Description Pixel Format values 58
Table 32 — Texture Vers-1 Image Format Description Pixel Data values 58
Table 33 — Texture Vers-1 Image Format Description Dimensionality values 58
Table 34 — Texture Vers-1 Image Format Description Shared Image Flag values 59
Table 35 — Texture Vers-2 Texture Type values 61
Table 36 — Texture Vers-2 Inline Image Storage Flag values 62
Table 37 — Texture Vers-3 Texture Type values 64
Table 38 — Texture Vers-3 Inline Image Storage Flag values 65
Table 39 — Draw Style Attribute Field Inhibit flag values 65
Table 40 — Draw Style Attribute Data Flag values 66
Table 41 — Light Set Attribute Version Number values 69
Table 42 — Base Light Data Cood System values 70
Table 43 — Base Light Data Shadow Caster Flag values 70
Table 44 — Point Light Attribute Version Number values 72
Table 45 — Point Light Attribute Spread Angle values 72
Table 46 — Linestyle Attribute Data Flag values 74
Table 47 — Pointstyle Attribute Data Flag values 75
Table 48 — Geometric Transform Attribute Stored Value Mask individual bit-flag values 77
Table 49 — Shader Effects Attribute Enable Flag values 78
Table 50 — Shader Effects Attribute Phong Shading Flag values 79
Table 51 — Mapping Plane Matrix Coordinate System values 81
Table 52 — Mapping Cylinder Matrix Coordinate System values 82
Table 53 — Mapping Sphere Matrix Coordinate System values 83
Table 54 — Mapping TriPlanar Matrix Coordinate System values 84
Table 55 — Vertex Shape LOD Bindings values 95
Table 56 — TopoMesh Compressed Rep Data V2 Field Type values 107
Table 57 — Primitive Set Shape Texture Coord Binding values 112
Table 58 — Primitive Set Shape Version Number values 113
Table 59 — Primitive Set Shape Texture Coord Gen Type values 113
Table 60 — Lossless Compressed Primitive Set Data Field values 114
Table 61 — Primitive Set “params#” Data Fields Interpretation 114
Table 62 — JT B-Rep Shell Topology Anti-Hole Flag values 125
Table 63 — JT B-Rep Face Reverse Normal Flag values 127
Table 64 — JT B-Rep Loops Topology Data Anti-Hole Flag values 128
Table 65 — JT B-Rep MCS Curve Reversed Flag values 129
Table 66 — JT B-Rep Surface Base Type value 132
Table 67 — JT B-Rep NURBS Surface Control Point Demensionality values 133
Table 68 — Trivial Domain Loops Exist Flag values 137
Table 69 — Trivial Box Loops Exist Flag values 138
Table 70 — Trivial Domain UV Curves Exist Flag values 138
Table 71 — Trivial UV Curve Para Domain Side Codes values 139
Table 72 — Property Proxy Meta Data Property Value Type values 148
Table 73 — PMI Manager Meta Data Version Number values 151
Table 74 — PMI 2D Base Data Font values 155
Table 75 — PMI 2D Non-Text Polyline Type values 159
Table 76 — PMI 3D Data Polyline Dimensionality values 164
Table 77 — PMI Reference Geometry Entity values 167

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 78 — PMI Design Group Attribute Type values 169
Table 79 — PMI Associations Source Data values 172
Table 80 — PMI Associations Reason Code values 173
Table 81 — PMI Model Views Active Flag values 176
Table 82 — Generic PMI Entity Type values 178
Table 83 — Generic PMI User Flag values 179
Table 84 — Common Property Keys and Their Value Encoding formats 180
Table 85 — PMI Property Atom Hidden Flag values 181
Table 86 — JT ULP Shell Anti-Hole Flag values 192
Table 87 — JT ULP Flag Bit Array Look Index values 194
Table 88 — JT ULP Supported Surface Type values 194
Table 89 — JT ULP Supported Knot Type Values 195
Table 90 — JT ULP Face Reverse Normal Flag values 195
Table 91 — JT ULP Loops Topology Flag Bit Array values 196
Table 92 — JT ULP Loops Topology Reverse Normal Flag values 197
Table 93 — JT ULP Recover Edge Indices Flag Bit Array values 199
Table 94 — JT ULP Recover Edge Indices PCS curve type values 200
Table 95 — JT ULP PCS Curve Type values 200
Table 96 — JT ULP PCS Curve Type XYZ Reversed Flag values 200
Table 97 — JT ULP PCS Curve Type isUVInc Flag values 200
Table 98 — JT ULP Edges Topology Recover MCS Curve Indices Flag Bit Array values 202
Table 99 — JT ULP Edges Topology Recover MCS Curve Type values 202
Table 100 — Parameter Domain 220
Table 101 — Common Compression CODEC Type values 238
Table 102 — Int32 Probability Contexts CODEC Type values 244
Table 103 — Float64 Compressed Data Packet CODEC Type values 248
Table 104 — Colour Quantizer values 260
Table 105 — Colour Quantizer HSV Flag values 261
Table 106 — Knot Type Exist Flag values 264
Table 107 — Compressed Curve Base Type values 267
Table 108 — NURB UV Curve entity dimensionality values 268
Table 109 — NURB XYZ Curve entity dimensionality values 268
Table 110 — Compressed CAD Tag Type values 271
Table 111 — Example assigned probability values 275
Table 112 — Example “probability line” values 275
Table 113 — Example input integer sequence values 275
Table 114 — Example integer number sequence values 277
Table 115 — CAD Property Conventions 285
Table 116 — CAD Optional Property Units 286
Table 117 — PMI Properties 287
Table 118 —Tessellatin Proptery values 392
Table 119 — Miscellaneous Proptery values 392
Table A.1 — Object Type Identifiers 396
Table F.1 — Object Nodes 457
Table F.2 — Field types in order one by one 458
Table F.3 — Entity Matrix relations 465
Table F.4 — Curve node common fields 469
Table F.5 — Line Fields 470
Table F.6 — Circle fields 471
Table F.7 — Ellipse fields 472
Table F.8 — NURB curve fields 475
Table F.9 — Curve intersection fields 479
Table F.10 — Trimmed curve fields 483
Table F.11 — Foreign Geometry curve fields 484
Table F.12 — SP curve fields 485
Table F.13 — Surface node fields 486
Table F.14 — Plane fields 487
Table F.15 — Cylinder fields 488
Table F.16 — Cone fields 490
Table F.17 — Sphere fields 491

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table F.18 — Torus fields 492
Table F.19 — Blended edge fields 493
Table F.20 — Blend boundary surface fields 495
Table F.21 — Offset surface fields 496
Table F.22 — B-Surface fields 497
Table F.23 — NURB Surface fields 498
Table F.24 — Swept surface fields 502
Table F.25 — Spun surface fields 503
Table F.26 — Foreign Geometry suface fields 504
Table F.27 — Point fields 505
Table F.28 — Transform fields 506
Table F.29 — Transform action fields 506
Table F.30 — Geometry owner fields 508
Table F.31 — World topology fields 508
Table F.32 — Assembly fields 509
Table F.33 — Instance fields 511
Table F.34 — Body fields 512
Table F.35 — Geometry to Topology attachment 515
Table F.36 — Region fields 516
Table F.37 — Shell fields 517
Table F.38 — Face fields 518
Table F.39 — Loop fields 519
Table F.40 — Fin fields 519
Table F.41 — Vertex fields 520
Table F.42 — Edge fields 521
Table F.43 — Associated List 522
Table F.44 — Pointer List Block 524
Table F.45 — Attribute Definition ID 524
Table F.46 — Field Names 524
Table F.47 — Attribut definition 525
Table F.48 — Attribute definition action fields 526
Table F.49 — Corresponding attribute classes 526
Table F.50 — Attribute fields 529
Table F.51 — Integer values 531
Table F.52 — Real values 531
Table F.53 — Character values 531
Table F.54 — Unicode values 531
Table F.55 — Point values 532
Table F.56 — Vector values 532
Table F.57 — Direction values 532
Table F.58 — Axis values 532
Table F.59 — Tag values 533
Table F.60 — Group fields 533
Table F.61 — Group member fields 534
Table F.62 — Node types 535
Table F.63 — Node classes 537
Table F.64 — Hatching 538
Table F.65 — Planar Hatch 538
Table F.66 — Radial Hatch 539
Table F.67 — Parametric Hatch 539
Table F.68 — Body Density 540
Table F.69 — Region Density 540
Table F.70 — Face Density 540
Table F.71 — Edge Density 541
Table F.72 — Vertex Density 541
Table F.73 — Region 541
Table F.74 — Colour 542
Table F.75 — Reflectivity 542
Table F.76 — Translucency 542
Table F.77 — Name 543

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table F.78 — Incremental faceting **543**
Table F.79 — Transparancy **543**
Table F.80 — Non-mergable edges **543**
Table F.81 — Group merge behaviour **544**
Table F.82 — Unicode name **544**

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

ISO 14306 was prepared by Technical Committee ISO TC 184/SC 4, *Automation systems and integration*, Subcommittee SC 4 *Industrial data*.

This first edition of ISO 14306 cancels and replaces ISO/PAS 14306:2011, which has been technically revised.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Introduction

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.

The JT format is both robust, and streamable, and contains best-in-class compression for compact and efficient representation. The 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.

Some of the highlights of the JT format include:

— built-in support for assemblies, sub-assemblies and part constructs;

— a flexible partitioning scheme, supporting single or multiple files;

— b-rep solid shape representations;

— product manufacturing information in support of paperless manufacturing initiatives;

— precise and imprecise wireframe shape representations;

— discrete purpose-built levels of detail;

— triangle sets, polygon sets, point sets, line sets and implicit primitive sets (such as cylinder, cone and sphere);

— a full array of visual attributes such as for materials, textures, lights, shaders;

— hierarchical bounding box and bounding spheres;

— 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, i.e. file granularity and flexibility optimized to meet the needs of enterprise data translation solutions;

— asynchronous streaming of content, i.e. viewing optimizations such as view frustum and occlusion culling and fixed-framerate display modes;

— layers, and layer filters.

The International Organization for Standardization (ISO) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents USA 20110199382, 8019788.

ISO takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO that he is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO. Information may be obtained from:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Siemens PLM Software Product Development
345 Woodcliff Drive
Fairport
NY 14450
USA

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) maintains an on-line database of patents relevant to its standards. Users are encouraged to consult the database for the most up to date information concerning patents.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

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

1 Scope

This International Standard defines the syntax and semantics of the JT Version 9.5 file format. At its core, the JT format is a scene graph with Computer Aided Design (CAD) specific node and attributes support.

The following are within scope of this International Standard:

— facet information (triangles), stored with geometry compression techniques;

— visual attributes such as lights, textures and materials ;

— product manufacturing information (PMI);

— boundary representation (b-rep) solid model shape representations and associated metadata;

— configuration representations;

— delivery methods such as asynchronous streaming of content.

The following are outside the scope of this International Standard:

— the implementation of, or definition of a run-time architecture for viewing and/or processing JT data;

— long term data retention;

— non-faceted geometric data exchange.

NOTE 1 Along with the pure syntactical definition of the format, there is also a series of conventions that, although not required to have a reference compliant JT file, have become commonplace within JT format translators. These conventions have been documented in Clause 14.

NOTE 2 Procedural shape information is included with this description for completeness. Due to the complexities of construction and interpretation of this type of data, it is advisable that more common geometry representations such as non-uniform rational b-splines (NURBS) be used in place of procedural shape information.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

2 Terms, definitions and abbreviated terms

2.1 Terms and definitions

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

2.1.1
assembly
related collection of model parts, represented in a JT format logical scene graph as a logical graph branch

2.1.2
attribute
objects associated with nodes in a logical scene graph and specifying one of several appearances, positioning, or rendering characteristics of a shape

© ISO 2012 – All rights reserved

2.1.3
boundary representation
solid model representation where the solid volume is specified by its surface boundary (both its geometric and topological boundaries)

2.1.4
code text
collection of data in encoded form

2.1.5
coordinate system
system which uses one or more numbers, or coordinates, to uniquely determine the position of a point or other geometric element

NOTE If not otherwise specified in a data field's description, it is assumed that the data is defined in local coordinate system.

2.1.6
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.

2.1.7
JT enabled application
application which supports reading and/or writing reference compliant JT format files

2.1.8
level of detail
LOD
alternative graphical representation for some model component such as part

2.1.9
local coordinate system
LCS
coordinate system that is used to specify the raw data of the shape geometry with no transforms applied

2.1.10
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.

2.1.11
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

2.1.12
model
representation, in JT format, of a physical or virtual product, part, assembly; or collections of such objects

2.1.13
model coordinate system
MCS
local coordinates transformed by any transforms specified as attributes at or above the node

2.1.14
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).

2.1.15
property
object associated with a logical scene graph node and identifying arbitrary application or enterprise specific information (meta-data) related to that node

2.1.16
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

2.1.17
scene graph
directed acyclic graph that arranges the logical and often (but not necessarily) spatial representation of a graphical scene

2.1.18
shader
user-definable program, expressed directly in a target assembly language or in high-level form to be compiled, that calculates colour values at a pixel based upon data such as lighting, surface colour and texture

NOTE 1 A shader program replaces a portion of the otherwise fixed-functionality graphics pipeline with some user-defined function.

NOTE 2 At present, hardware manufacturers have made it possible to run a shader for each vertex that is processed or each pixel that is rendered.

2.1.19
streaming
loading from disk based medium only the portions of data that are required by the user to perform the tasks at hand

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

NOTE 1 The motivation for sreaming 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 sreaming 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.

2.1.20
shape
logical scene graph leaf node containing or referencing the geometric shape definition data (such as vertices, polygons and normals) of a model component

2.1.21
texture channel
texture unit plus the texture environment

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

2.1.22
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].

2.1.23
texture unit
piece of hardware that takes a sample of a texture

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

2.1.24
view coordinate system
world coordinates transformed by a view matrix

2.1.25
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)

2.2 Abbreviated terms

Abs	Absolute Value
Bbox	Bounding Box
B-Rep	Boundary Representation
CAD	Computer Aided Design
CODEC	Coder-Decoder
GD&T	Geometric Dimensioning and Tolerancing

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

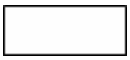
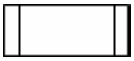
GUID	Globally Unique Identifier
HSV	Hue, Saturation, Value
LsbFirst	Least Significant Byte First
Max	Maximum
Min	Minimum
MsbFirst	Most Significant Byte First
N/A	Not Applicable
PCS	Parameter Coordinate Space
PLM	Product Lifecycle Management
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha
TOC	Table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

3 Notational conventions

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 stepsthat has been compressed.
	Rectangles with the right side corners clipped off represent information that has been compressed.
	Arrows convey the ordering of the information.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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.”

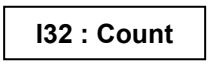


Figure 1 — rectangle box diagram

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”.

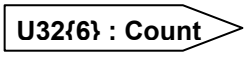


Figure 4 — rectangle box with clipped right side corners

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 the format used to encode/compress a collection of data into a series of Int32 based symbols. A complete description for Int32 Compressed Data Packet can be found in 13.1.1 Int32 Compressed Data Packet.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Int32CDP2 – The Int32CDP2 (i.e.Int32 Compressed Data Packet Mk. 2) represents a second-generation version of the above compressed data packet, and sports a simplified and more compact file layout, and the ability to more efficiently encode clustered data and bitfields. A complete description for Int32 Compressed Data Packet Mk. 2 can be found in 13.1.2 Int32 Compressed Data Packet Mk. 2.

Float64CDP – The Float64CDP (i.e. Float64 Compressed Data Packet) represents the format used to encode/compress a collection of data into a series of Float64 based symbols. A complete description for Float64 Compressed Data Packet can be found in 13.1.3 Float64 Compressed Data Packet.

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

In the example 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.”

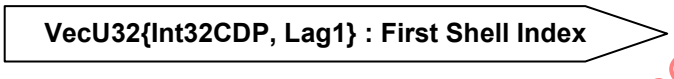


Figure 5 — compressed data packet diagram

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 or Float64CDP specifies, using the above described notation format, what Predictor Type algorithm is being used on the data. The JT format supported Predictor Type algorithms are as follows (note that a sample implementation of decoding the predictor *residual* values back into the primal values can be found in Annex C.

Table 2 — Predictor Type

Predictor Type	Description
Lag1	Predicts as last value
Lag2	Predicts as value before last
Stride1	Predicts using stride from last two values
Stride2	Predicts using stride from values 2 and 4 back

StripIndex	This is a completely empirical predictor. Looks at the values two back and four back in the stream, and uses the stride between these two values to predict the current value if and only if the stride lays between -8 and 8 *noninclusive*, else it predicts the value as the one two back plus two. In pseudo-code form the predicted values is computed as follows:
	`if(val2back - val4back < 8 && val2back - val4back > -8)`
	`  iPredicted = val2back + (val2back - val4back);`
	`else`
	`  iPredicted = val2back + 2;`

Ramp	Predict value “i” as values “i’s” index
Xor1	Predict as last, but use XOR instead of subtract to compute residual
Xor2	Predict as value before last, but use XOR instead of subtract to compute residual
NULL	No prediction applied

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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.”

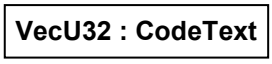


Figure 6 — data type : field name diagram

Note that for some JT file Segment Types there is ZLIB compression also applied to all bytes of element data stored in the segment. This ZLIB compression applied to all the segment’s data is not indicated in the diagrams through the use of “rectangle box with clipped right side corners”. Instead, one shall examine information stored with the first Element in the file segment to determine if ZLIB compression is applied to all data in the segment. A complete description of the JT format data compression and encoding can be found in 4.1.3 Data Segment and 13 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.

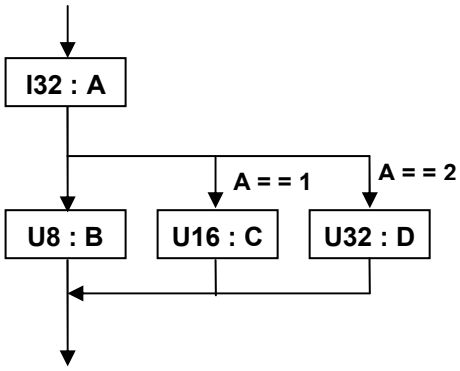


Figure 7 — data filed dependency example

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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:

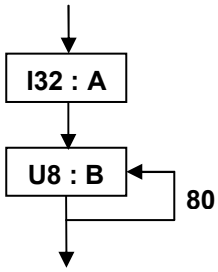


Figure 8 — loop construct example

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

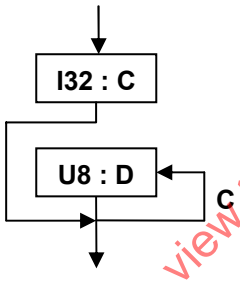


Figure 9 — loop construct with iterations example

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

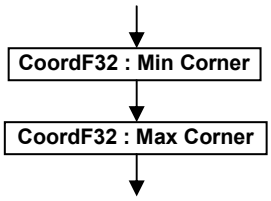
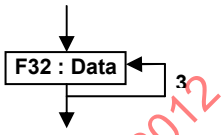
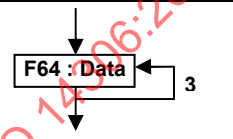
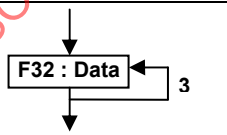
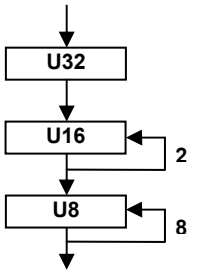
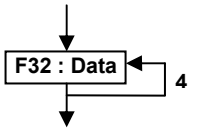
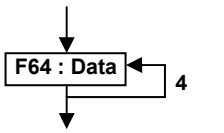
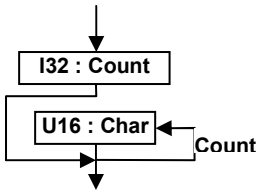
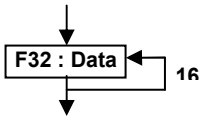
Type	Description
UChar	An unsigned 8-bit byte.
U8	An unsigned 8-bit integer value.
U16	An unsigned 16-bit integer value.
U32	An unsigned 32-bit integer value.
U64	An unsigned 64-bit integer value.
I16	A signed two's complement 16-bit integer value.
I32	A signed two's complement 32-bit integer value.
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

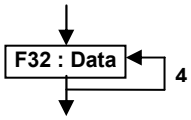
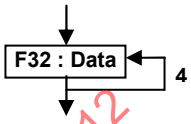
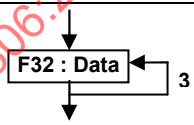
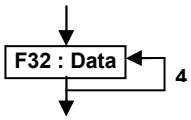
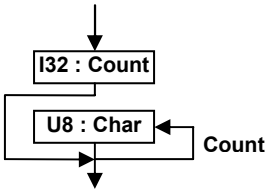
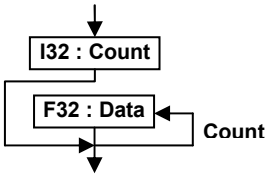
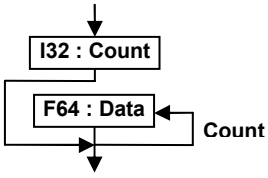
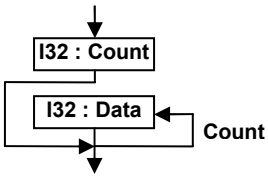
© ISO 2012 – All rights reserved

Table 4 — Composite Data Types

Type	Description	Symbolic Diagram
BBoxF32	The BBoxF32 type defines a bounding box using two CoordF32 types to store the XYZ coordinates for the bounding box minimum and maximum corner points.	
CoordF32	The CoordF32 type defines X, Y, Z coordinate values. So a CoordF32 is made up of three F32 base types.	
CoordF64	The CoordF64 type defines X, Y, Z coordinate values. So a CoordF64 is made up of three F64 base types.	
DirF32	The DirF32 type defines X, Y, Z components of a direction vector. So a DirF32 is made up of three F32 base types.	
GUID	The GUID type is a 16 byte (128-bit) number. GUID is stored/written to the JT file using a four-byte word (U32), 2 two-byte words (U16), and 8 one-byte words (U8) such as: {3F2504E0-4F89-11D3-9A-0C-03-05-E8-2C-33-01} In the JT format GUIDs are used as unique identifiers (e.g. Data Segment ID, Object Type ID, etc.)	
HCoordF32	The HCoordF32 type defines X, Y, Z, W homogeneous coordinate values. So an HCoordF32 is made up of four F32 base types.	
HCoordF64	The HCoordF64 type defines X, Y, Z, W homogeneous coordinate values. So an HCoordF64 is made up of four F64 base types	
MbString	The MbString type starts with an I32 that defines the number of characters (NumChar) the string contains. The number of 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.	

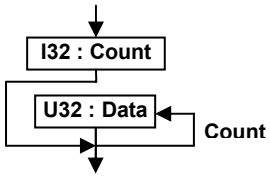
STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

PlaneF32	The PlaneF32 type defines a geometric Plane using the General Form of the plane equation (Ax + By + Cz + D = 0). The PlaneF32 type is made up of four F32 base types where the first three F32 define the plane unit normal vector (A, B, C) and the last F32 defines the negated perpendicular distance (D), along normal vector, from the origin to the plane.	
Quaternion	The Quaternion type defines a 3-dimensional orientation (no translation) in quaternion linear combination form (a + b*i* + c*j* + d*k*) where the four scalar values (a, b, c, d) are associated with the 4 dimensions of a quaternion (1 real dimension, and 3 imaginary dimensions). So the Quaternion type is made up of 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.	
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.	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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.	

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 best practice 14.4 Empty Field

4 File Format

All objects represented in the JT format are assigned an “object identifier” (e.g. see 5.1.1.1.1 Base Node Data, or 5.1.2.1.1 Base Attribute Data) and all references from one object to another object are represented in the JT format using the referenced 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).

4.1 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 5 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.

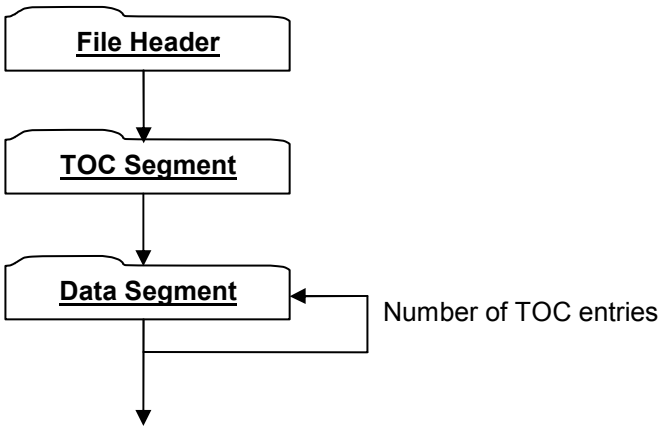


Figure 10 — JT File Structure

4.1.1 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:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

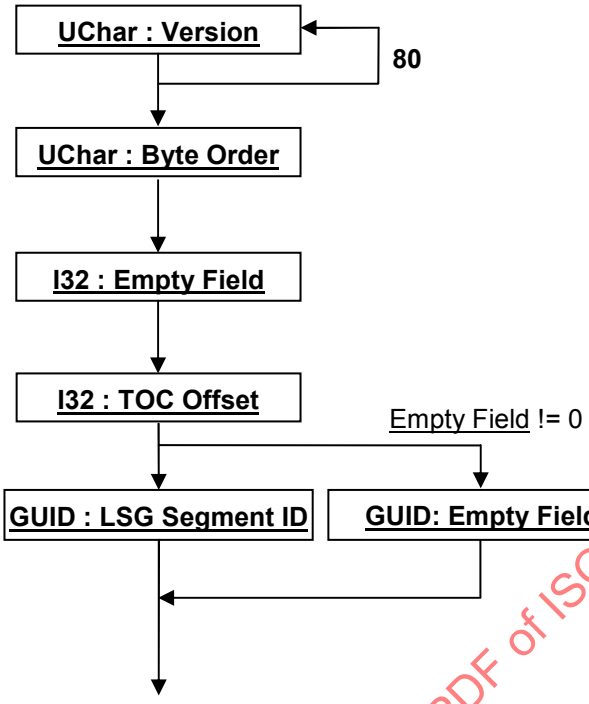


Figure 11 — File Header data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

UChar : Version

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

Version M.n Comment

Where ***M*** is replaced by the major version number, ***n*** is replaced by the minor version number, and ***Comment*** provides other information.

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

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

These final 5 characters (shown above and referred to as ASCII/binary translation detection bytes) can be used by JT file readers to validate that the JT files has not been corrupted by ASCII mode FTP transfers.

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

“Version 9.5 JT DM 7.3.4.0 \n\r\n “

UChar : Byte Order

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

© ISO 2012 – All rights reserved

0 – Least Significant byte first (LsbFirst)

1 – Most Significant byte first (MsbFirst)

I32 : Empty Field

Refer to best practice 14.4 Empty Field.

I32 : TOC Offset

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

GUID : LSG Segment ID

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

GUID: Empty Field

Refer to best practice 14.4 Empty Field

4.1.2 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.

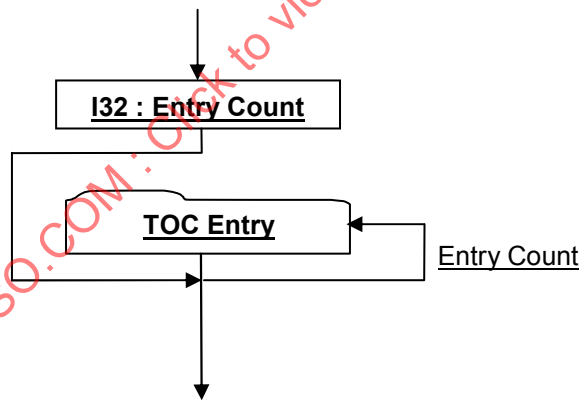


Figure 12 — TOC Segment data collection

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

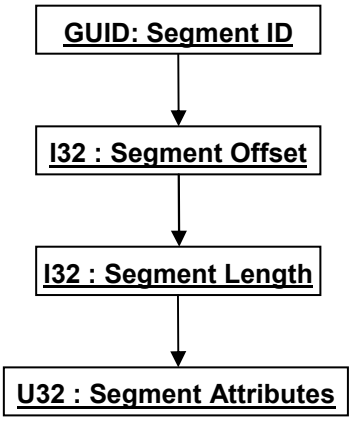


Figure 13 — TOC Entry data collection

GUID : Segment ID

Segment ID is the globally unique identifier for the segment.

I32 : Segment Offset

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

I32 : Segment Length

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

U32 : Segment Attributes

Segment Attributes 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.

4.1.3 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 a ZLIB compression conditionally applied to all the Data bytes of information persisted within the segment. Whether ZLIB compression is conditionally applied to a segment’s Data bytes of information is indicated by information stored with the first “Element” in the segment. Also Table 6 — Segment Types has a column indicating whether the Segment Type may have ZLIB compression applied to its Data bytes.

All Data Segments have the same basic structure.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

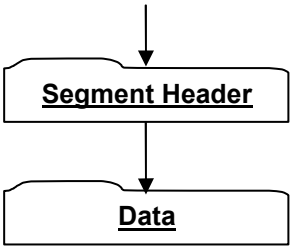


Figure 14 — Data Segment data collection

4.1.3.1 Segment Header

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

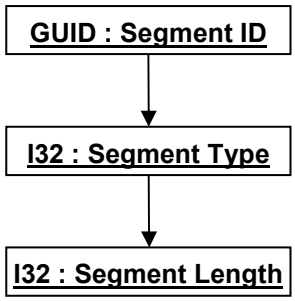


Figure 15 — Segment Header data collection

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

Table 6 — Segment Types

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Type	Data Contents	ZLIB Applied?
15	Shape LOD8	No
16	Shape LOD9	No
17	XT B-Rep	Yes
18	Wireframe Representation	Yes
20	ULP	Yes
24	LWPA	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.

I32 : Segment Length

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

4.1.3.2 Data

The interpretation of the Data section depends on the Segment Type. See 4.2 Data Segments for complete description for all Data Segment that may be contained in a JT file.

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

Individual data fields of an Element data collection (and its children data collections) may have advanced compression/encoding applied to them as indicated through compression related data values stored as part of the particular Element's storage format. In addition, another level of compression (i.e. ZLIB compression) may be conditionally applied to all bytes of information stored for all Elements within a particular Segment. Not all Segment types support ZLIB compression on all Segment data as indicated Table 8 — Segment Types If a particular file Segment is of the type which supports ZLIB compression on all the Segment data, whether this compression is applied or not is indicated by data values stored in the Logical Element Header ZLIB data collection of the first Element within the Segment. An in-depth description of JT file compression/encoding techniques can be found in 13 Data Compression and Encoding.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

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

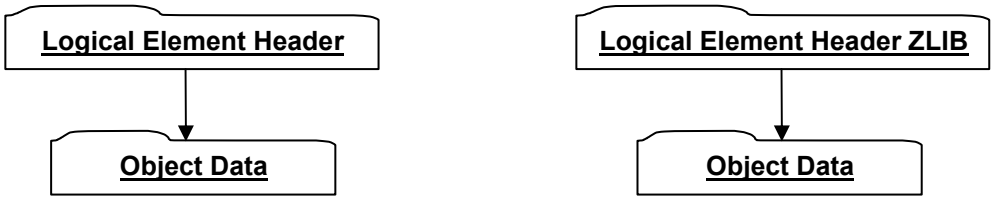


Figure 16 — Data collection

Logical Element Header

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

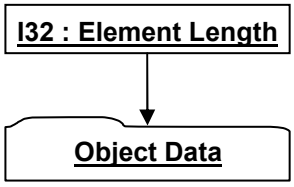


Figure 17 — Logical Element Header data collection

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.

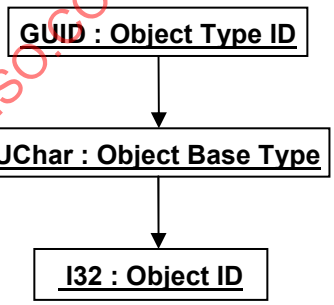


Figure 18 — Element Header data collection

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 **Table 7 — Object** Base Types then the loader should simply skip (read pass) Element Length number of bytes.

Valid Object Base Types include the following:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 7 — Object Base Types

Base Type	Description	Base Type's Data Format
255	None	None
0	Base Graph Node Object	5.1.1.1.1 Base Node Data
1	Group Graph Node Object	5.1.1.3.1 Group Node Data
2	Shape Graph Node Object	0 Base Shape Data
3	Base Attribute Object	5.1.2.1.1 Base Attribute Data
4	Shape LOD	None
5	Base Property Object	0 Base Property Atom Data
6	JT Object Reference Object	5.2.5 JT Object Reference Property Atom Element without the Logical Element Header ZLIB data collection.
8	JT Late Loaded Property Object	5.2.7 Late Loaded Property Atom Element without the Logical Element Header ZLIB data collection.
9	JtBase (none)	None

I32 : Object ID

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

Logical Element Header ZLIB

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

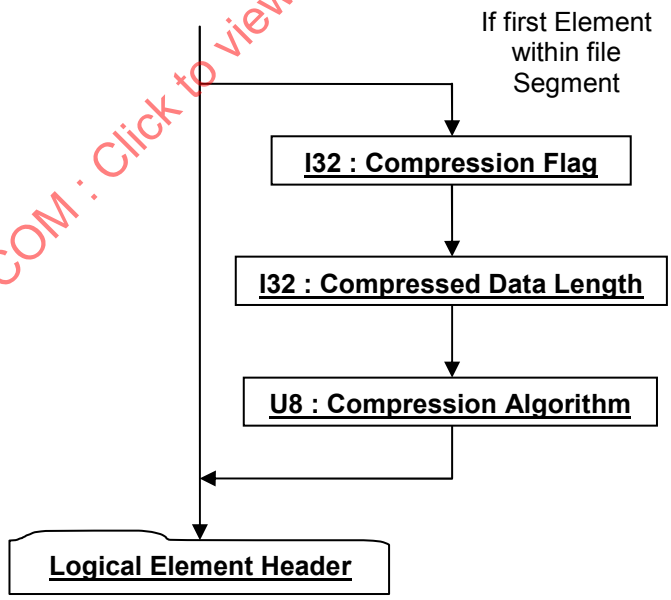


Figure 19 — Logical Element Header ZLIB data collection

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

I32 : Compression Flag

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 8 — Compression flag values

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

I32 : Compressed Data Length

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

U8 : Compression Algorithm

Compression Algorithm specifies the compression algorithm applied to all data in the Segment. Valid values include the following:

Table 9 — Compression algorithm values

= 1	No compression
= 2	ZLIB compression

Object Data

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

4.2 Data Segments

A JT file 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.

— 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 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.

— 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. Reference 11 PMI Data Segment for best practice

— 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

5 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 ZLIB compression on all element data, so all elements in LSG Segment use the Logical Element Header ZLIB form of element header data.

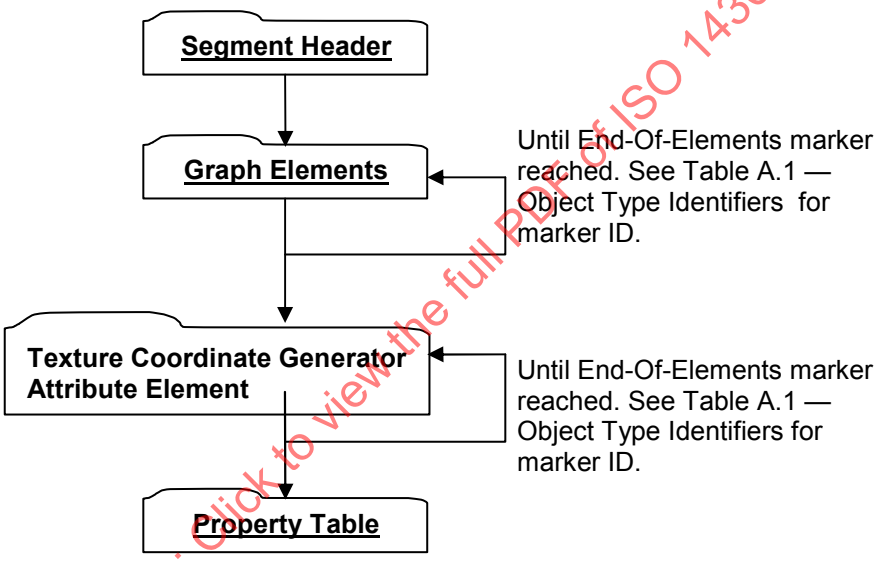


Figure 20 — LSG Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

5.1 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

5.1.1 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.

5.1.1.1 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.

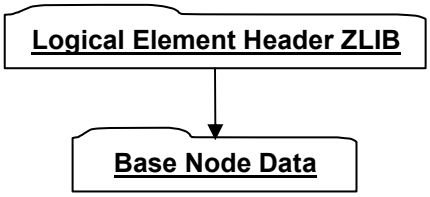


Figure 21 — Base Node Element data collection

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

5.1.1.1.1 Base Node Data

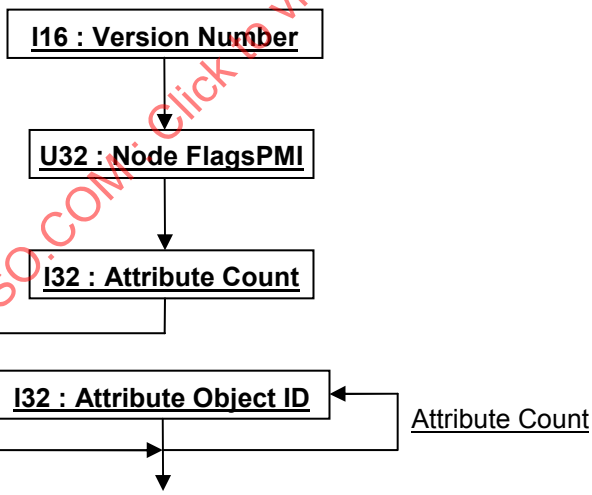


Figure 22 — Base Node Data collection

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers

U32 : Node FlagsPMI

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".

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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.

5.1.1.2 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

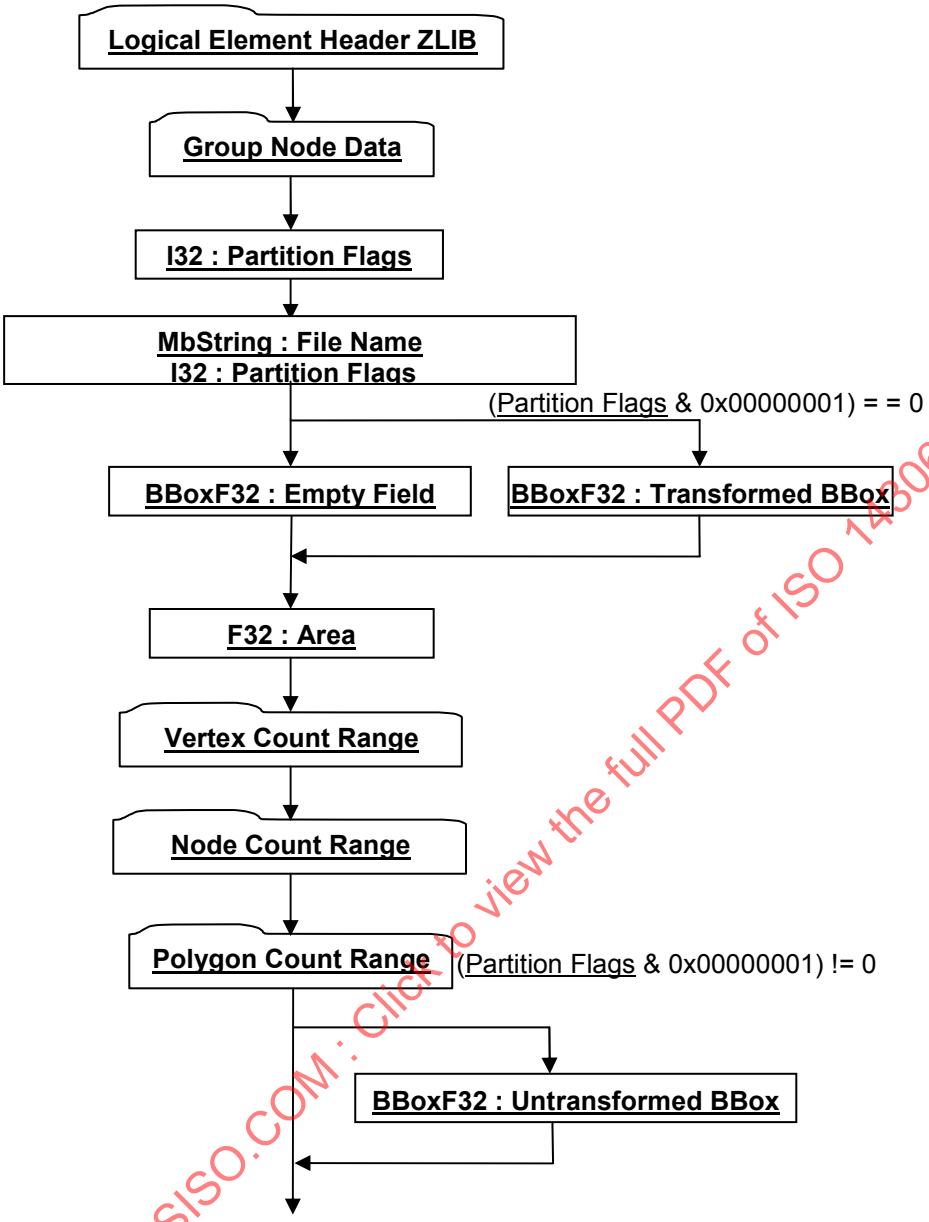


Figure 23 — Partition Node Element data collection

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

Complete description for Group Node Data can be found in 5.1.1.3.1 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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

BBoxF32 : Empty Field

Refer to best practice 14.4 Empty Field

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 descendents. This value is stored in MCS coordinate space (i.e. values scaled by MCS scaling).

BBoxF32 : Untransformed BBox

The Untransformed BBox is only present if Bit 0x00000001 of Partition Flags data field is ON. The Untransformed BBox is an LCS axis-aligned bounding box and represents the untransformed geometry extents for all geometry contained in the Partition Node. This bounding box information may be used by a renderer of 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).

5.1.1.2.1 Vertex Count Range

Vertex Count Range is the aggregate minimum and maximum vertex count for all descendants of the Partition Node. There is a minimum and maximum value to accommodate descendant branches having LOD nodes, which encompass a range of count values within the branch, and to accommodate nodes that can themselves generate varying representations. The 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.

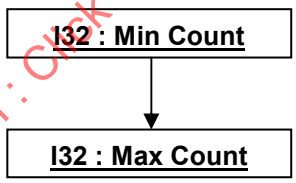


Figure 24 — Vertex Count Range data collection

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.

5.1.1.2.2 Node Count Range

Node Count Range is the aggregate minimum and maximum count of all node descendants of the Partition Node. There is a minimum and maximum value to accommodate descendant branches having LOD nodes, which encompass a range of descendant node count values within the branch. The minimum value represents the least node count that can be achieved by 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 5.1.1.2.1 Vertex Count Range.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

5.1.1.2.3 Polygon Count Range

Polygon Count Range is the aggregate minimum and maximum polygon count for all descendants of the Partition Node. There is a minimum and maximum value to accommodate descendant branches having LOD nodes, which encompass a range of count values within the branch, and to accommodate nodes that can themselves generate varying representations. 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 5.1.1.2.1 Vertex Count Range.

5.1.1.3 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.

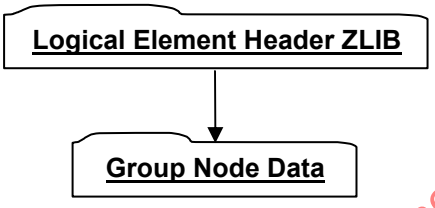


Figure 25 — Group Node Element data collection

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

5.1.1.3.1 Group Node Data

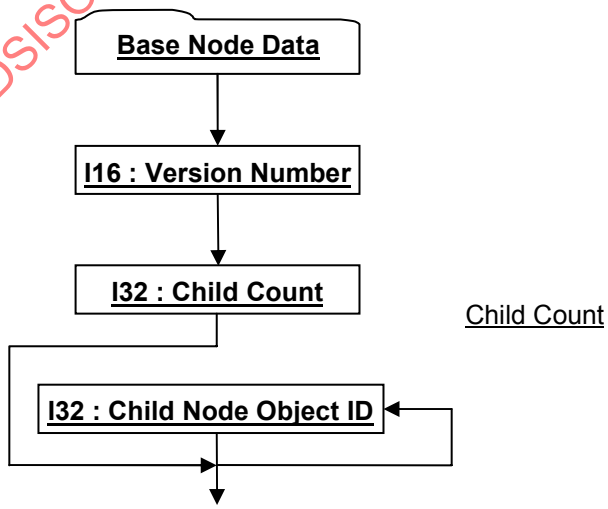


Figure 26 — Group Node Data collection

Complete description for Base Node Data can be found in 5.1.1.1.1 Base Node Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 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.

5.1.1.4 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.

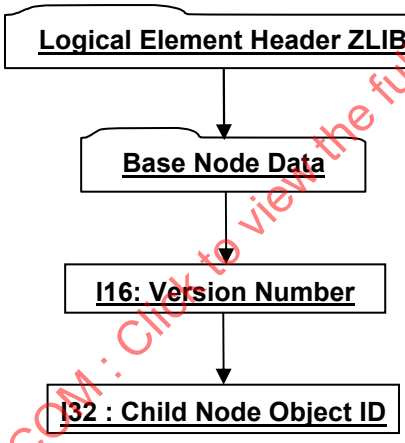


Figure 27 — Instance Node Element data collection

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

Complete description for Base Node Data can be found in 5.1.1.1.1 Base Node Data.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

I32 : Child Node Object ID

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

5.1.1.5 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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

typically references (using Late Loaded Property Atoms) additional Part specific geometric data and/or properties (e.g. B-Rep data, PMI data).

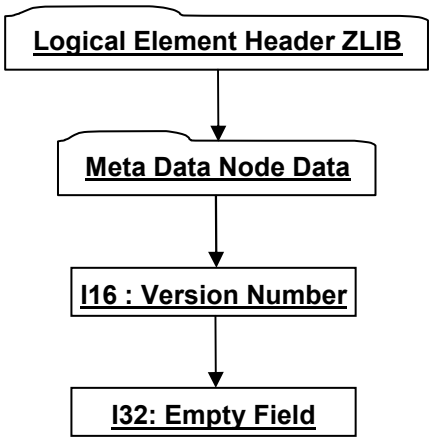


Figure 28 — Part Node Element data collection

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

Complete description for Meta Data Node Data can be found in 5.1.1.6.1 Meta Data Node Data.

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

I32: Empty Field

Refer to best practice 14.4 Empty Field

5.1.1.6 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 10 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

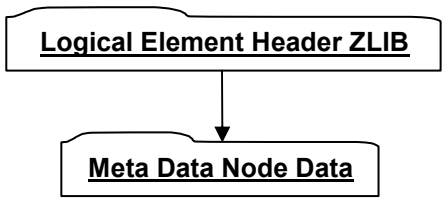


Figure 29 — Meta Data Node Element data collection

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

© ISO 2012 – All rights reserved

5.1.1.6.1 Meta Data Node Data

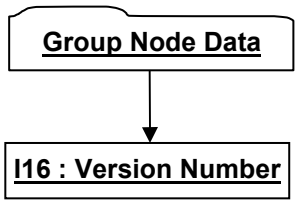


Figure 30 — Meta Data Node Data collection

Complete description for Group Node Data can be found in 5.1.1.3.1 Group Node Data.

I16 : Version Number

Version Number is the version identifier for this data. For information on local version numbers see best practice 14.5 Local version numbers.

5.1.1.7 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.

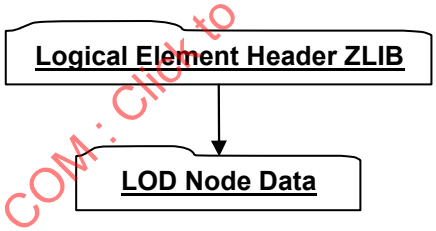


Figure 31 — LOD Node Element data collection

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

5.1.1.7.1 LOD Node Data

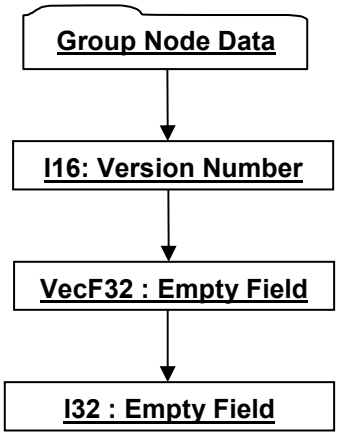


Figure 32 — LOD Node Data collection

Complete description for Group Node Data can be found in 5.1.1.3.1 Group Node Data.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

VecF32 : Empty Field

A vector data field. Refer to best practice 14.4 Empty Field

I32 : Empty Field

Refer to best practice 14.4 Empty Field

5.1.1.8 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

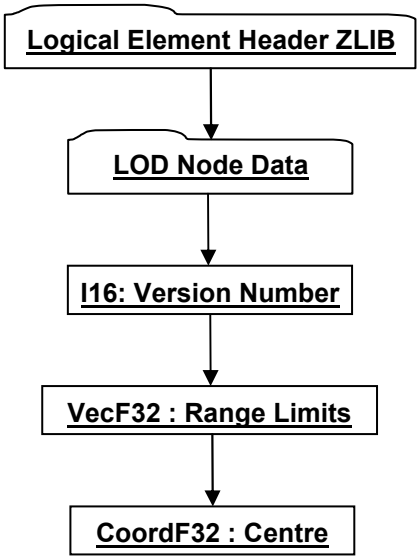


Figure 33 — Range LOD Node Element data collection

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

Complete description for LOD Node Data can be found in 5.1.1.7.1 LOD Node Data

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 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 alternative 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

5.1.1.9 Switch Node Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

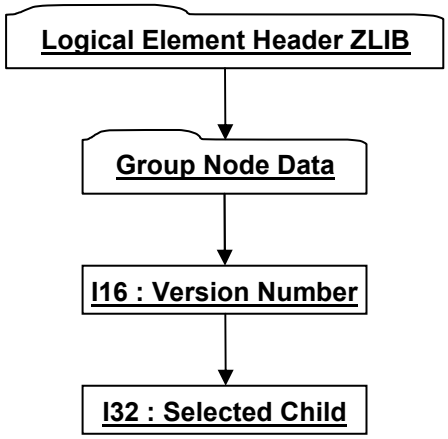


Figure 34 — Switch Node Element data collection

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

Complete description for Group Node Data can be found in 5.1.1.3.1 Group Node Data.

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers

I32 : Selected Child

Selected Child is the index for the selected child node. Valid Selected Child values reside within the following range: "-1 < Selected Child < Child Count". Where "-1" indicates that no child is to be selected and "Child Count" is the data field value from 5.1.1.3.1 Group Node Data.

5.1.1.10 Shape Node Elements

Shape Node Elements are "leaf" nodes within the LSG structure and contain or reference the geometric shape definition data (e.g. vertices, polygons, normals, etc.).

Typically Shape Node Elements do not directly contain the actual geometric shape definition data, but instead reference (using Late Loaded Property Atoms) Shape LOD Segments within the file for the actual geometric shape definition data. Storing the geometric shape definition data within separate independently addressable data segments in the JT file, allows a JT file reader to be structured to support the "best practice" of delaying the loading/reading of associated data until it is actually needed. Complete descriptions for Late Loaded Property Atom Elements and Shape LOD Segments can be found in 5.2.7 Late Loaded Property Atom Element and 6 Shape LOD Segment respectively.

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

5.1.1.10.1 Base Shape Node Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

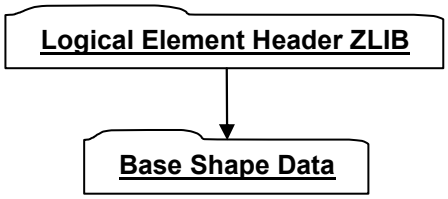
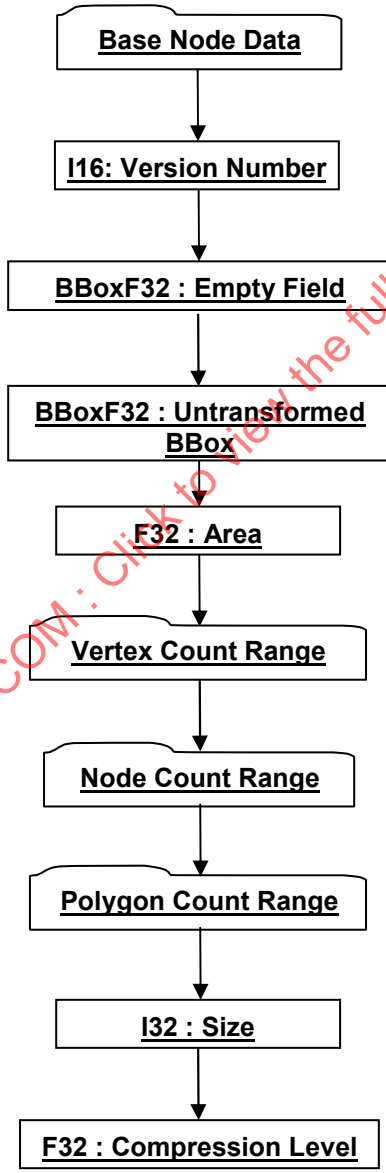


Figure 35 — Base Shape Node Element data collection

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

Base Shape Data



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 36 — Base Shape Data collection

Complete description for Base Node Data can be found in 5.1.1.1.1 Base Node Data

© ISO 2012 – All rights reserved

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

BBoxF32 : Empty Field

Refer to best practice 14.4 Empty Field

BBoxF32 : Untransformed BBox

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

F32 : Area

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

I32 : Size

Size specifies the in memory length in bytes of the associated/referenced Shape LOD Element

. This Size value has no relevancy to the on-disk (JT File) size of the associated/referenced Shape LOD Element

. A value of zero indicates that the in memory size is unknown. See 6.1 Shape LOD Element

for complete description of Shape LOD Elements. JT file loaders/readers can leverage this Size value during late load processing to help pre-determine if there is sufficient memory to load the Shape LOD Element

.

F32 : Compression Level

Compression Level specifies the qualitative compression level applied to the associated/referenced Shape LOD Element

. See 6.1 Shape LOD Element

for complete description of Shape LOD Elements. This compression level value is a qualitative representation of the compression applied to the Shape LOD Element

. The absolute compression (derived from this qualitative level) applied to the Shape LOD Element

is physically represented in the JT format by other data stored with both the Shape Node and the Shape LOD Element

(e.g. Quantization Parameters), and thus it's not necessary to understand how to map this qualitative value to absolute compression values in order to uncompress/decode the data

Table 12 — Compression level values

= 0.0	“Lossless” compression used.
= 0.1	“Minimally Lossy” compression used. This setting generally results in modest compression ratios with little if any visual difference when compared to the same images rendered from “Lossless” compressed Shape LOD Element.
= 0.5	“Moderate Lossy” compression used. The setting results in more data loss than “Minimally Lossy” and thus higher compression ratio is obtained. Some visual

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

	difference will likely be noticeable when compared to the same images rendered from “Lossless” compressed Shape LOD Element.
= 1.0	“Aggressive Lossy” compression used. With this setting as much data as possible will be thrown away, resulting in highest compression ratio, while still maintaining a modestly useable representation of the underlying data. Visual differences may be evident when compared to the same images rendered from “Lossless” compressed Shape LOD Element.

Vertex Count Range

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

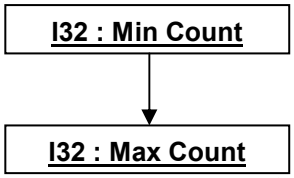


Figure 37 —Vertex Count Range data collection

I32 : Min Count

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

I32 : Max Count

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

Node Count Range

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

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

Polygon Count Range

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

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

5.1.1.10.2 Vertex Shape Node Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

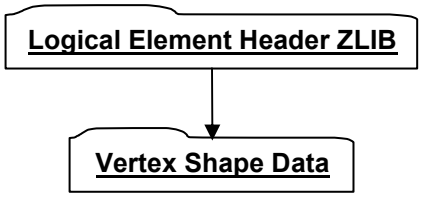


Figure 38 — Vertex Shape Node Element data collection

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

Vertex Shape Data

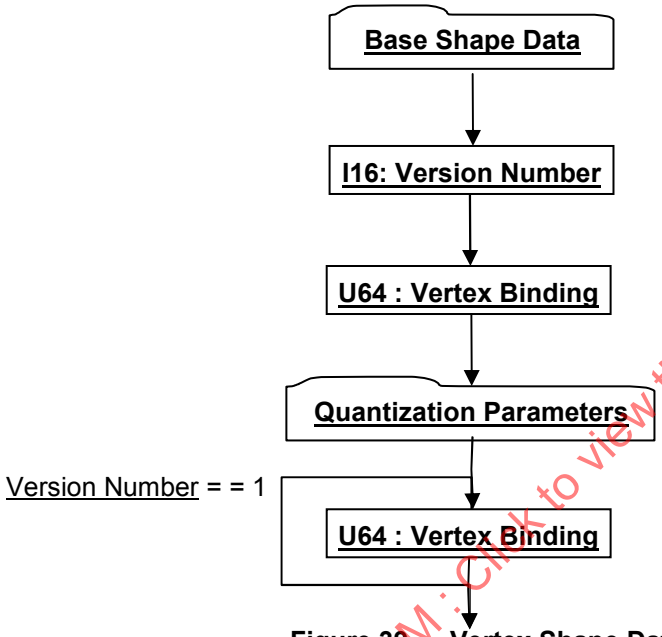


Figure 39 — Vertex Shape Data collection

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

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

U64 : Vertex Binding

Vertex Bindings is a collection of normal, texture coordinate, and colour binding information encoded within a single U64. All bits fields that are not defined as in use should be set to "0". For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

Quantization Parameters

Quantization Parameters specifies for each shape data type grouping (i.e. Vertex, Normal, Texture Coordinates, Colour) the number of quantization bits used for given qualitative compression level. Although these Quantization Parameters values are saved in the associated/referenced Shape LOD Element, they are also saved here so that a JT File loader/reader does not have to load the Shape LOD Element in order to determine the Shape quantization level. See 6.1 Shape LOD Element

for complete description of Shape LOD Elements.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

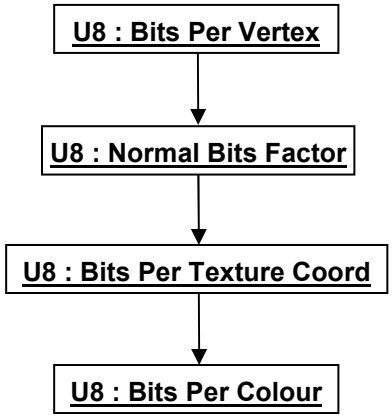


Figure 40 — Quantization Parameters data collection

U8 : Bits Per Vertex

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

U8 : Normal Bits Factor

Normal Bits Factor is a parameter used to calculate the number of quantization bits for normal vectors. Value shall be within range [0:13] inclusive. The actual number of quantization bits per normal is computed using this factor and the following formula: "BitsPerNormal = 6 + 2 * Normal Bits Factor"

U8 : Bits Per Texture Coord

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

U8 : Bits Per Colour

Bits Per Colour specifies the number of quantization bits per colour component. Value shall be within range [0:24] inclusive.

5.1.1.10.3 Tri-Strip Set Shape Node Element

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

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

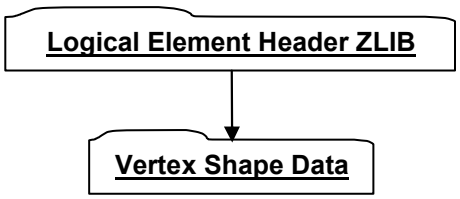


Figure 41 — Tri-Strip Set Shape Node Element data collection

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

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

5.1.1.10.4 Polyline Set Shape Node Element

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

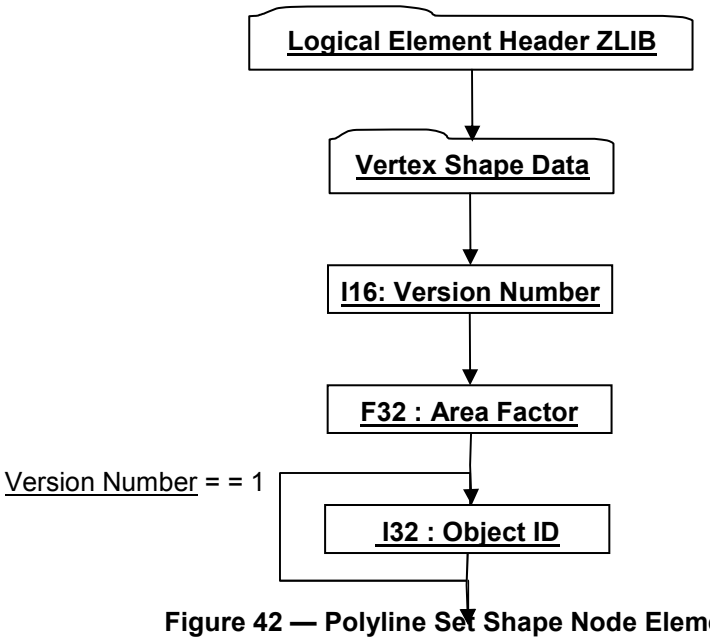


Figure 42 — Polyline Set Shape Node Element data collection

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

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

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

F32 : Area Factor

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

U64: Vertex Bindings

Vertex Bindings is a collection of normal, texture coordinate, and colour binding information encoded within a single U64. All bits fields that are not defined as in use should be set to “0”. For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

5.1.1.10.5 Point Set Shape Node Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

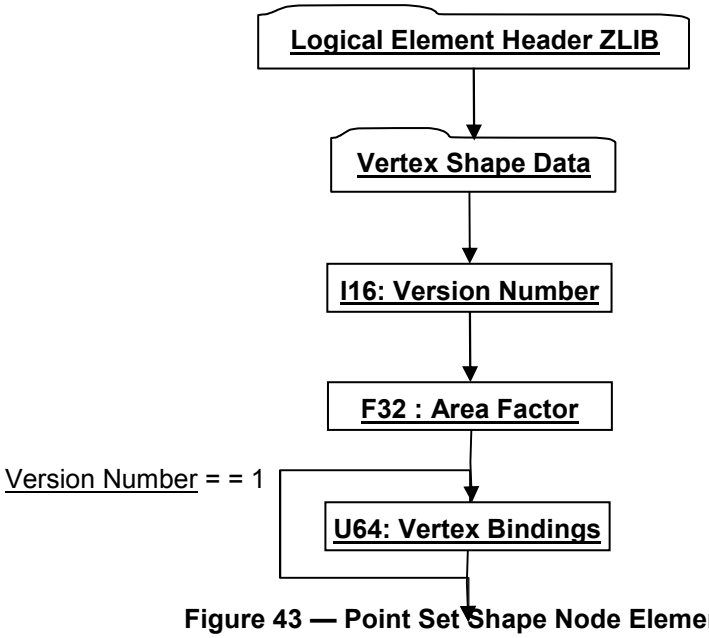


Figure 43 — Point Set Shape Node Element data collection

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

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

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers

F32 : Area Factor

Area Factor specifies a multiplier factor applied to the Point Set computed surface area. In JT data viewer applications there may be LOD selection semantics that are based on screen coverage calculations. The computed "surface area" of a Point Set is equal to the larger (i.e. whichever is greater) of either the area of the Point Set's bounding box, or "1.0". Area Factor scales the result of this "surface area" computation..

U64: Vertex Bindings

Vertex Bindings is a collection of normal, texture coordinate, and colour binding information encoded within a single U64. All bits fields that are not defined as in use should be set to "0". For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

5.1.1.10.6 Polygon Set Shape Node Element

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

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

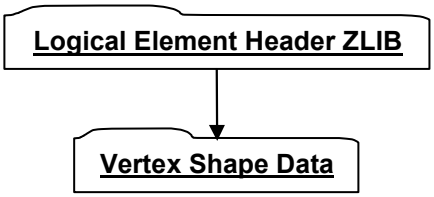


Figure 44 — Polygon Set Shape Node Element data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

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

5.1.1.10.7 NULL Shape Node Element

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

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

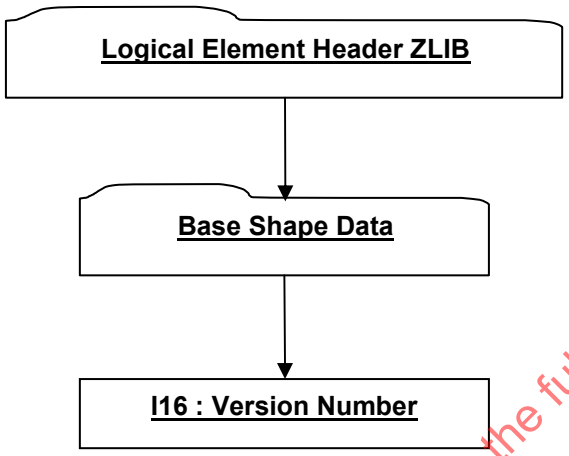


Figure 45 — NULL Shape Node Element data collection

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

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

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

5.1.1.10.8 Primitive Set Shape Node Element

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

A Primitive Set Shape Node Element represents a list/set of primitive shapes (e.g. box, cylinder, sphere, etc.) whose LODs can be procedurally generated. "Procedurally generate" means that the raw geometric shape definition data (e.g. vertices, polygons, normals, etc) for LODs is not directly stored; instead some basic shape information is stored (e.g. sphere centre and radius) from which LODs can be generated.

Primitive Set Shape Node Elements actually do not even directly contain this basic shape definition data, but instead reference (using Late Loaded Property Atoms) Primitive Set Shape Elements within the file for the actual basic shape definition data. Storing the basic shape definition data within separate independently addressable data segments in the JT file, allows a JT file reader to be structured to support the "best practice" of delaying the loading/reading of associated data until it is actually needed. Complete descriptions for Late Loaded Property Atom Elements and Primitive Set Shape Element can be found in 5.2.7 Late Loaded Property Atom Element and 6.2 Primitive Set Shape Element respectively.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

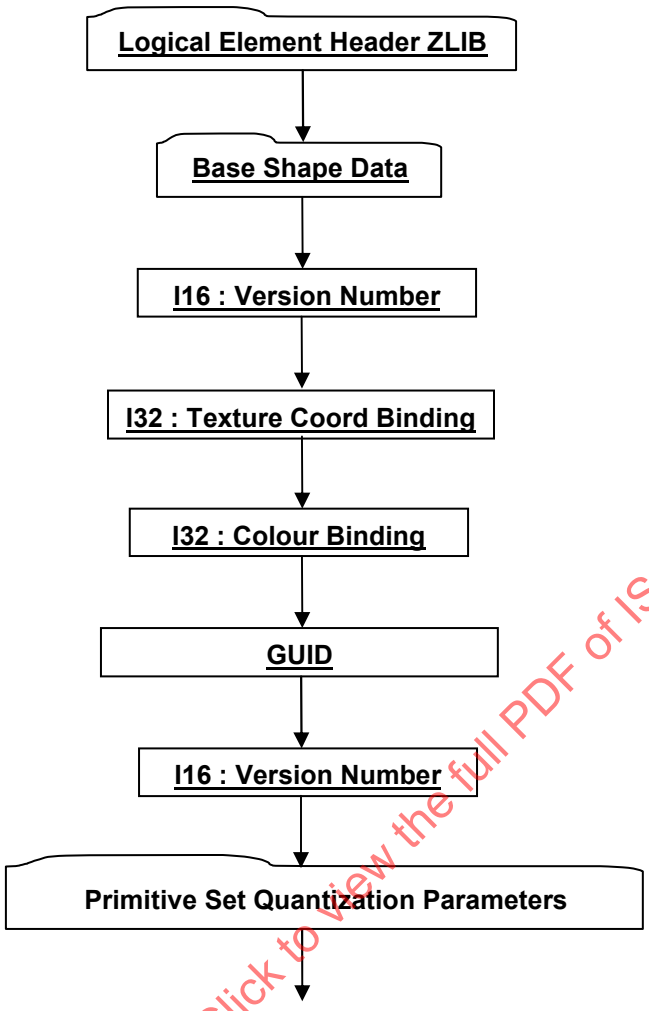


Figure 46 — Primitive Set Shape Node Element data collection

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

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

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

I32 : Texture Coord Binding

Texture Coord Binding specifies how (at what granularity) texture coordinate data is supplied ("bound") for the shape in the associated/referenced Shape LOD Element. Valid values are as follows:

Table 13 — Texture Coord Binding values

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

I32 : Colour Binding

Colour Binding specifies how (at what granularity) colour data is supplied ("bound") for the shape in the associated/referenced Shape LOD Element. Valid values are the same as documented for Texture Coord Binding data field.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Version Number

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

Table 14 — Version Number values

= 0	Version 0 Format
= 1	Version 1 Format

I32 : Texture Coord Gen Type

Texture Coord Gen Type specifies how a texture is applied to each face of the primitive. Single tile means one copy of the texture will be stretched to fit the face, isotropic means that the texture will be duplicated on the longer dimension of the face in order to maintain the texture's aspect ratio

Table 15 — Texture Coord Gen Type values

= 0	Single Tile…Indicates that a single copy of a texture image will be applied to significant primitive features (i.e. cube face, cylinder wall, end cap) no matter how eccentrically shaped.
= 1	Isotropic…Implies that multiple copies of a texture image may be mapped onto eccentric surfaces such that a mapped texel stays approximately square.

Primitive Set Quantization Parameters

Primitive Set Quantization Parameters specifies for the two shape data type grouping (i.e. Vertex, Colour) the number of quantization bits used for given qualitative compression level. Although these Quantization Parameters values are saved in the associated/referenced Shape LOD Element, they are also saved here so that a JT File loader/reader does not have to load the Shape LOD Element in order to determine the Shape quantization level. See 6.1 Shape LOD Element

for complete description of Shape LOD Elements.

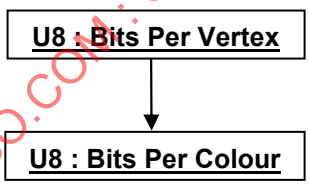


Figure 47 —

Primitive Set Quantization Parameters data collection

U8 : Bits Per Vertex

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

U8 : Bits Per Colour

Bits Per Colour specifies the number of quantization bits per colour component. Value shall be within range [0:24] inclusive.

5.1.2 Attribute Elements

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

To that end each attribute type defines its own application and accumulation semantics, but in general attributes at lower levels in the LSG take precedence and replace or accumulate with attributes set at higher levels. Nodes without associated attributes inherit those of their parents. Attributes inherit only from their parents, thus a node's attributes do not affect that node's siblings. The root of a partition inherits the attributes in effect at the referring partition node.

Attributes can be declared "final" (see 5.1.2.1.1 Base Attribute Data), which terminates accumulation of that attribute type at that attribute and propagates the accumulated values there to all descendants of the associated node. Descendants can explicitly do a one-shot override of "final" using the attribute "force" flag (see 5.1.2.1.1 Base Attribute Data), but do not by default. Note that "force" does not turn OFF "final" – it is simply a one-shot override of "final" for the specific attribute marked as "forcing." An analogy for this "force" and "final" interaction is that "final" is a back-door in the attribute accumulation semantics, and that "force" is a doggy-door in the back-door!

5.1.2.1 Common Attribute Data Containers

5.1.2.1.1 Base Attribute Data

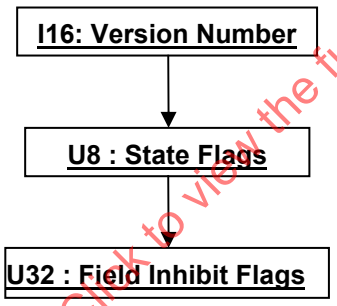


Figure 48 — Base Attribute Data collection

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

U8 : State Flags

State Flags is a collection of flags. The flags are combined using the binary OR operator and store various state information for Attribute Elements; such as indicating that the attributes accumulation is final. All bits fields that are not defined as in use should be set to "0".

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 16 — State Flag values

0x01	Accumulation Final flag. Provides a means to terminate a particular attribute type's accumulation at any node of the LSG and thereby force all descendants to have that value of the attribute. = 0 – Accumulation is to occur normally = 1 – Accumulation is "final"
0x02	Accumulation Force flag. Provides a way to assign nodes in LSG, attributes that shall not be overridden by ancestors. = 0 – Accumulation of this attribute obeys ancestor's Final flag setting. = 1 – Accumulation of this attribute is forced (overrides ancestor's Final flag setting)
0x04	Accumulation Ignore Flag. Provides a way to indicate that the attribute is to be ignored (not accumulated). = 0 – Attribute is to be accumulated normally (subject to values of Force/Final flags) = 1 – Attribute is to be ignored.
0x08	Attribute Persistable Flag. Provides a way to indicate that the attribute is to be persistable to a JT file. = 0 – Attribute is to be non-persistable. = 1 – Attribute is to be persistable.

U32 : Field Inhibit Flags

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

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

5.1.2.2 Material Attribute Element

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

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

The Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments for the Material Attribute Element data fields, are as follows:

Table 17 — Material Attribute data field inihibit values

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

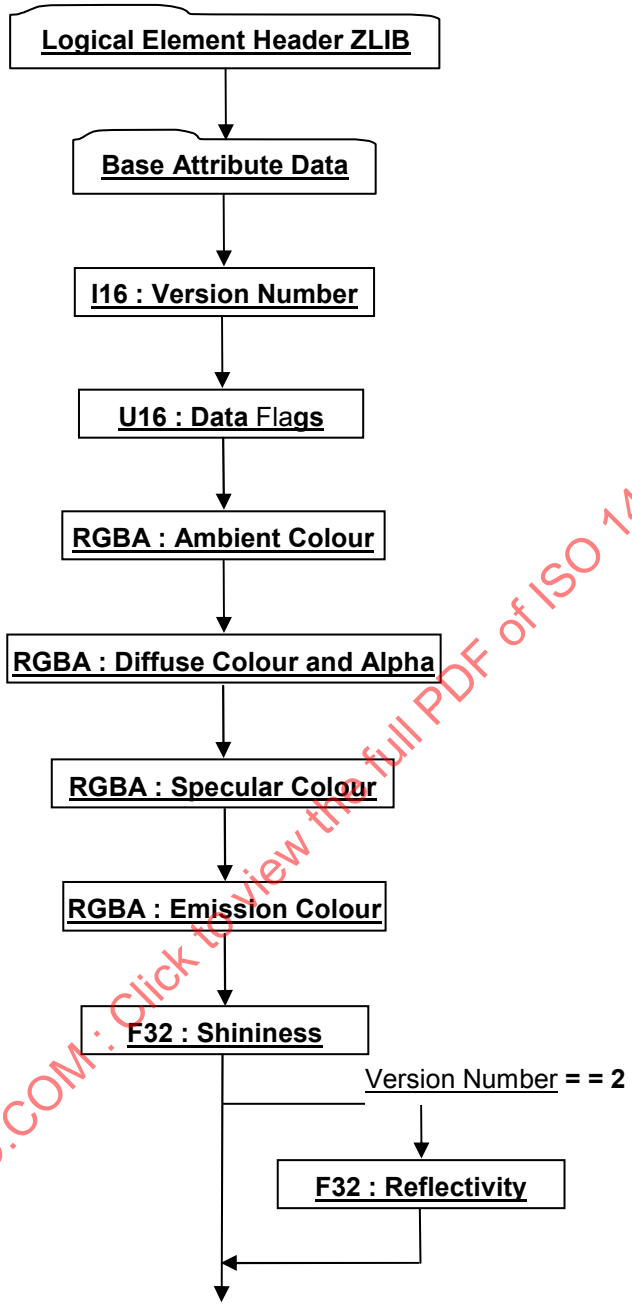


Figure 49 — Material Attribute Element data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

I16 : Version Number

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

© ISO 2012 – All rights reserved

Table 18 — Material Attribute Version number value

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

U16 : Data Flags

Data Flags is a collection of flags and factor data. The flags and factor data are combined using the binary OR operator. The flags store information to be used for interpreting how to read subsequent Material data fields. All bits fields that are not defined as in use should be set to "0".

Table 19 — Material Attribute Data Flag values

0x0010	Blending Flag. Blending is a colour combining operation in the graphics pipeline that happens just before writing a colour to the framebuffer. If Blending is ON then incoming fragment RGBA colour values are used (based on Source Blend Factor) and existing framebuffer's RGBA colour values are used (based on Destination Blend Factor) to blend between the incoming fragment RGBA and the current frame buffer RGBA to arrive at a new RGBA colour to write into the framebuffer. If Blending is OFF then incoming fragment RGBA colour is written directly into framebuffer unmodified (i.e. completely overriding existing framebuffer RGBA colour). = 0 – Blending OFF. = 1 – Blending ON
0x0020	Override Vertex Colours Flag. If ON, then a shape's per vertex colours are to be overridden by the accumulated Material colour. = 0 – Override OFF = 1 – Override ON
0x07C0	Source Blend Factor (stored in bits 6 – 10 or in binary notation 0000011111000000). If Blending Flag enabled, this value indicates how the incoming fragment's (i.e. the source) RGBA colour values are to be used to blend with the current framebuffer's (i.e. the destination) RGBA colour values. Additional information on the interpretation of the Blending Factor values and how one might leverage them to render an image can be found in reference [1] listed in the bibliography section. = 0 – Interpret same as OpenGL **GL_ZERO** Blending Factor = 1 – Interpret same as OpenGL **GL_ONE** Blending Factor = 2 – Interpret same as OpenGL **GL_DST_COLOUR** Blending Factor = 3 – Interpret same as OpenGL **GL_SRC_COLOUR** Blending Factor = 4 – Interpret same as OpenGL **GL_ONE_MINUS_DST_COLOUR** Blending Factor = 5 – Interpret same as OpenGL **GL_ONE_MINUS_SRC_COLOUR** Blending Factor = 6 – Interpret same as OpenGL **GL_SRC_ALPHA** Blending Factor = 7 – Interpret same as OpenGL **GL_ONE_MINUS_SRC_ALPHA** Blending Factor = 8 – Interpret same as OpenGL **GL_DST_ALPHA** Blending Factor = 9 – Interpret same as OpenGL **GL_ONE_MINUS_DST_ALPHA** Blending Factor = 10 – Interpret same as OpenGL **GL_SRC_ALPHA_SATURATE** Blending Factor
0xF800	Destination Blend Factor (stored in bits 11 – 15 or in binary notation 1111100000000000).). If Blending Flag enabled, this value indicates how the current framebuffer's (the destination) RGBA colour values are to be used to blend with the incoming fragment's (the source) RGBA colour values. Additional information on the interpretation of the Blending Factor values and how one might leverage them to render an image can be found in reference [1] listed the bibliography section. = 0 – Interpret same as OpenGL **GL_ZERO** Blending Factor = 1 – Interpret same as OpenGL **GL_ONE** Blending Factor = 2 – Interpret same as OpenGL **GL_DST_COLOUR** Blending Factor

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

	= 3 – Interpret same as OpenGL **GL_SRC_COLOUR** Blending Factor = 4 – Interpret same as OpenGL **GL_ONE_MINUS_DST_COLOUR** Blending Factor = 5 – Interpret same as OpenGL **GL_ONE_MINUS_SRC_COLOUR** Blending Factor = 6 – Interpret same as OpenGL **GL_SRC_ALPHA** Blending Factor = 7 – Interpret same as OpenGL **GL_ONE_MINUS_SRC_ALPHA** Blending Factor = 8 – Interpret same as OpenGL **GL_DST_ALPHA** Blending Factor = 9 – Interpret same as OpenGL **GL_ONE_MINUS_DST_ALPHA** Blending Factor = 10 – Interpret same as OpenGL **GL_SRC_ALPHA_SATURATE** Blending Factor

RGBA : Ambient Colour

Ambient Colour specifies the ambient red, green, blue, alpha colour values of the material.

RGBA : Diffuse Colour and Alpha

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

RGBA : Specular Colour

Specular Colour specifies the specular red, green, blue, alpha colour values of the material.

RGBA : Emission Colour

Emission Colour specifies the emissive red, green, blue, alpha colour values of the material.

F32 : Shininess

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

F32 : Reflectivity

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

5.1.2.3 Texture Image Attribute Element

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

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

Note that additional information on the interpretation of the various Texture Image Attribute Element data fields can be found in the OpenGL references listed in the bibliography section [1GUID].

The Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments for the Texture Image Attribute Element data fields, are as follows:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 20 — Texture Image Attribute data field inihibit values

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

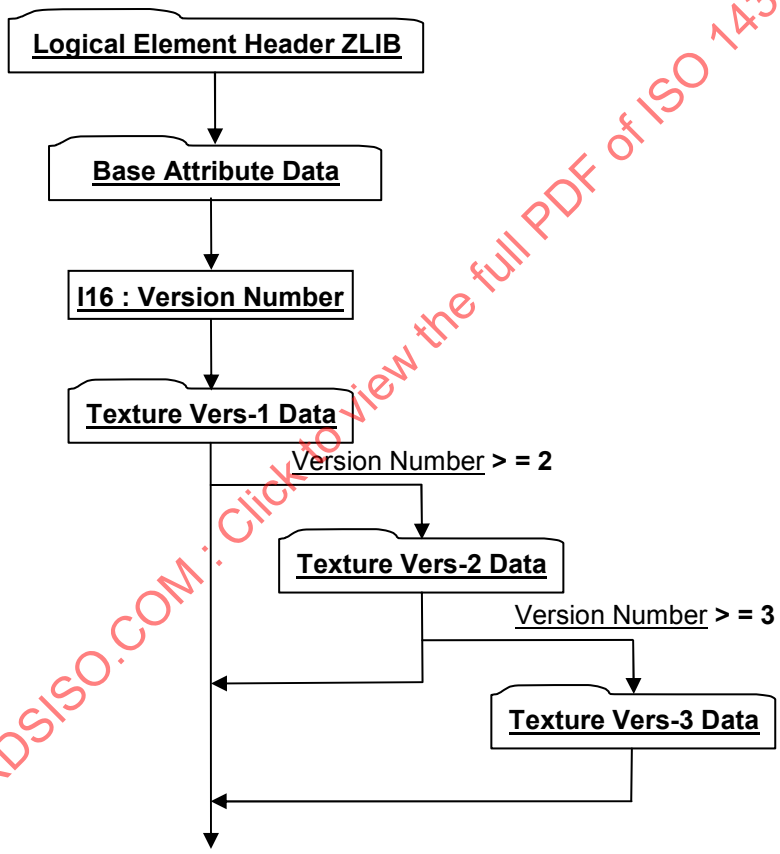


Figure 50 — Texture Image Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

Complete description for Texture Vers-1 Data can be found in 5.1.2.3.1 Texture Vers-1 Data.

Complete description for Texture Vers-2 Data can be found in 5.1.2.3.2 Texture Vers-2 Data.

Complete description for Texture Vers-3 Data can be found in 5.1.2.3.3 Texture Vers-3 Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Version Number

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

Table 21 — Texture Image Version Number values

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

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

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

5.1.2.3.1 Texture Vers-1 Data

Texture Vers-1 Data format is stored in JT file if the Texture Image Element is a vanilla/basic texture image (i.e. if texture does not use any advanced features as described in 5.1.2.3.2 Texture Vers-2 Data and 5.1.2.3.3 Texture Vers-3 Data). However, advanced textures *also* write a Texture Vers-1 Data block because of the need to be backward-compatible with older readers that may not understand Vers-2 and Vers-3 data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

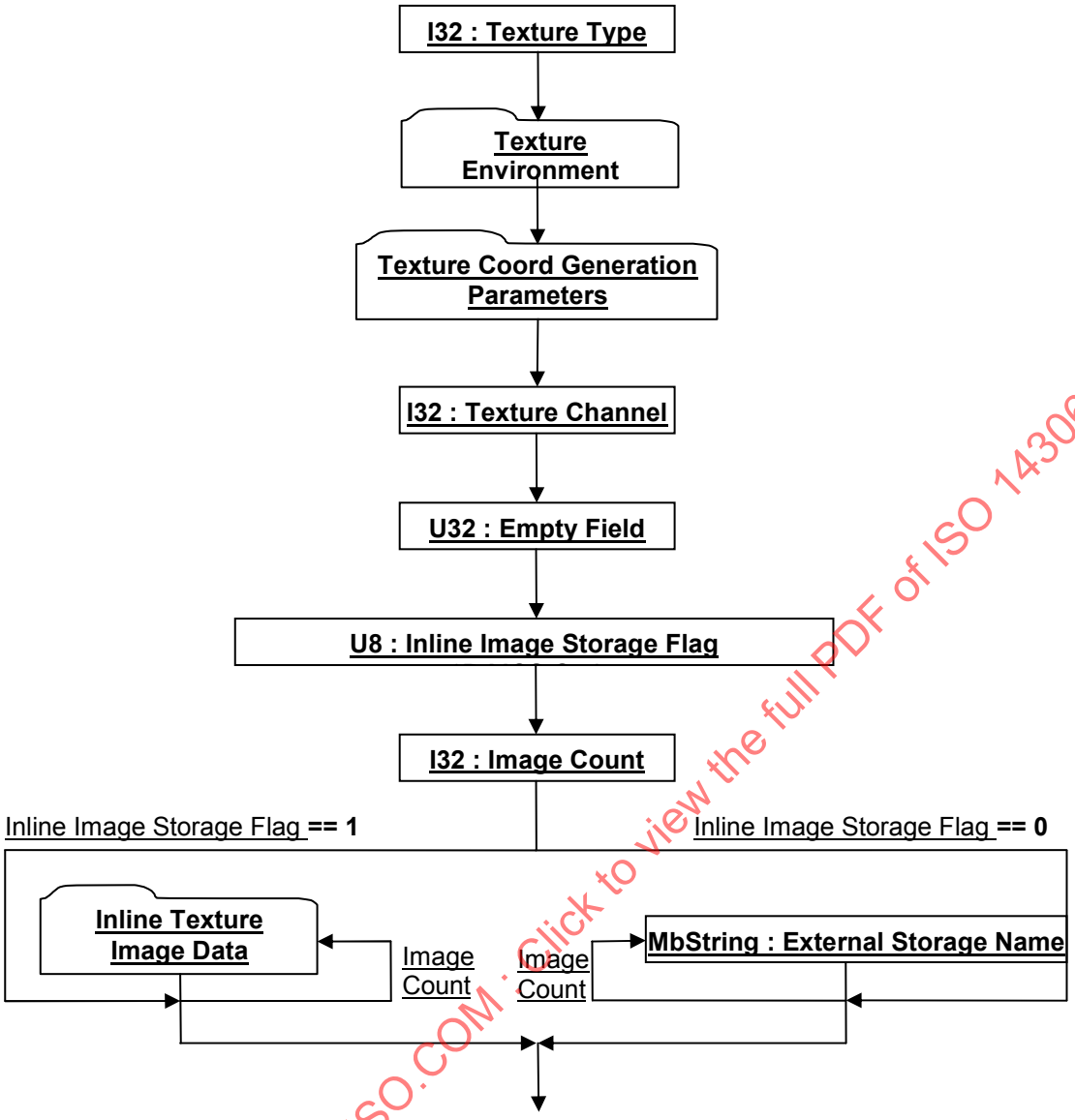


Figure 51 — Texture Vers-1 Data collection

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

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

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

I32 : Texture Type

Texture Type specifies the type of texture.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 22 — Texture Vers-1 type values

= 0	None.
= 1	One-Dimensional. A one-dimensional texture has a height (T-Dimension) and depth (R-Dimension) equal to “1” and no top or bottom border.
= 2	Two-Dimensional. A two-dimensional texture has a depth (R-Dimension) equal to “1.”
= 3	Three-Dimensional. A three-dimensional texture can be thought of as layers of two-dimensional sub image rectangles arranged in a sequence.
= 4	Bump Map. A bump map texture is a texture where the image texel data (e.g. RGB colour values) represents surface normal XYZ components.
= 5	Cube Map. A cube map texture is a texture cube centreed at the origin and formed by a set of six two-dimensional texture images.
= 6	Depth Map. A depth map texture is a texture where the image texel data represents depth values.

I32 : Texture Channel

Texture Channel specifies the texture channel number for the Texture Image Element. For purposes of multi-texturing, the JT concept of a texture channel corresponds to the OpenGL concept of a “texture unit.” The Texture Channel value shall be between 0 and 31 inclusive. Best practices suggest that renderer of JT data ignore all but channel-0 if the renderer does not support multi-textured geometry. Also for purposes of blending, renderer of JT data should assume that higher numbered texture channels “blend over” lower numbered ones.

U32 : Empty Field

Refer to best practice 14.4 Empty Field

U8 : Inline Image Storage Flag

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

Table 23 — Texture Vers-1 Inline Image Storage Flag vaules

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

I32 : Image Count

Image Count specifies the number of texture images. A “Cube Map” I32 : Texture Type shall have six images while all other Texture Types should only have one image.

MbString : External Storage Name

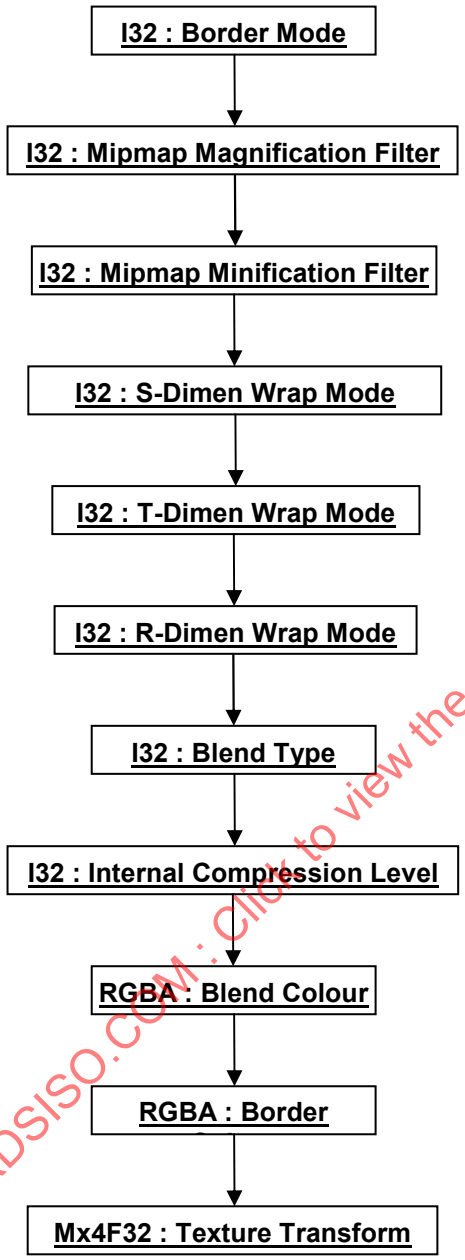
External Storage Name is a string identifying the name of an external texture image storage. External Storage Name is only present if data field Inline Image Storage Flag equals “0.” If present there will be data field Image Count number of External Storage Name instances. This External Storage Name string is a relative path based name for the texture image file. Where “relative path” should be interpreted to mean the string contains the file name along with any additional path information that locates the texture image file relative to the location of the referencing JT file.

Texture Environment

The Texture Environment is a collection of data defining various aspects of how a texture image is to be mapped/applied to a surface.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 52 — Texture Environment data collection

I32 : Border Mode

Border Mode specifies the texture border mode.

Table 24 — Texture Vers-1 Texture Environment Border Mode values

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

© ISO 2012 – All rights reserved

I32 : Mipmap Magnification Filter

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

Table 25 — Texture Vers-1 Texture Environment Mipmap Magnification Filter values

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

I32 : Mipmap Minification Filter

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

Table 26 — Texture Vers-1 Texture Environment Mipmap Minification Filter values

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

I32 : S-Dimen Wrap Mode

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

Table 27 — Texture Vers-1 Texture Environment S-Dimen Wrap Mode values

= 0	None.
= 1	Clamp. Any values greater than 1.0 are set to 1.0; any values less than 0.0 are set to 0.0
= 2	Repeat Integer parts of the texture coordinates are ignored (i.e. retains only the fractional component o texture coordinates greater than 1.0 and only one-minus the fractional component of values less than zero). Resulting in copies of the texture map tiling the surface
= 3	Mirror Repeat. Like Repeat, except the surface tiles “flip-flop” resulting in an alternating mirror pattern of surface tiles.
= 4	Clamp to Edge. Border is always ignored and instead texel at or near the edge is chosen for coordinates outside the range [0, 1]. Whether the exact nearest edge texel or some average of the nearest edge texels is used is dependent upon the mipmap filtering value.
= 5	Clamp to Border. Nearest border texel is chosen for coordinates outside the range [0, 1]. Whether the exact nearest border texel or some average of the nearest border texels is used is dependent upon the mipmap filtering value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I32 : T-Dimen Wrap Mode

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

I32 : R-Dimen Wrap Mode

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

I32 : Blend Type

Blend Type contains information indicating how the values in the texture map are to be modulated/combined/blended with the original colour of the surface or some other alternative colour to compute the final colour to be painted on the surface. Additional information on the interpretation of the Blend Type values and how one might leverage them to render an image can be found in reference [1] listed in the bibliography section.

Table 28 — Texture Vers-1 Texture Environment Blend Type values

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

I32 : Internal Compression Level

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

Table 29 — Texture Vers-1 Texture Environment Internal Compression Level values

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

RGBA : Blend Colour

Blend Colour specifies the colour to be used for the “Blend” mode of Blend Type operations.

RGBA : Border Colour

Border Colour specifies the constant border colour to use for “Clamp to Border” style wrap modes when the texture itself does not have a border.

Mx4F32 : Texture Transform

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

Texture Coord Generation Parameters

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

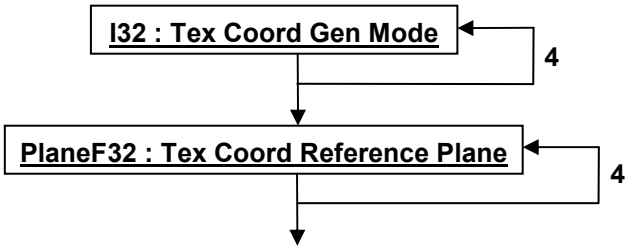


Figure 53 — Texture Coord Generation Parameters data collection

I32 : Tex Coord Gen Mode

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

Table 30 — Texture Vers-1 Texture Coord Generation Gen Mode values

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

PlaneF32 : Tex Coord Reference Plane

Reference Plane specifies the reference plane used for “Model Coordinate System Linear” and “View Coordinate System Linear” texture coordinate generation modes. There are four Reference Planes stored, one for each component of texture coordinate. The Reference Planes are stored in S, T, R, Q order. Even if a components “Tex Coord Gen Mode” is one that does not require a reference plane, dummy reference planes are still stored in JT file.

Inline Texture Image Data

Inline Texture Image Data is a collection of data defining the texture format properties and image texel data for one texture image. Inline Texture Image Data is only present if data field Inline Image Storage Flag equals “1.” If present there will be data field Image Count number of Inline Texture Image Data instances.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

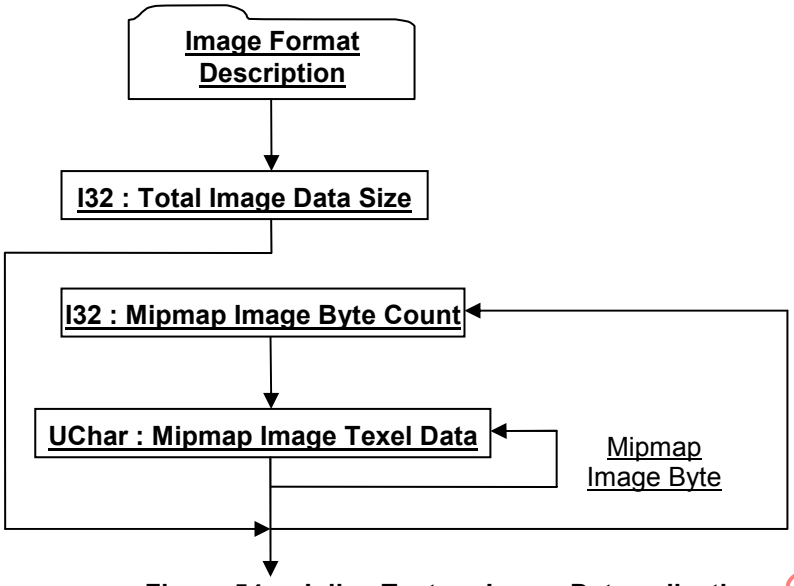


Figure 54 — Inline Texture Image Data collection

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

I32 : Total Image Data Size

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

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

I32 : Mipmap Image Byte Count

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

UChar : Mipmap Image Texel Data

Mipmap Image Texel Data is the mipmap's block of image data. The length of this field in bytes is specified by the value of data field Mipmap Image Byte Count.

Image Format Description

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

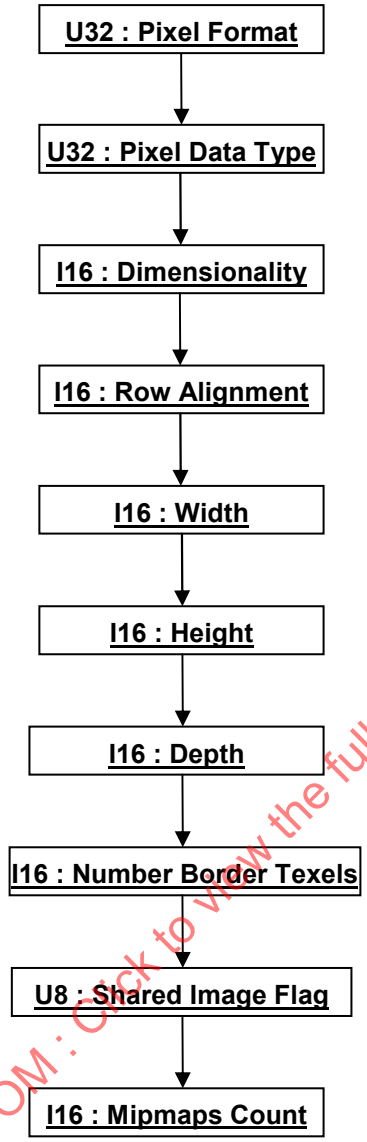


Figure 55 — Image Format Description data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

U32 : Pixel Format

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

Table 31 — Texture Vers-1 Image Format Description Pixel Format values

= 0	No format specified. Texture mapping is not applied.
= 1	RGB: A red colour component followed by green and blue colour components
= 2	RGBA: A red colour component followed by green, blue, and alpha colour components
= 3	LUM: A single luminance component
= 4	LUMA: A luminance component followed by an alpha colour component.
= 5	A single stencil index.
= 6	A single depth component
= 7	A single red colour component
= 8	A single green colour component
= 9	A single blue colour component
= 10	A single alpha colour component
= 11	A blue colour component, followed by green and red colour components
= 12	A blue colour component, followed by green , red, and alpha colour components
= 13	A depth component, followed by a stencil component

U32 : Pixel Data Type

Pixel Data Type specifies the data type used to store the per texel data. If the Pixel Format represents a multi component value (e.g. red, green, blue) then each value requires the Pixel Data Type number of bytes of storage (e.g. a Pixel Format Type of "1" with Pixel Data Type of "3" would require 3 bytes of storage for each texel).

Table 32 — Texture Vers-1 Image Format Description Pixel Data values

= 0	No type specified. Texture mapping is not applied.
= 1	Signed 8-bit integer
= 2	Single-precision 32-bit floating point
= 3	Unsigned 8-bit integer
= 4	Single bits in unsigned 8-bit integers
= 5	Unsigned 16-bit integer
= 6	Signed 16-bit integer
= 7	Unsigned 32-bit integer
= 8	Signed 32-bit integer
= 9	16-bit floating point according to IEEE-754 format (i.e. 1 sign bit, 5 exponent bits, 10 mantissa bits)

I16 : Dimensionality

Dimensionality specifies the number of dimensions the texture image has. Valid values include:

Table 33 — Texture Vers-1 Image Format Description Dimensionality values

= 1	One-dimensional texture
= 2	Two-dimensional texture
= 3	Three-dimensional texture

I16 : Row Alignment

Row Alignment specifies the byte alignment for image data rows. This data field shall have a value of 1, 2, 4, or 8. If set to 1 then all bytes are used (i.e. no bytes are wasted at end of row). If set to 2, then if necessary, an extra wasted byte(s) is/are stored at the end of the row so that the first byte of the next row has an address that is a multiple of 2 (multiple of four for Row Alignment equal 4 and multiple of 8 for row alignment equal 8). The actual formula (using C syntax) to determine number of bytes per row is as follows:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

BytesPerRow = (numBytesPerPixel * ImageWidth + RowAlignment – 1) & ~(RowAlignment – 1)

I16 : Width

Width specifies the width dimension (number of texel columns) of the texture image in number of pixels.

I16 : Height

Height specifies the height dimension (number of texel rows) of the texture image in number of pixels. Height is 1 for one-dimensional images.

I16 : Depth

Depth specifies the depth dimension (number of texel slices) of the texture image in number of pixels. Depth is 1 for one-dimensional and two-dimensional images.

I16 : Number Border Texels

Number Border Texels specifies the number of border texels in the texture image definition. Valid values are 0 and 1.

U8 : Shared Image Flag

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

Table 34 — Texture Vers-1 Image Format Description Shared Image Flag values

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

I16 : Mipmaps Count

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

5.1.2.3.2 Texture Vers-2 Data

Texture Vers-2 Data collection supports texturing effects not representable in the Texture Vers-1 Data format (e.g. more precise texture types, automatic texture channel, etc.). Any Texture Image Attribute Element using the Texture Vers-2 Data format will contain a “degenerate” Texture Vers-1 Data block, where Image Count data field has a value of “0”.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

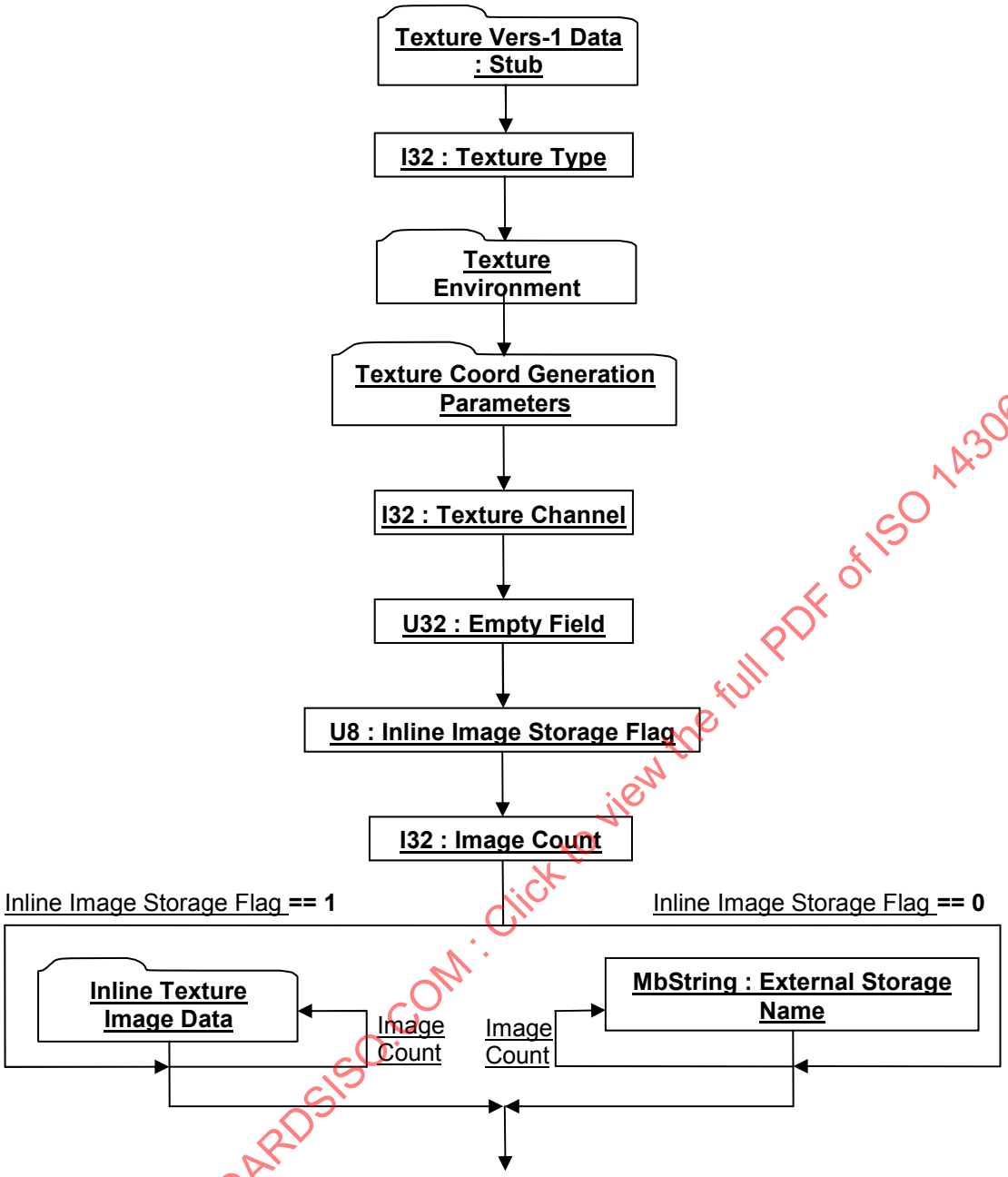


Figure 56 — Texture Vers-2 Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

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

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

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

Texture Vers-1 Data : Stub

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

© ISO 2012 – All rights reserved

I32 : Texture Type

Texture Type specifies the type of texture. There is a complete restructuring and redefinition of what a “texture type” implies in Texture Vers-2 Data. It is a much stronger concept now, that not only describes generally what the texture image contains, but also defines precisely what the texture is being used for. In the following list, “image” refers to an image texture, “pre-lit” indicates that the image texture is to be applied before lighting when rendering the object to which it is applied, and “post-lit” indicates that the image texture is to be applied after lighting. A gloss map is a pre-lit texture that applies itself to the specular material component of lighting instead of the diffuse component. A light map is an environment texture (texture at infinity surrounding the whole model) that serves as a source of illumination during shading calculations.

Table 35 — Texture Vers-2 Texture Type values

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

I32 : Texture Channel

Texture Channel specifies the texture channel number for the Texture Image Element. For purposes of multi-texturing, the JT concept of a texture channel corresponds to the OpenGL concept of a “texture unit.” The Texture Channel value shall be between -1 and 31 inclusive. The value -1 is accepted to denote a texture whose channel number is to be automatically assigned. This assignment will never displace another texture with an explicit texture channel assignment from its slot. Best practices suggest that renderer of JT data ignore all but channel-0 if the renderer does not support multi-textured geometry. Also for purposes of blending, any renderer of JT data should ensure that higher numbered texture channels “blend over” lower numbered ones.

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

U32 : Empty Field

Refer to best practice 14.4 Empty Field

© ISO 2012 – All rights reserved

U8 : Inline Image Storage Flag

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

Table 36 — Texture Vers-2 Inline Image Storage Flag values

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

I32 : Image Count

Image Count specifies the number of texture images. A “Cube Map” I32 : Texture Type shall have six images while all other Texture Types may only have one image.

MbString : External Storage Name

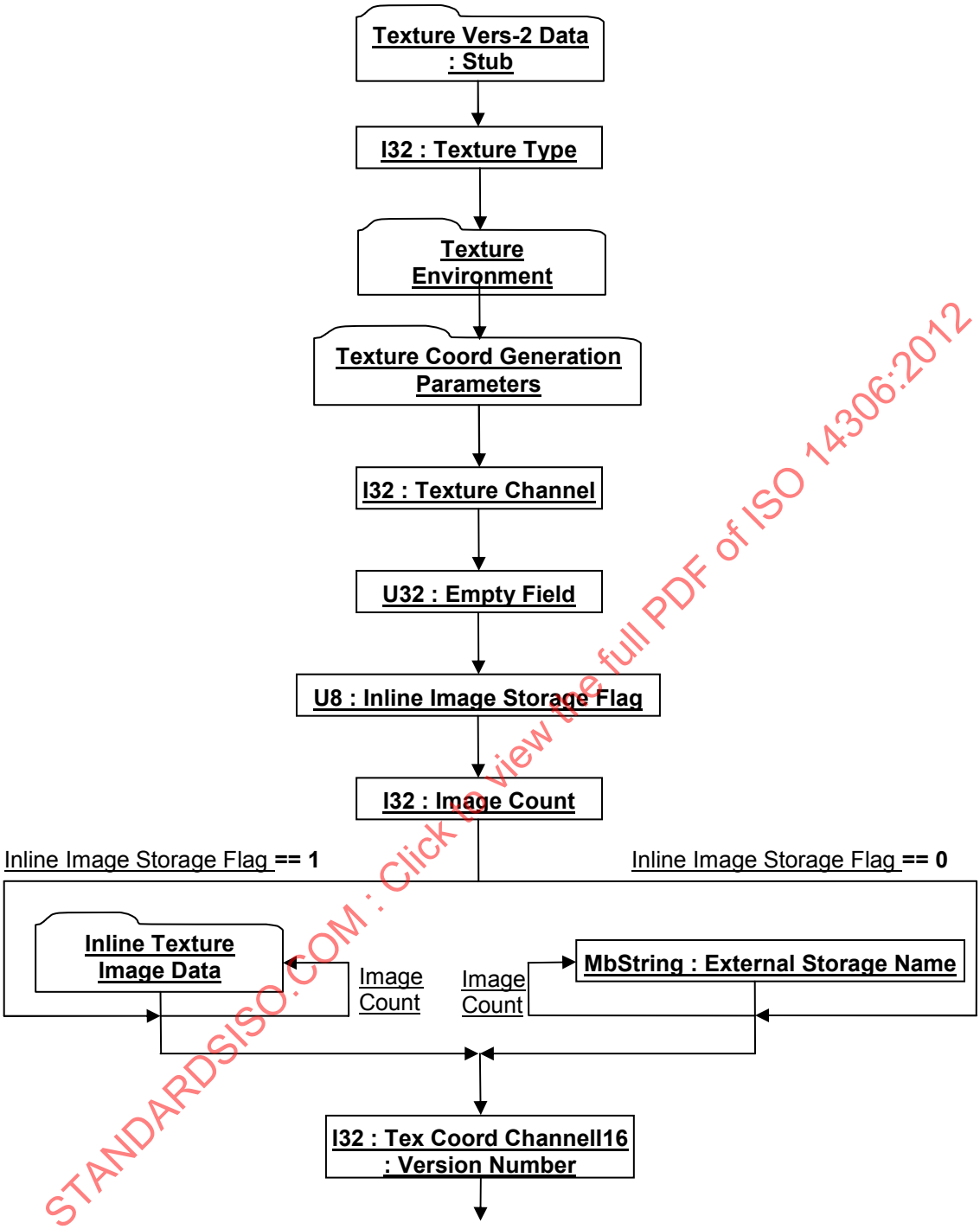
External Storage Name is a string identifying the name of an external texture image storage. External Storage Name is only present if data field Inline Image Storage Flag equals 0. If present, there will be data field Image Count number of External Storage Name instances. This External Storage Name string is a relative path based name for the texture image file. Where “relative path” should be interpreted to mean the string contains the file name along with any additional path information that locates the texture image file relative to the location of the referencing JT file.

5.1.2.3.3 Texture Vers-3 Data

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 57 — Texture Vers-3 Data collection

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

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

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

© ISO 2012 – All rights reserved

Texture Vers-2 Data : Stub

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

I32 : Texture Type

Texture Type specifies the type of texture. A new texture type, separator texture, is defined in Texture Vers-3 Data to support resetting the texture accumulation state mid-graph. Shadow maps and prefiltered light maps, however, are a general exception to this rule. In the following list, “image” refers to an image texture, “pre-lit” indicates that the image texture is to be applied before lighting when rendering the object to which it is applied, and “post-lit” indicates that the image texture is to be applied after lighting. A gloss map is a pre-lit texture that applies itself to the specular material component of lighting instead of the diffuse component. A light map is an environment texture (texture at infinity surrounding the whole model) that serves as a source of illumination during shading calculations.

Table 37 — Texture Vers-3 Texture Type values

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

I32 : Texture Channel

Texture Channel specifies the texture channel number for the Texture Image Element. For purposes of multi-texturing, the JT concept of a texture channel corresponds to the OpenGL concept of a “texture unit.” The Texture Channel value shall be between -1 and 2,147,483,647 inclusive. The value -1 is accepted to denote a texture whose channel number is to be automatically assigned. This assignment will never displace another texture with an explicit texture channel assignment from its slot. Best practices suggest that renderer of JT data ignore all but channel-0 if the renderer does not support multi-textured geometry. Also for purposes of blending, any renderer of JT data should ensure that higher numbered texture channels “blend over” lower numbered ones.

© ISO 2012 – All rights reserved

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

U32 : Empty Field

Refer to best practice 14.4 Empty Field

U8 : Inline Image Storage Flag

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

Table 38 — Texture Vers-3 Inline Image Storage Flag values

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

I32 : Image Count

Image Count specifies the number of texture images. A "Cube Map" I32 : Texture Type shall have six images while all other Texture Types should only have one image.

MbString : External Storage Name

External Storage Name is a string identifying the name of an external texture image storage. External Storage Name is only present if data field Inline Image Storage Flag equals "0." If present there will be data field Image Count number of External Storage Name instances. This External Storage Name string is a relative path based name for the texture image file. Where "relative path" should be interpreted to mean the string contains the file name along with any additional path information that locates the texture image file relative to the location of the referencing JT file.

I32 : Tex Coord Channel

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

5.1.2.4 Draw Style Attribute Element

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

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

The Field Inhibit flag (see 5.1.2.1.1Base Attribute Data) bit assignments for the Draw Style Attribute Element data fields, are as follows:

Table 39 — Draw Style Attribute Field Inhibit flag values

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

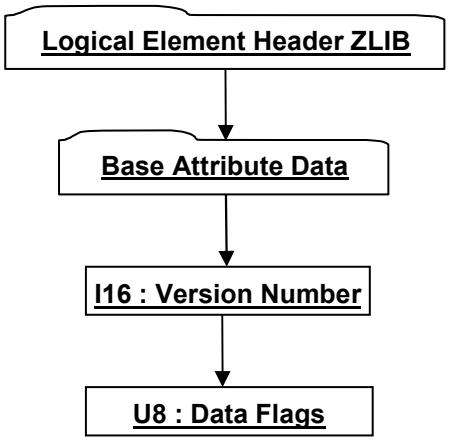


Figure 58 — Draw Style Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

I16 : Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers.

U8 : Data Flags

Data Flags is a collection of flags. The flags are combined using the binary OR operator and store various state settings for Draw Style Attribute Elements. All bits fields that are not defined as in use should be set to "0".

Table 40 — Draw Style Attribute Data Flag values

0x01	Back-face Culling Flag. Indicates if back-facing polygons should be discarded (culled). = 0 – Back-facing polygons not culled. = 1 – Back-facing polygons culled.
0x02	Two Sided Lighting Flag. Indicates if two sided lighting should be enabled to insure that polygons are illuminated on both sides. = 0 – Disable two sided lighting. = 1 – Enable two sided lighting.
0x04	Outlined Polygons Flag. Indicates if polygons should be draw as "wireframes" i.e. not filled. = 0 – Polygons drawn as filled. = 1 – Only polygon's outline drawn.
0x08	Lighting Enabled Flag. Indicates if lighting should be enabled. If lighting disabled, then renderer should perform no calculations concerning normals, light sources, material properties, etc. = 0 – Disable lighting. = 1 – Enable lighting.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

0x10	Flat Shading Flag. Indicates if the geometry should be rendered with single colour (flat shading) or with many different colour (smooth/Gouraud) shading. = 0 – Disable flat shading (i.e. use smooth/Gouraud shading). = 1 – Enable flat shading.
0x20	Separate Specular Flag. Indicates if the application of the specular colour should be delayed until after texturing. If no texture mapping then this flag setting is irrelevant. = 0 – Apply specular colour contribution before texture mapping. = 1 – Apply specular colour contribution after texture mapping.

5.1.2.5 Light Set Attribute Element

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

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

Light Set Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

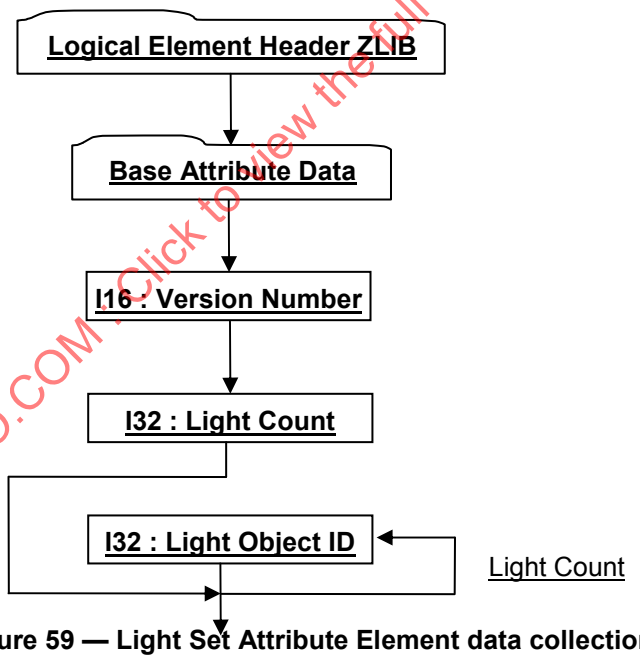


Figure 59 — Light Set Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

I16 : Version Number

Version Number is the version identifier for this element. For information on local version numbers see best practice 14.5 Local version numbers

I32 : Light Count

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I32 : Light Object ID

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

5.1.2.6 Infinite Light Attribute Element

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

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

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

Infinite Light Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

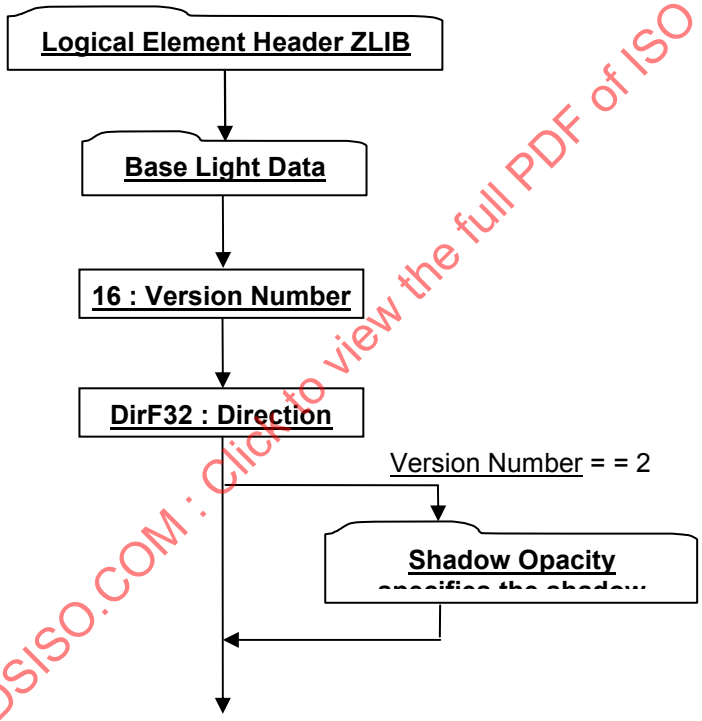


Figure 60 — Infinite Light Attribute Element data collection

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

Complete description for Base Light Data can be found in 5.1.2.6.1 Base Light Data.

Complete description for Shadow Parameters can be found in 5.1.2.6.2 Shadow Parameters.

16 : Version Number

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 41 — Light Set Attribute Version Number values

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

DirF32 : Direction

Direction specifies the direction the light is pointing in.

5.1.2.6.1 Base Light Data

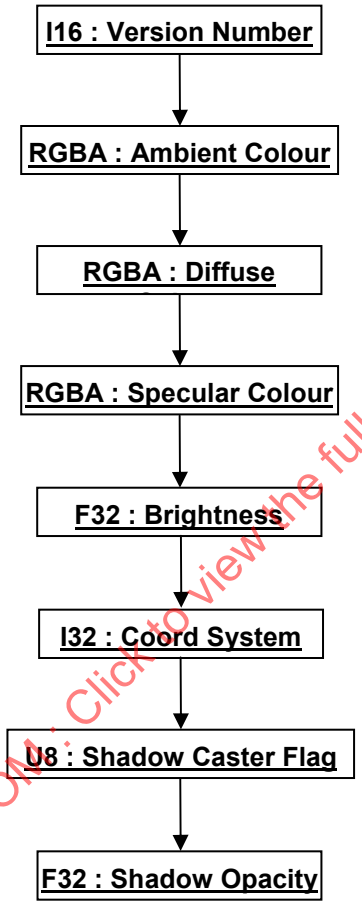


Figure 61 — Base Light Data collection

I16 : Version Number

Version number is the version identifier for this element. For information on local version numbers see best practice 14.5 Local version numbers

RGBA : Ambient Colour

Ambient Colour specifies the ambient red, green, blue, alpha colour values of the light.

RGBA : Diffuse Colour

Diffuse Colour specifies the diffuse red, green, blue, alpha colour values of the light.

RGBA : Specular Colour

Specular Colour specifies the specular red, green, blue, alpha colour values of the light.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

F32 : Brightness

Brightness specifies the Light brightness. The Brightness value shall be greater than or equal to “-1”.

I32 : Coord System

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

Table 42 — Base Light Data Cood System values

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

U8 : Shadow Caster Flag

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

Table 43 — Base Light Data Shadow Caster Flag values

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

F32 : Shadow Opacity

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

5.1.2.6.2 Shadow Parameters

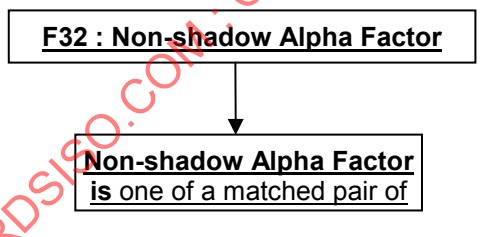


Figure 62 — Shadow Parameters data collection

F32 : Non-shadow Alpha Factor

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

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

F32 : Shadow Alpha Factor

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

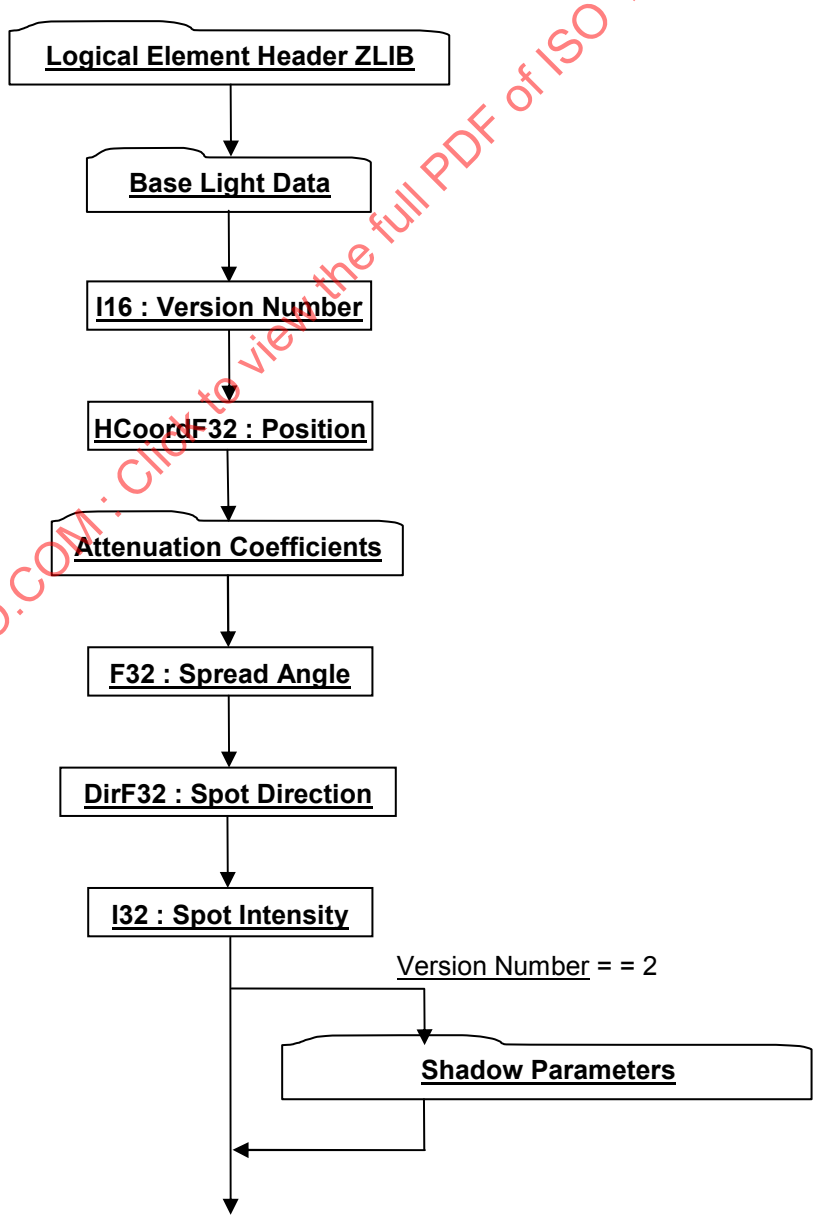
5.1.2.7 Point Light Attribute Element

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

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

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

Point Light Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 63 — Point Light Attribute ElementPoint Light Attribute Element data collection

© ISO 2012 – All rights reserved

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

Complete description for Base Light Data can be found in 5.1.2.6.1 Base Light Data.

Complete description for Attenuation Coefficients can be found in 5.1.2.7.1 Attenuation Coefficients.

Complete description for Shadow Parameters can be found in 5.1.2.6.2 Shadow Parameters.

I16 : Version Number

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

Table 44 — Point Light Attribute Version Number values

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

HCoordF32 : Position

Position specifies the light position in homogeneous coordinates.

F32 : Spread Angle

Spread Angle, as shown in Figure 64 — Spread Angle value with respect to the light cone below, specifies in degrees the half angle of the light cone. Valid Spread Angle values are clamped and interpreted as follows:

Table 45 — Point Light Attribute Spread Angle values

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

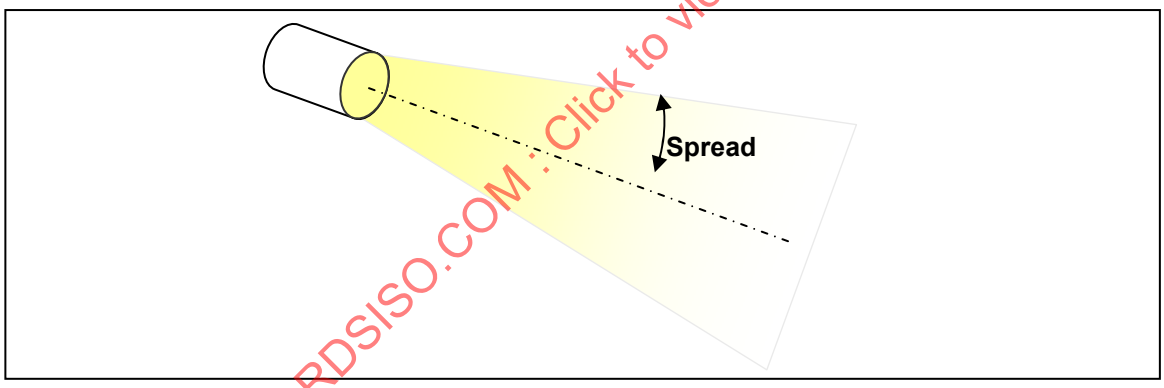


Figure 64 — Spread Angle value with respect to the light cone

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

I32 : Spot Intensity

Spot Intensity specifies the intensity distribution of the light within the spot light cone. Spot Intensity is really a "spot exponent" in a lighting equation and indicates how focused the light is at the centre. The larger the value, the more focused the light source. Only non-negative Spot intensity values are valid.

5.1.2.7.1 Attenuation Coefficients

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

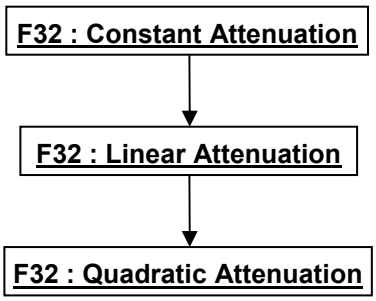


Figure 65 — Attenuation Coefficients data collection

F32 : Constant Attenuation

Constant Attenuation specifies the constant coefficient for how light intensity decreases with distance. Value shall be greater than or equal to “0”.

F32 : Linear Attenuation

Linear Attenuation specifies the linear coefficient for how light intensity decreases with distance. Value shall be greater than or equal to “0”.

F32 : Quadratic Attenuation

Quadratic Attenuation specifies the quadratic coefficient for how light intensity decreases with distance. Value shall be greater than or equal to “0”.

5.1.2.8 Linestyle Attribute Element

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

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

Linestyle Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

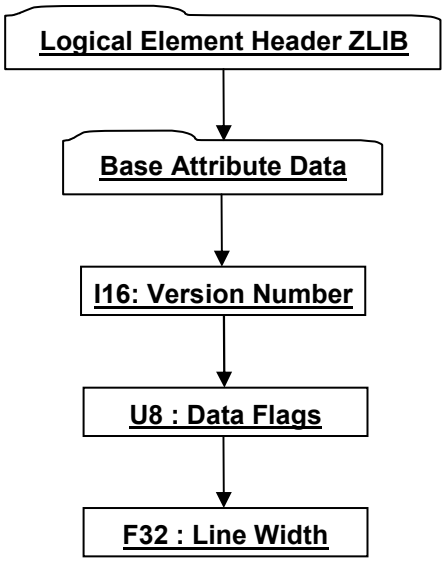


Figure 66 — Linestyle Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers

U8 : Data Flags

Data Flags is a collection of flags and line type data. The flags and line type data are combined using the binary OR operator and store various polyline rendering attributes. All bits fields that are not defined as in use should be set to "0".

Table 46 — Linestyle Attribute Data Flag values

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

F32 : Line Width

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

5.1.2.9 Pointstyle Attribute Element

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

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

Pointstyle Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

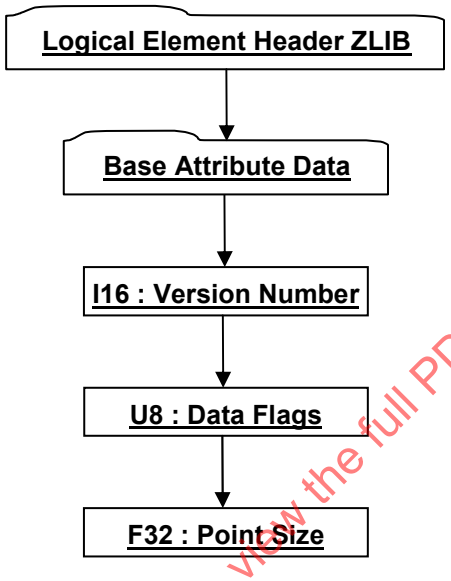


Figure 67 — Pointstyle Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

I16 : Version Number

Version Number is the version identifier for this element. For information on local version numbers see best practice 14.5 Local version numbers

U8 : Data Flags

Data Flags is a collection of flags and point type data. The flags and point type data are combined using the binary OR operator and store various point rendering attributes. All bits fields that are not defined as in use should be set to “0”.

Table 47 — Pointstyle Attribute Data Flag values

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

F32 : Point Size

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

5.1.2.10 Geometric Transform Attribute Element

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

Geometric Transform Attribute Element contains a 4x4 homogeneous transformation matrix that positions the associated LSG node's coordinate system relative to its parent LSG node. JT format LSG traversal semantics state that geometric transform attributes accumulate down the LSG through matrix multiplication as follows:

$$p' = pAM$$

Where *p* is a point of the model, *p'* is the transformed point, *M* is the current modeling transformation matrix inherited from ancestor LSG nodes and previous Geometric Transform Attribute Element, and *A* is the transformation matrix of this Geometric Transform Attribute Element. The matrix is allowed to contain translation, rotation, and uniform- and non-uniform scaling factors, including negative scales. It is not allowed to contain shearing or projective components, or scaling factors of zero (which would make the matrix singular).

Geometric Transform Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

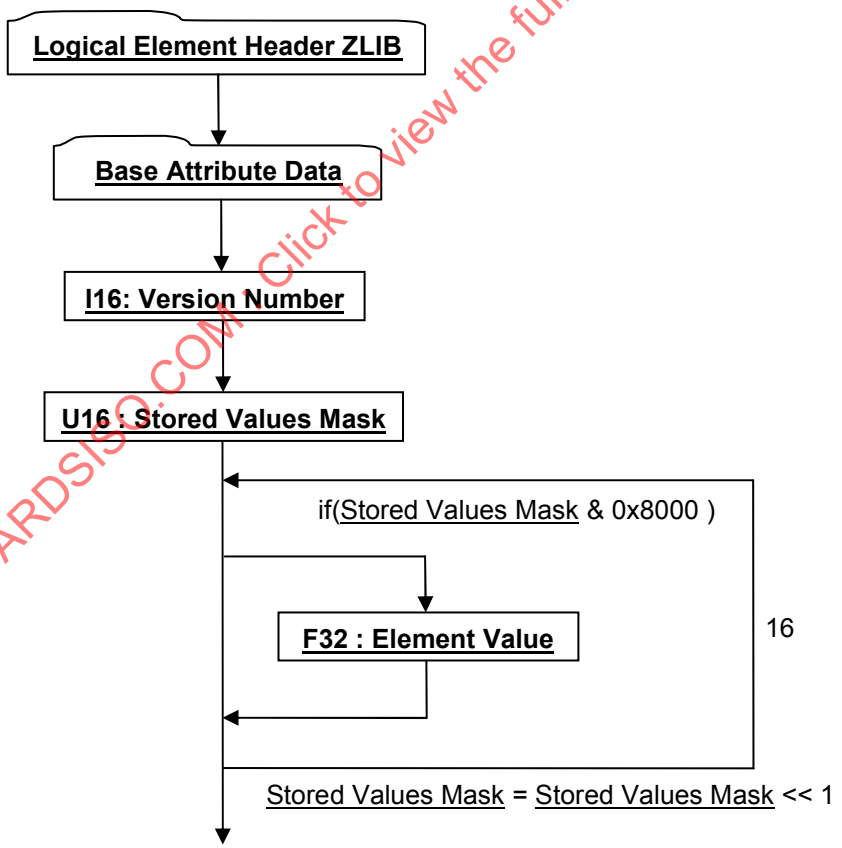


Figure 68 — Geometric Transform Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16: Version Number

Version Number is the version identifier for this node. For information on local version numbers see best practice 14.5 Local version numbers

U16 : Stored Values Mask

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

Bit15 Bit14 Bit13 Bit12

Bit11 Bit10 Bit9 Bit8

Bit7 Bit6 Bit5 Bit4

Bit3 Bit2 Bit1 Bit0

The individual bit-flag values are interpreted as follows:

Table 48 — Geometric Transform Attribute Stored Value Mask individual bit-flag values

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

F32 : Element Value

Element Value specifies a particular matrix element value.

5.1.2.11 Shader Effects Attribute Element

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

Shader Effects Attribute Element contains information specifying “high-level” shader functionality (e.g. Phong shading, bump mapping, etc.) that should be used for rendering the geometry this attribute element is associated with.

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

Shader Effects Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

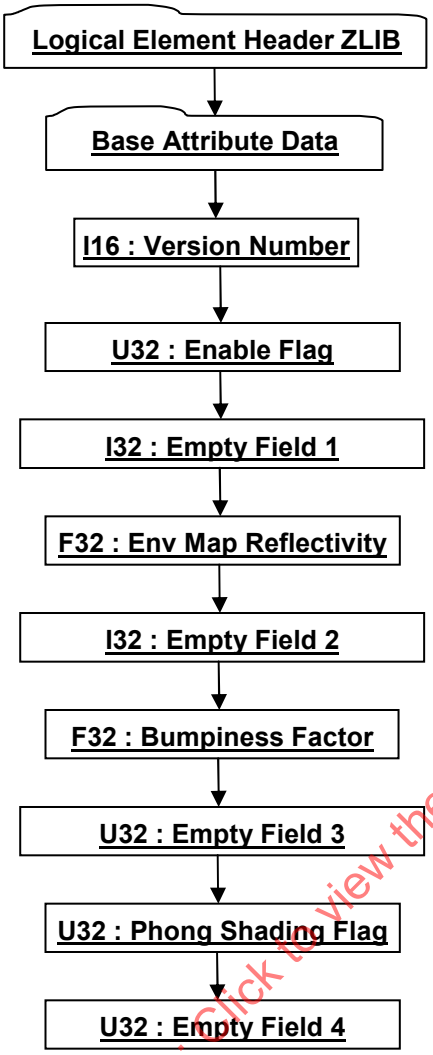


Figure 69 — Shader Effects Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1 Base Attribute Data.

I16 : Version Number

Version Number is the version identifier for this element. For information on local version numbers see best practice 14.5 Local version numbers

U32 : Enable Flag

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

Table 49 — Shader Effects Attribute Enable Flag values

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

I32 : Empty Field 1

Refer to best practice 14.4 Empty Field

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

F32 : Env Map Reflectivity

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

I32 : Empty Field 2

Refer to best practice 14.4 Empty Field

F32 : Bumpiness Factor

Bumpiness Factor specifies the degree of “bumpiness”, or the relative “height” of the bump map. Larger values make the bumps appear deep and more severe. Negative values invert the sense of the bump map, making the surface appear engraved, rather than embossed. This value only has an effect with tangent space bump maps.; it has no effect on the appearance of object space bump maps.

U32 : Empty Field 3

Refer to best practice 14.4 Empty Field

U32 : Phong Shading Flag

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

Table 50 — Shader Effects Attribute Phong Shading Flag values

= 0	Phong Shading disabled
= 1	Phong Shading enabled

U32 : Empty Field 4

Refer to best practice 14.4 Empty Field

5.1.2.12 Texture Coordinate Generator Attribute Element

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

Texture Coordinate Generator Attribute Element defines texture coordinate generation for texture mapping. Multiple texture coordinate generation at a given node is supported by way of the “texture coordinate channel” concept. JT format LSG traversal semantics state that Texture Coordinate Generator attributes accumulate down the LSG by replacement on a per-channel basis.

Texture Coordinate Generator Attribute Element does not have any Field Inhibit flag (see 5.1.2.1.1 Base Attribute Data) bit assignments.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

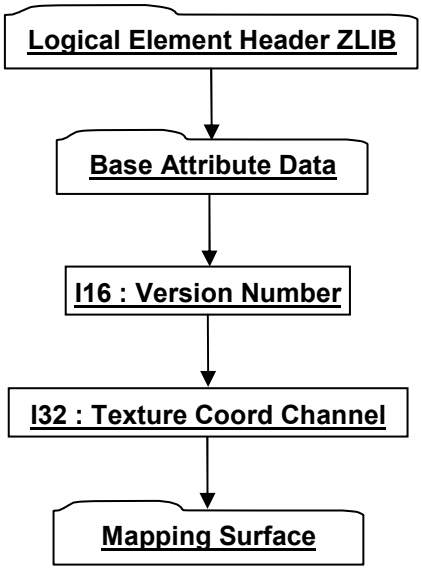


Figure 70 — Texture Coordinate Generator Attribute Element data collection

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

Complete description for Base Attribute Data can be found in 5.1.2.1.1Base Attribute Data.

Complete description for Mapping Surface can be found in 5.1.2.12.1Mapping Surface.

I16 : Version Number

Version Number is the version identifier for this element. For information on local version numbers see best practice 14.5 Local version numbers

I32 : Texture Coord Channel

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

5.1.2.12.1 Mapping Surface

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

Mapping TriPlanar Element, are defined to support texture coordinate generation.

Mapping Plane Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

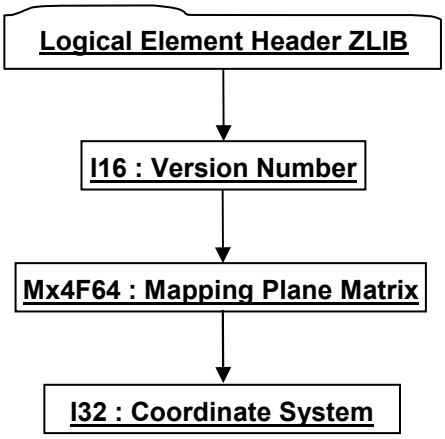


Figure 71 — Mapping Plane Element data collection

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

I16 : Version Number

Version Number is the version identifier for this element. For informaiton on local version numbers see best practice 14.5 Local version numbers

Mx4F64 : Mapping Plane Matrix

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

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

I32 : Coordinate System

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

Table 51 — Mapping Plane Matrix Coordinate System values

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

Mapping Cylinder Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

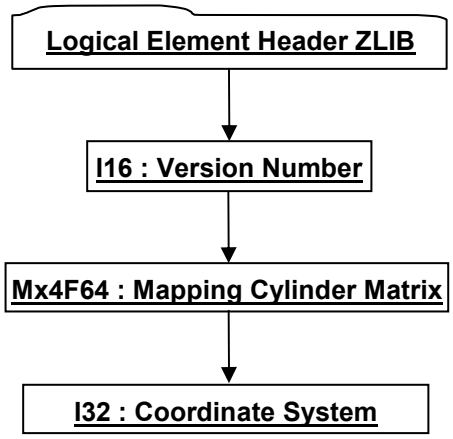


Figure 72 — Mapping Cylinder Element data collection

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

I16 : Version Number

Version Number is the version identifier for this element. For informaiton on local version numbers see best practice 14.5 Local version numbers

Mx4F64 : Mapping Cylinder Matrix

Mx4F64 : Mapping Cylinder Matrix specifies the transformation matrix and mapping parameters for the mapping cylinder. The transformation matrix defines the mapping coordinate system transformed from I32 : Coordinate System. The mapping parameters specifies the horizontal sweep angle and height of the mapping cylinder. The mapping cylinder's axis is parallel to the z-axis of the mapping coordinate system, and the horizontal sweep angle starts from the +x-axis in a counter clockwise direction. In the mapping process, the geometry vertex coordinates in Model Coordinate System are transformed to the mapping coordinate system at first, and then the transformed vertex coordinates are mapped to texture coordinates as following:

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

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

I32 : Coordinate System

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

Table 52 — Mapping Cylinder Matrix Coordinate System values

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

Mapping Sphere Element

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

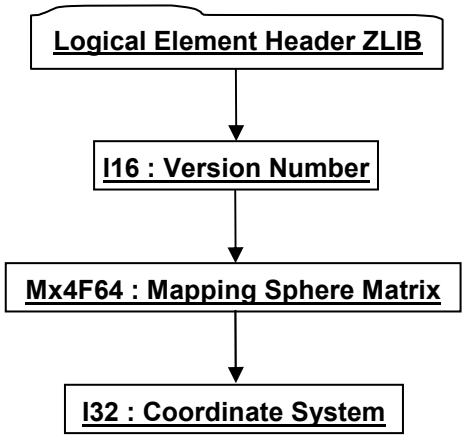


Figure 73 — Mapping Sphere Element data collection

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

I16 : Version Number

Version Number is the version identifier for this element. For informaiton on local version numbers see best practice 14.5 Local version numbers

Mx4F64 : Mapping Sphere Matrix

Mx4F64 : Mapping Sphere Matrix specifies the transformation matrix and mapping parameters of the mapping sphere. The transformation matrix defines the mapping coordinate system transformed from I32 : Coordinate System. The mapping parameters specify the horizontal sweep angle and vertical sweep angle of the mapping sphere. The mapping sphere's centre is at the origin of the mapping coordinate system, and the poles of the sphere are parallel to the z-axis of the coordinate system. The horizontal sweep angle starts from the +x-axis in a counter clockwise direction, and the vertical sweep angle is from the +z-axis to the –z-axis. In the mapping process, the geometric vertex coordinates in Model Coordinate System are transformed to the mapping coordinate system at first, and then the transformed vertex coordinates are mapped to texture coordinates as following:

s-coordinate = the horizontal sweep angle of the vertex / the horizontal sweep angle of the mapping sphere

t-coordinate = the vertical sweep angle of the vertex / the vertical sweep angle of the mapping sphere

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

I32 : Coordinate System

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

Table 53 — Mapping Sphere Matrix Coordinate System values

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Mapping TriPlanar Element

Object Type ID: 0x92f5b094, 0x6499, 0x4d2d, 0x92, 0xaa, 0x60, 0xd0, 0x5a, 0x44, 0x32, 0xcf

Mapping TriPlanar Element defines the mapping triplanar surface for texture coordinate generation.

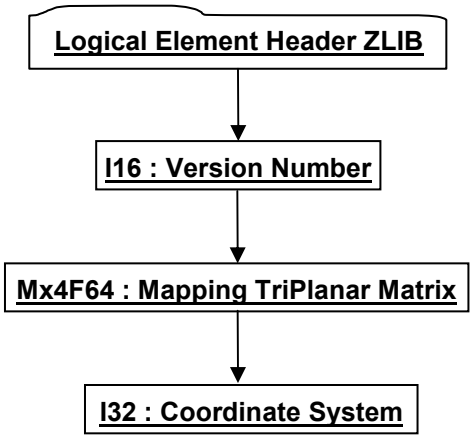


Figure 74 —

Mapping TriPlanar Element data collection

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

I16 : Version Number

Version Number is the version identifier for this element. For informaiton on local version numbers see best practice 14.5 Local version numbers

Mx4F64 : Mapping TriPlanar Matrix

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

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

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

I32 : Coordinate System

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

Table 54 — Mapping TriPlanar Matrix Coordinate System values

= 0	Undefined Coordinate System.
= 1	Viewpoint Coordinate System. Mapping triplanar surface is to move together with the viewpoint.
= 2	Model Coordinate System. Mapping triplanar surface is affected by whatever model transforms that are current when the mapping triplanar surface is encountered in LSG.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

= 3	World Coordinate system. Mapping triplanar surface is not affected by model transforms in the LSG.

5.2 Property Atom Elements

Property Atom Elements are meta-data objects associated with nodes or Attributes. Property Atom Elements are not nodes or attributes themselves, but can be associated with any node or Attribute to maintain arbitrary application- or enterprise information (meta-data) pertaining to that node or Attribute. Each Node Element or Attribute Element in an LSG may hold zero or more Property Atom Elements and this relationship information is stored within 5.3 Property Table section of a JT file.

An individual property is specified as a *key/value* Property Atom Element pair, where the *key* identifies the type and meaning of the *value*. The JT format supports many different Property Atom Element key/value object types. The different Property Atom Element key/value object types are documented in the following subsections.

Some “Best Practices” for placing application or enterprise properties/meta-data on Nodes in JT files can be found in 14.8 Metadata Conventions section of this reference.

5.2.1 Base Property Atom Element

Object Type ID: 0x10dd104b, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Base Property Atom Element represents the simplest form of a property that can exist within the LSG and has no type specific value data associated with it.

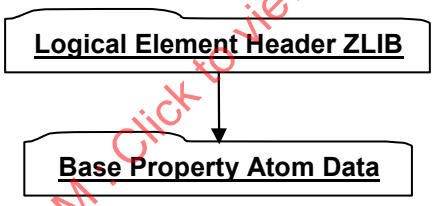


Figure 75 — Base Property Atom Element data collection

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

Base Property Atom Data

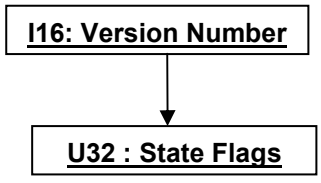


Figure 76 — Base Property Atom Data collection

I16: Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

U32 : State Flags

State Flags is a collection of flags. The flags are combined using the binary OR operator and store various state information for property atoms. Bits 0 – 7 are freely available for an application to store whatever property atom information desired. All other bits are reserved for future expansion of the file format.

5.2.2 String Property Atom Element

Object Type ID: 0x10dd106e, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

String Property Atom Element represents a character string property atom.

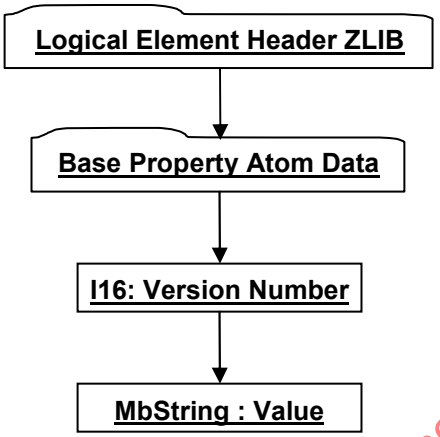


Figure 77 — String Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16: Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

MbString : Value

Value contains the character string value for this property atom.

5.2.3 Integer Property Atom Element

Object Type ID: 0x10dd102b, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Integer Property Atom Element represents a property atom whose value is of I32 data type.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

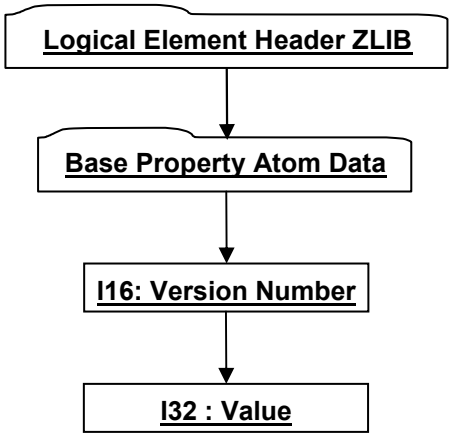


Figure 78 — Integer Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16: Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : Value

Value contains the integer value for this property atom.

5.2.4 Floating Point Property Atom Element

Object Type ID: 0x10dd1019, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Floating Point Property Atom Element represents a property atom whose value is of F32 data type.

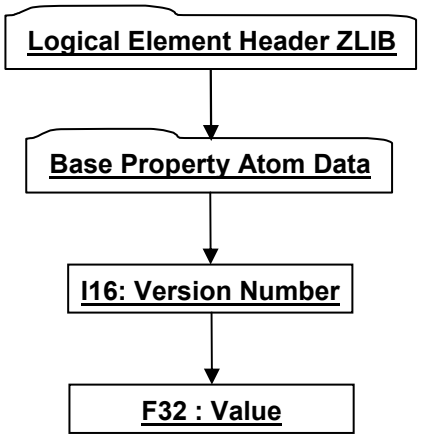


Figure 79 — Floating Point Property Atom Element data collection

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16: Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

F32 : Value

Value contains the floating point value for this property atom.

5.2.5 JT Object Reference Property Atom Element

Object Type ID: 0x10dd1004, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

JT Object Reference Property Atom Element represents a property atom whose value is an object ID for another object within the JT file.

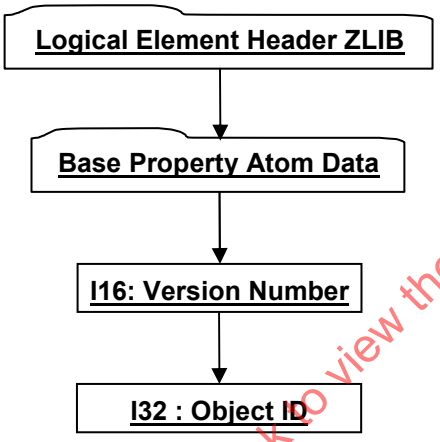


Figure 80 — JT Object Reference Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16: Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : Object ID

Object ID specifies the identifier within the JT file for the referenced object.

5.2.6 Date Property Atom Element

Object Type ID: 0xce357246, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1

Date Property Atom Element represents a property atom whose value is a “date”.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

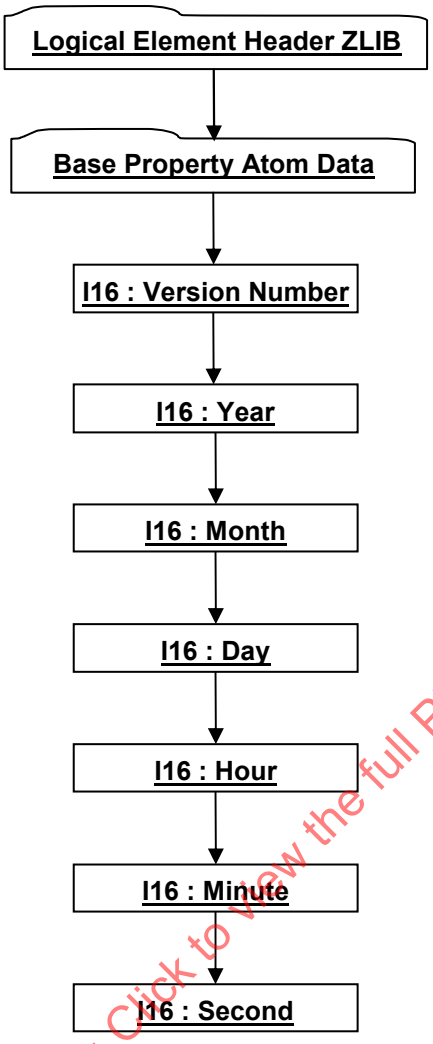


Figure 81 — Date Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16 : Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

I16 : Year

Year specifies the date year value. Valid values are [1900, 2999] inclusive.

I16 : Month

Month specifies the date month value. Valid values are [0, 11] inclusive.

I16 : Day

Day specifies the date day value. Valid values are [1, 31] inclusive.

I16 : Hour

Hour specifies the date hour value. Valid values are [0, 23] inclusive.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Minute

Minute specifies the date minute value. Valid values are [0, 59] inclusive.

I16 : Second

Second specifies the date Second value. Valid values are [0, 59] inclusive.

5.2.7 Late Loaded Property Atom Element

Object Type ID: 0xe0b05be5, 0xfbbd, 0x11d1, 0xa3, 0xa7, 0x00, 0xaa, 0x00, 0xd1, 0x09, 0x54

Late Loaded Property Atom Element is a property atom type used to reference an associated piece of atomic data in a separate addressable segment of the JT file. The "Late Loaded" connotation derives from the associated data being stored in a separate addressable segment of the JT file, and thus a JT file reader can be structured to support the "best practice" of delaying the loading/reading of the associated data until it is actually needed.

Late Loaded Property Atom Elements are used to store a variety of data, including, but not limited to, Shape LOD Segments and B-Rep Segments (see 6 Shape LOD Element

and 7 JT B-Rep Segment).

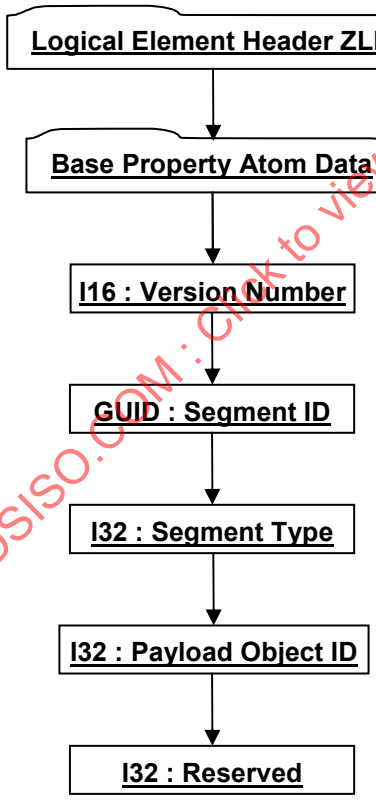


Figure 82 — Late Loaded Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16 : Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

GUID : Segment ID

Segment ID is the globally unique identifier for the associated data segment in the JT file. See 4.1.2 TOC Segment for additional information on how this Segment ID can be used in conjunction with the file TOC Entries to locate the associated data in the JT file.

The complete list of segment types can be found Table 8 — Segment Types.

I32 : Segment Type

Segment Type defines a broad classification of the associated data 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.

I32 : Payload Object ID

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

I32 : Reserved

Reserved data field that is guaranteed to always be greater than or equal to 1

5.2.8 Vector4f Property Atom Element

Object Type ID: 0x2e7db4be, 0xc71a, 0x4b18, 0x9d, 0x7, 0xc7, 0x22, 0x7e, 0x9f, 0xef, 0x76

Vector4f Property Atom Element represents a property atom whose value is of VecF32 data type with the length to be equal to 4 .

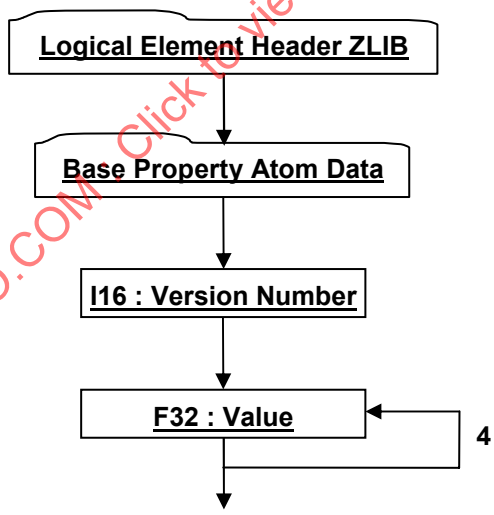


Figure 83 — Vector4f Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in Base Property Atom Data.

I16 : Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

F32 : Value

Value contains the floating point value for this property atom

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

5.3 Property Table

The Property Table is where the data connecting Node Elements and Attribute Elements with their associated Properties is stored. The Property Table contains an Element Property Table for each element in the JT File which has associated Properties. An Element Property Table is a list of key/value Property Atom Element pairs for all Properties associated with a particular Node Element Object or Attribute Element Object.

For a reference compliant JT File all Node Elements, Attribute Elements, and Property Atom Elements contained in a JT file should have been read by the time a JT file reader reaches the Property Table section of the file. This means that all Node Objects, Attribute Objects, and Property Atom Objects referenced in the Property Table (through Object IDs), should have already been read, and if not, then the file is corrupt (i.e. not reference compliant).

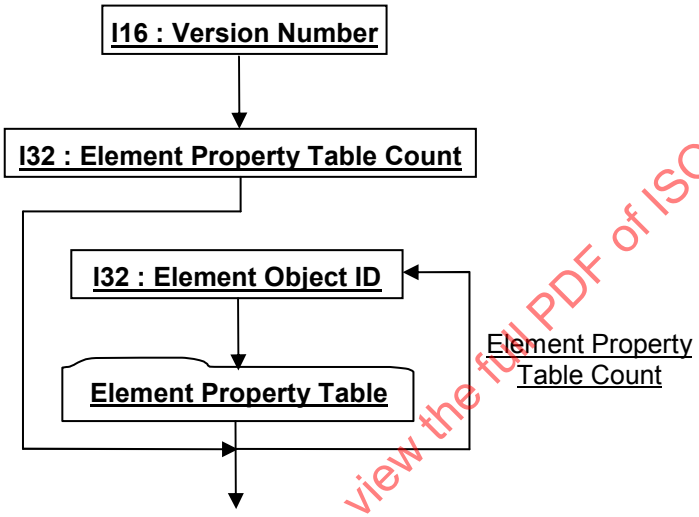


Figure 84 — Property Table data collection

I16 : Version Number

Version Number is the version identifier for this Property Table. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : Element Property Table Count

Element Property Table Count specifies the number of Element Property Tables to follow. This value is equivalent to the total number of Node Elements (see 5.1.1 Node Elements) and Attribute Elements (see 5.1.2 Attribute Elements) that have associated Property Atom Elements (see Property Atom Elements).

I32 : Element Object ID

Element Object ID is the identifier for the Node Element object (see 5.1.1 Node Elements) or the Attribute Element object (see 5.1.2 Attribute Elements) that the following Element Property Table is for (i.e. Node Element or Attribute Element that all properties in the following Element Property Table are associated with).

5.3.1 Element Property Table

The Element Property Table is a list of key/value Property Atom Element pairs for all properties associated with a particular Node Element Object or Attribute Element Object. The list is terminated by a “0” value for Key Property Atom Object ID.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

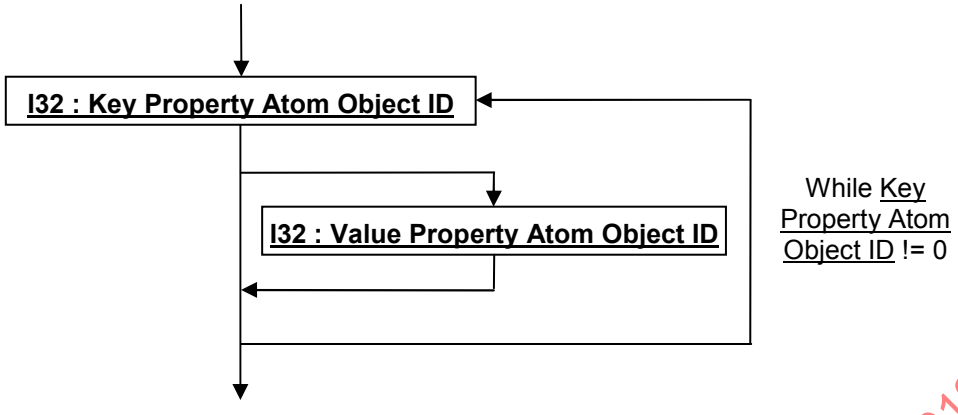


Figure 85 — Element Property Table data collection

I32 : Key Property Atom Object ID

Key Property Atom Object ID is the identifier for the Property Atom Element object (see Property Atom Elements) representing the "key" part of the property key/value pair. A value of "0" indicates the end of the Node Property Table.

I32 : Value Property Atom Object ID

Value Property Atom Object ID is the identifier for the Property Atom Element object (see Property Atom Elements) representing the "value" part of the property key/value pair. A value is not stored if Key Property Atom Object ID has a value of "0".

6 Shape LOD Segment

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. Shape LOD Segments are typically referenced by Shape Node Elements using Late Loaded Property Atom Elements (see 5.1.1.10 Shape Node Elements and 5.2.7 Late Loaded Property Atom Element respectively).

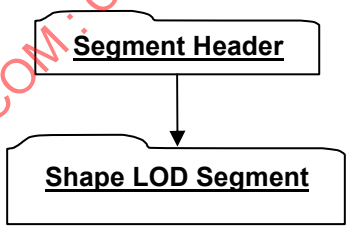


Figure 86 — Shape LOD Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

6.1 Shape LOD Element

A Shape LOD Element is the holder/container of the geometric shape definition data (e.g. vertices, polygons, normals, etc.) for a single LOD. Much of the "heavyweight" data contained within a Shape LOD Element may be optionally compressed and/or encoded. The compression and/or encoding state is indicated through other data stored in each Shape LOD Element.

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

6.1.1 Base Shape LOD Element

Object Type ID: 0x10dd10a4, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Base Shape LOD Element serves as the underlying representation for all LODs.

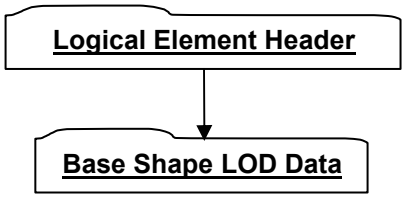


Figure 87 — Base Shape LOD Element data collection

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

6.1.1.1 Base Shape LOD Data

Base shape LOD data contains the common items to all shape LODs.

Figure 88 — Base Shape LOD Data collection

I16 : Version Number

Version Number is the version identifier for this Base Shape LOD Data. For informaiton on local version numbers see best practice 14.5 Local version numbers

6.1.2 Vertex Shape LOD Element

Object Type ID: 0x10dd10b0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

Vertex Shape LOD Element represents LODs defined by collections of vertices.

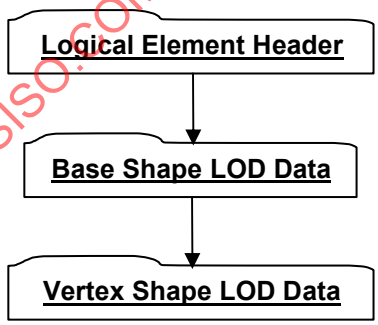


Figure 89 — Vertex Shape LOD Element data collection

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

6.1.2.1 Vertex Shape LOD Data

Vertex Shape LOD Data collection is an abstract container for geometric *primitives* such as triangle strips, line strips, or points, depending on the specific type of Vertex Shape. The set of primitives are further partitioned into so-called "face groups." The Vertex Shape LOD Data also contains the vertex attribute bindings and quantization settings used to store the vertex records referenced by the primitives.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

One use for face groups is to establish a correspondence between Brep faces and their triangle representation. A convention for mapping JTBrep and XTBrep faces to face groups is described in section 14.12 Brep Face Group Associations.

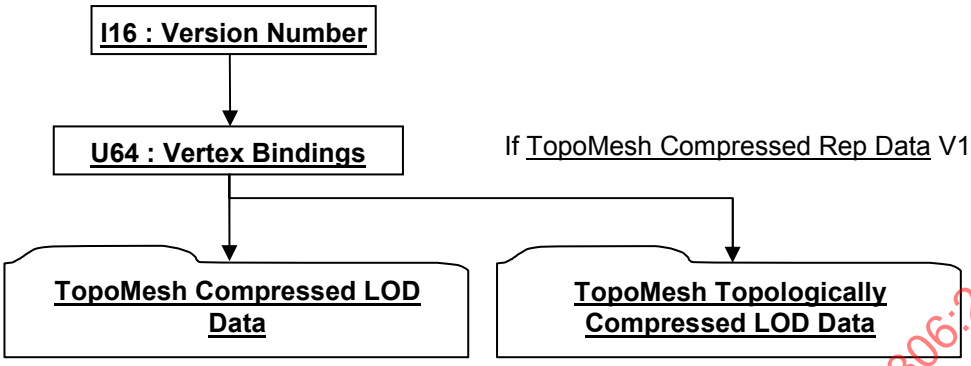


Figure 90 — Vertex Shape LOD Data collection

Complete description for TopoMesh Compressed LOD Data and TopoMesh Topologically Compressed LOD Data can be found in TopoMesh Compressed LOD Data and TopoMesh Topologically Compressed LOD Data.

I16 : Version Number

Version Number is the version identifier for this Vertex Shape LOD Data. For informaiton on local version numbers see best practice 14.5 Local version numbers

U64 : Vertex Bindings

Binding Attributes is a collection of normal, texture coordinate, and colour binding information encoded within a single U64 using the following bit allocation. All bits fields that are not defined as in use should be set to “0”..

Table 55 — Vertex Shape LOD Bindings values

Bits 1-3	Vertex Coordinate Binding. The Vertex Coordinate Binding denotes per vertex coordinate field data is present when one of the bits is set. Bit 1 - 2 Component Vertex Coordinates Bit 2 - 3 Component Vertex Coordinates Bit 3 - 4 Component Vertex Coordinates
Bit 4	Normal Binding. The Normal Binding denotes per vertex normal field data is present when the bit is set. Normal field data is always stored in 3 Component Normals when present.
Bits 5-6	Colour Binding. The Colour Binding denotes per vertex colour field data is present when one of the bits is set. Bit 5 - 3 Component Colours Bit 6 - 4 Component Colour
Bit 7	Vertex Flag Binding. The Vertex Flag Binding denotes the per vertex flag field is present on the shape when the bit is set.
Bits 9-12	Texture Coordinate 0 Binding. The Texture Coordinate 0 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 9 - 1 Component Texture Coordinates Bit 10 - 2 Component Texture Coordinates Bit 11 - 3 Component Texture Coordinates Bit 12 - 4 Component Texture Coordinates

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Bits 13-16	Texture Coordinate 1 Binding. The Texture Coordinate 1 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 13 - 1 Component Texture Coordinates Bit 14 - 2 Component Texture Coordinates Bit 15 - 3 Component Texture Coordinates Bit 16 - 4 Component Texture Coordinates
Bits 17-20	Texture Coordinate 2 Binding. The Texture Coordinate 2 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 17 - 1 Component Texture Coordinates Bit 18 - 2 Component Texture Coordinates Bit 19 - 3 Component Texture Coordinates Bit 20 - 4 Component Texture Coordinates
Bits 21-24	Texture Coordinate 3 Binding. The Texture Coordinate 3 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 21 - 1 Component Texture Coordinates Bit 22 - 2 Component Texture Coordinates Bit 23 - 3 Component Texture Coordinates Bit 24 - 4 Component Texture Coordinates
Bits 25-28	Texture Coordinate 4 Binding. The Texture Coordinate 4 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 25 - 1 Component Texture Coordinates Bit 26 - 2 Component Texture Coordinates Bit 27 - 3 Component Texture Coordinates Bit 28 - 4 Component Texture Coordinates
Bits 29-32	Texture Coordinate 5 Binding. The Texture Coordinate 5 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 29 - 1 Component Texture Coordinates Bit 30 - 2 Component Texture Coordinates Bit 31 - 3 Component Texture Coordinates Bit 32 - 4 Component Texture Coordinates
Bits 33-36	Texture Coordinate 6 Binding. The Texture Coordinate 6 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 33 - 1 Component Texture Coordinates Bit 34 - 2 Component Texture Coordinates Bit 35 - 3 Component Texture Coordinates Bit 36 - 4 Component Texture Coordinates
Bits 37-40	Texture Coordinate 7 Binding. The Texture Coordinate 7 binding denotes per vertex texture coordinates field data is present when one of the bits is set: Bit 37 - 1 Component Texture Coordinates Bit 38 - 2 Component Texture Coordinates Bit 39 - 3 Component Texture Coordinates Bit 40 - 4 Component Texture Coordinates
Bit 64	Auxiliary Vertex Field Binding. The Auxiliary Vertex Field Binding denotes per vertex auxiliary field data is present on the shape when the bit is set.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

6.1.2.2 TopoMesh LOD Data

TopoMesh LOD Data collection contains the common items to all TopoMesh LOD elements.

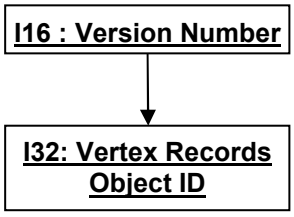


Figure 91 — TopoMesh LOD Data collection

I16 : Version Number

Version Number is the version identifier for this TopoMesh LOD Data. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32: Vertex Records Object ID

Vertex Records Object ID is the identifier for the vertex records associated with this Object. Other objects referencing these vertex records will do so using this Object ID.

6.1.2.2.1 TopoMesh Compressed LOD Data

TopoMesh Compressed LOD Data collection contains the common items to all TopoMesh Compressed LOD data elements.

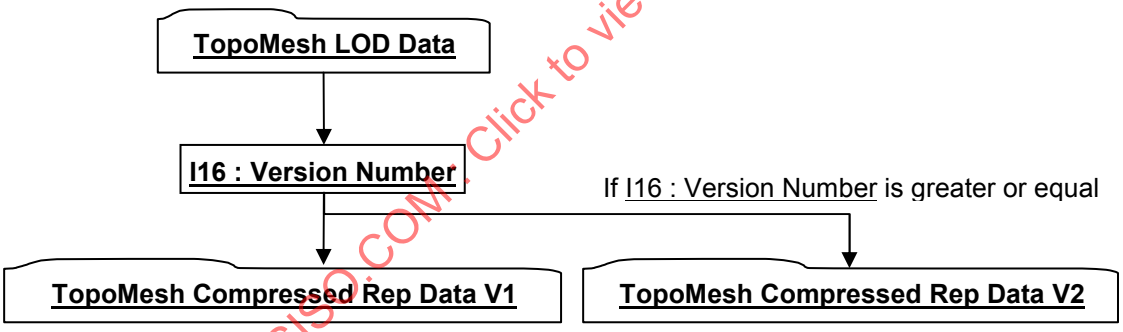


Figure 92 — TopoMesh LOD Data collection

Complete description for TopoMesh LOD Data, TopoMesh Compressed Rep Data V1, and TopoMesh Compressed Rep Data V2 can be found in TopoMesh LOD Data, TopoMesh Compressed Rep Data V1, and TopoMesh Compressed Rep Data V2.

I16 : Version Number

Version Number is the version identifier for this TopoMesh LOD Data. For informaiton on local version numbers see best practice 14.5 Local version numbers

6.1.2.2.2 TopoMesh Topologically Compressed LOD Data

TopoMesh Topologically Compressed LOD Data collection contains the common items to all TopoMesh Topologically Compressed LOD data elements.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

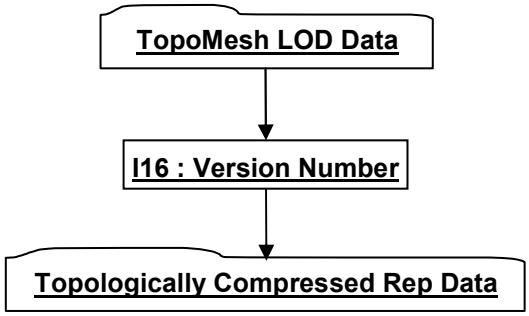


Figure 93 — TopoMesh Topologically Compressed LOD Data collection

Complete description for TopoMesh LOD Data and Topologically Compressed Rep Data can be found in TopoMesh LOD Data and Topologically Compressed Rep Data.

I16 : Version Number

Version Number is the version identifier for this TopoMesh Topologically Compressed LOD Data. For informaiton on local version numbers see best practice 14.5 Local version numbers

6.1.2.2.3 Topologically Compressed Rep Data

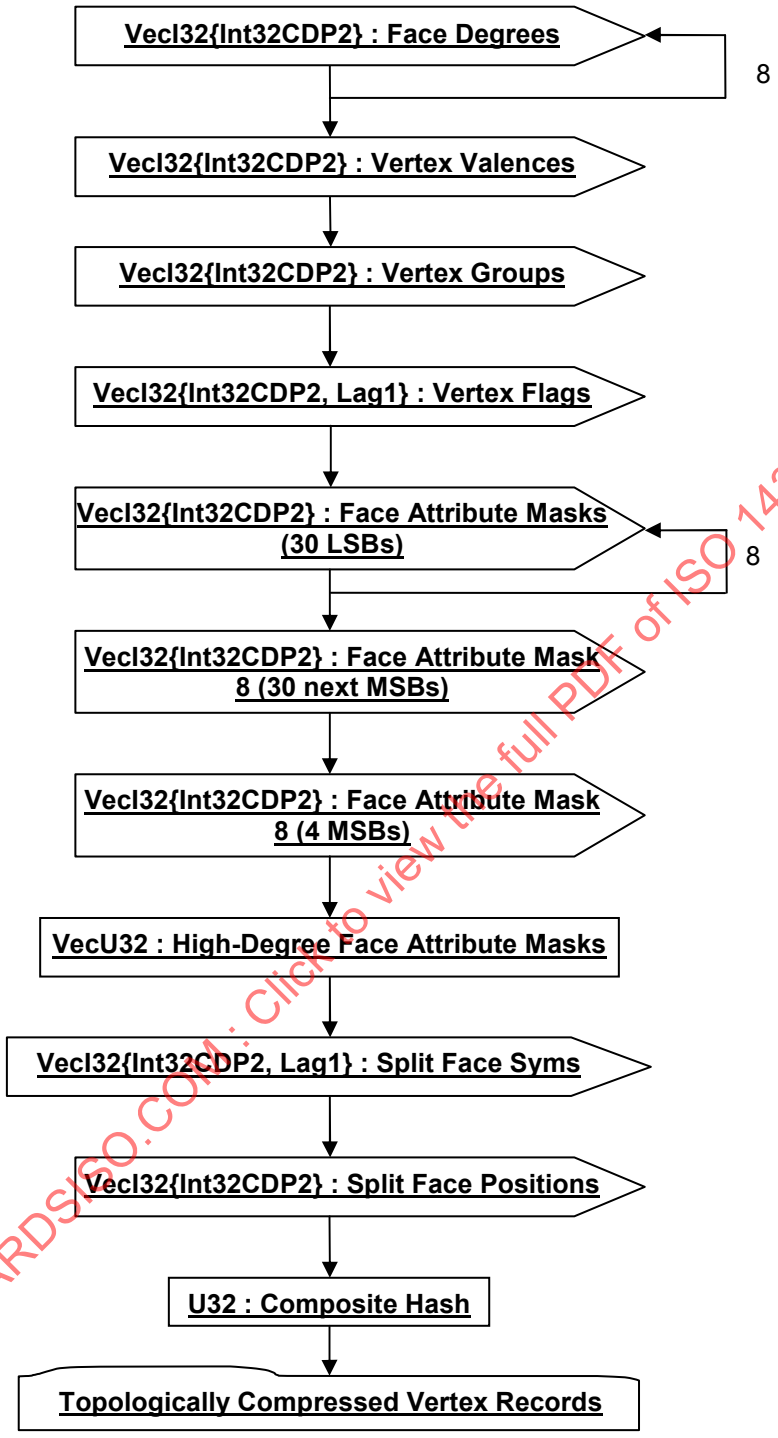
JT v9 represents triangle strip data very differently than it does in the JT v8 format. The new scheme stores the triangles from a TriStripSet as a topologically-connected triangle mesh. Even though *more* information is stored to the JT file, the additional structure provided by storing the full topological adjacency information actually provides a handsome reduction in the number of bytes needed to encode the triangles. More importantly, however, the topological information aids us in a more significant respect -- that of only storing the *unique* vertex records used by the TriStripSet. Combined, these two effects reduce the typical storage footprint of TriStripSet data by approximately half relative to the JT v8 format.

The tristrip information itself is no longer stored in the JT file -- only the triangles themselves. The reader is expected to re-tristrip (or not) as it sees fit, as tristrips may no longer provide a performance advantage during rendering. There may, however, remain some memory savings for tristripping, and so the decision to tristrip is left to the user.

To begin the decoding process, first read the compressed data fields shown in Figure 94. These fields provide all the information necessary to reconstruct the per face-group organized sets of triangles. The first 22 fields represent the topological information, and the remaining fields constitute the set of unique vertex records to be used. The next step is to run the topological decoder algorithm detailed in Annex D on this data to reconstruct the topologically connected representation of the triangle mesh in a so-called "dual VFMesh." The triangles in this heavy-weight data structure can then be exported to a lighter-weight form, and the dual VFMesh discarded if desired.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 94 — Topologically Compressed Rep Data Collection

Vecl32{Int32CDP2} : Face Degrees

Similarly to the way valences are encoded, the topology encoder emits the *degree* (number of incident vertices) of each face *in the order they were visited*. The number of face degrees in this array is equal to the number of faces in the mesh.

Vecl32{Int32CDP2} : Vertex Valences

As the coder visits each vertex in the mesh, it emits the *valence* (number of incident faces) of each vertex. These valences are collect *in the order they were visited* into this array. The number of valences in this array is equal to the number of (topological) vertices in the mesh.

© ISO 2012 – All rights reserved

VecI32{Int32CDP2} : Vertex Groups

This array is parallel to the Vertex Valences array above. As the coder emits the valence of each vertex, it also emits the face group number to which the dual vertex belongs into this array.

VecI32{Int32CDP2, Lag1} : Vertex Flags

This array is also parallel to the Vertex Valences array, and contains a value of 0 when the dual face was present in the original triangle mesh, and a value of 1 if the dual face is a *cover face* that was added to artificially close the original mesh.

VecI32{Int32CDP2} : Face Attribute Masks (30 LSBs)

This field is written 8 times – once for each of the 8 context groups listed above – and encodes the face attribute bit vector associated with a single face.

VecI32{Int32CDP2} : Face Attribute Mask 8 (30 next MSBs)

This field encodes the next 30 most significant bits of the 8th context group of face attribute bit vectors.

VecI32{Int32CDP2} : Face Attribute Mask 8 (4 MSBs)

This field encodes the 4 most significant bits of the 8th context group of face attribute bit vectors, rounding out its full 64-bit width.

VecU32 : High-Degree Face Attribute Masks

This field encodes all remaining face attribute bit vectors, adjoined end-to-end, and encoded as a single array of unsigned integers.

VecI32{Int32CDP2, Lag1} : Split Face Syms

Encodes the list of “split face” ID numbers in the order the coder encountered them.

VecI32{Int32CDP2} : Split Face Positions

Encodes the list of “split face” positions in the active vertex queue in the order the code encountered them.

U32 : Composite Hash

This field is a hash value computed on all of the above data using the hash function described in Annex C. It is written into the JT file so that a reader can perform the same hash on the above data and compare against this value in order to guarantee that it has read and decoded correct data from the JT file. It is *highly* encouraged that all readers perform this check, as even a single bit error in the topology information above can have catastrophic consequences on the topology decoder and the resulting mesh. Any writers are *required* to write this field using the method provided so that other readers may validate the data they read.

```
UInt32 uHash       = 0;
UInt32 anDegSyms[8] = {0},
       nValSyms = 0,
       nVGrpSyms = 0,
       nVtxFlags = 0,
       anAttrMasks[8] = {0},
       nLrgAttrMasks = 0,
       nSplitVtxSyms = 0,
       nSplitVtxPos = 0;
VecI32 vFaceDegreeSymbols[8], vviValenceSymbols, vFaceGroupSyms,
       vvuAttrMasks[8], viSplitVtxSyms, viSplitVtxPos;
VecI16 vFaceFlags;
VecU32 vuTmp, vuAttrMasksLrg;
...
for (i=0 ; i<8 ;i++)
  uHash = hash32((UInt32*) vFaceDegreeSymbols[i].ptr(), anDegSyms[i], uHash );
uHash = hash32((UInt32*) vviValenceSymbols.ptr(), nValSyms, uHash );
uHash = hash32((UInt32*)vVtxGroupSyms.ptr(), nVGrpSyms, uHash );
uHash = hash16((UInt16*)vVtxFlags.ptr(), nFlags, uHash );
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
for (i=0 ; i<7 ;i++)
  uHash = hash32((UInt32*)vvuAttrMasks[i].ptr(), anAttrMasks[i], uHash );
vuTmp = vvuAttrMasks[7] & 0x3fffffff;  // Lower 30 bits of each element
uHash = hash32(vuTmp.ptr(), anAttrMasks[7], uHash );
vuTmp = (vvuAttrMasks[7] >> 30) & 0x3fffffff; // Next 30 bits of each element
uHash = hash32(vuTmp.ptr(), anAttrMasks[7], uHash );
vuTmp = (vvuAttrMasks[7] >> 60) & 0x0f; // Upper 4 bits of each element
uHash = hash32(vuTmp.ptr(), anAttrMasks[7], uHash );
uHash = hash32(vuAttrMasksLrg.ptr(), nLrgAttrMasks, uHash );
uHash = hash32((UInt32*)viSplitVtxSyms.ptr(), nSplitVtxSyms, uHash );
uHash = hash32((UInt32*)viSplitVtxPos.ptr(), nSplitVtxPos, uHash );
```

6.1.2.3 Topologically Compressed Vertex Records

Documented here is the format of the vertex data written by the topological encoder from Annex E. Some additional explanation is necessary, however, because only the *unique* vertex coordinates are written to the JT file, while the remaining vertex attributes (normals, colours, texture coordinates, vertex flags) may not be unique.

Vertex coordinates are written to the file in the order that they were visited by the topology encoder. Note that this means that the number of vertex coordinates written is equal to the number of topological vertices in the mesh (i.e. all vertex coordinates are unique).

By contrast one set of vertex attribute records is written to the file corresponding to each 1 bit across all encoded dual Face Attribute Masks. The vertex attribute records are written in the order that the topology encoder visited them. The reader shall then use the topology decoder's output to correctly associate each vertex attribute record to the correct vertex coordinate using the dual Face Attribute Masks.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

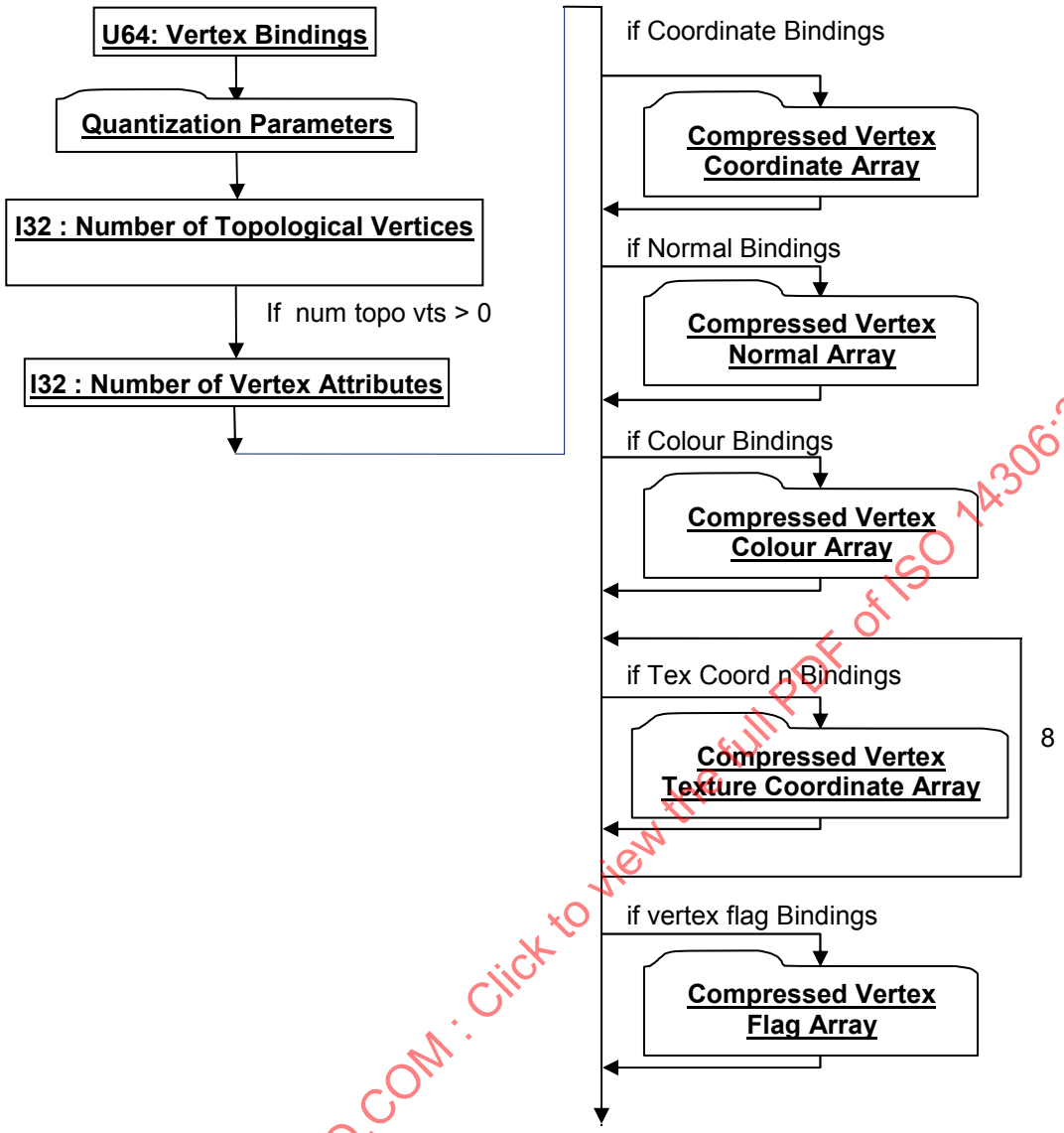


Figure 95 — Topologically Compressed Vertex Records data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

U64: Vertex Bindings

Vertex Bindings is a collection of normal, texture coordinate, and colour binding information encoded within a single U64. All bits fields that are not defined as in use should be set to "0".. For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

I32 : Number of Topological Vertices

This field is the number of topological vertices encoded by the topology encoder. This is the number of unique vertex coordinates that will be written in the later Compressed Vertex Coordinate Array field.

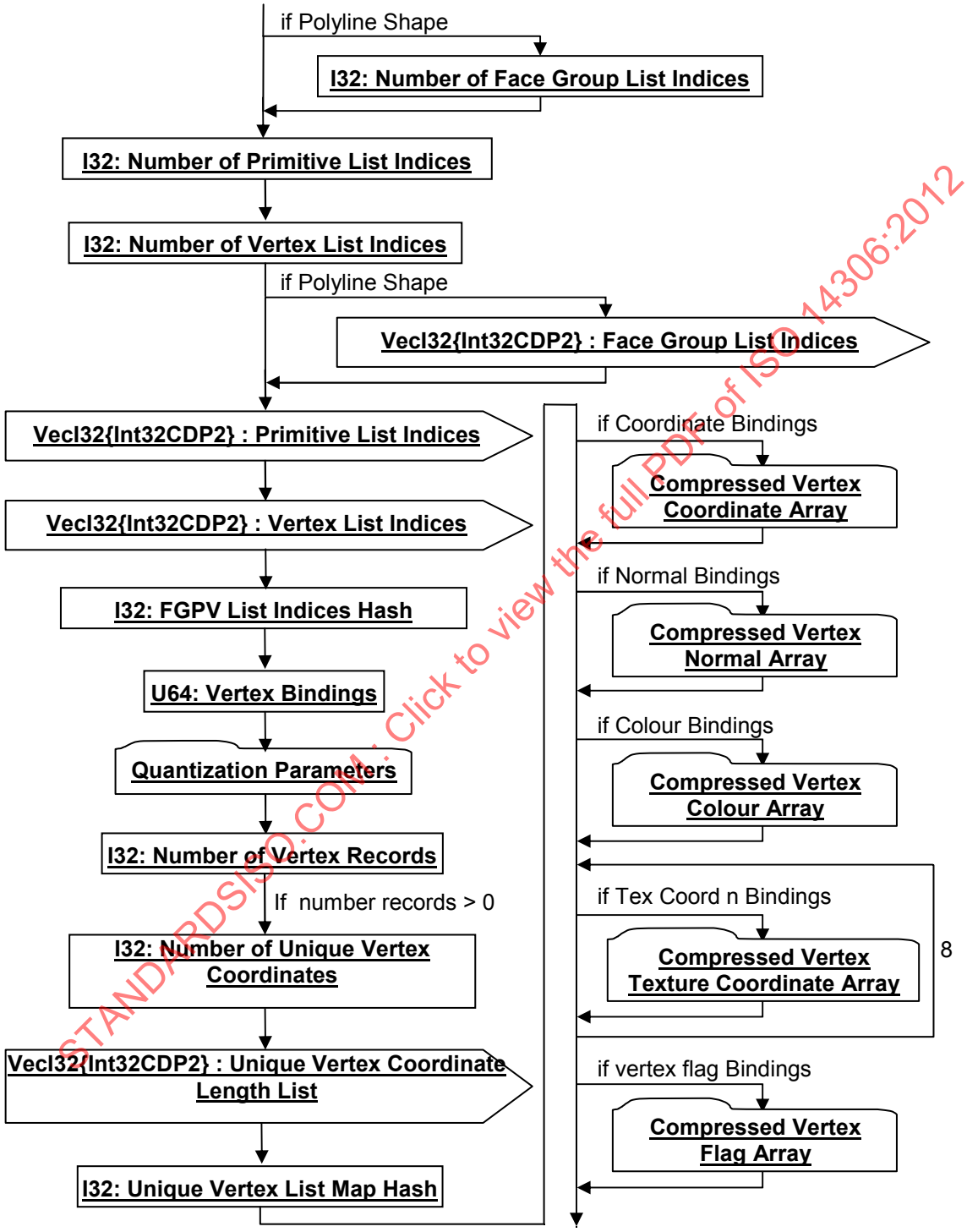
I32 : Number of Vertex Attributes

One set of vertex attribute records is written to the file corresponding to each 1 bit across all encoded dual Face Attribute Masks. The vertex attribute records are written in the order that the topology encoder visited them. The reader shall then use the topology decoder's output to correctly associate each vertex attribute record to the correct vertex coordinate using the dual Face Attribute Masks.

© ISO 2012 – All rights reserved

6.1.2.4 TopoMesh Compressed Rep Data V1

TopoMesh Compressed Rep Data V1 contains the geometric shape definition data (e.g. vertices, colours, normals, etc.) in a lossy or lossless compressed formed.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 96 — TopoMesh Compressed Rep Data V1 data collection

Complete description for Quantization Parameters can be found in Quantization Parameters.

© ISO 2012 – All rights reserved

I32: Number of Face Group List Indices

Number of Face Group List Indices.

I32: Number of Primitive List Indices

Number of Primitive List Indices.

I32: Number of Vertex List Indices

Number of Vertex List Indices.

VecI32{Int32CDP2} : Face Group List Indices

Face Group List Indices is a vector of indices into the uncompressed Raw Primitive Data marking the start/beginning of Faces. Face Group List Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP2} : Primitive List Indices

Primitive List Indices is a vector of indices into the uncompressed Raw Vertex Data marking the start/beginning of primitives. Primitive List Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP2} : Vertex List Indices

Vertex List Indices is a vector of indices (one per vertex) into the uncompressed/dequantized unique vertex data arrays (Vertex Coords, Vertex Normals, Vertex Texture Coords, Vertex Colours) identifying each Vertex's data (i.e. for each Vertex there is an index identifying the location within the unique arrays of the particular Vertex's data). The Compressed Vertex Index List uses the Int32 version of the CODEC to compress and encode data.

I32: FGPV List Indices Hash

The FGPV Hash is the combined hash value of the Face Group List Indices (if Polyline), Primitive List Indices, and Vertex List Indices. Refer to section 14.6 for a more detailed description on hashing.

```
UInt32 uHash    = 0;
UInt32 nFGIdx   = 0,
       nPrimIdx = 0,
       nVtxIdx  = 0;
vecI32 vFGIndices, vPrimIdices, vVertexIndices;
...
if (bLineStrip)
  uHash = hash32( (UInt32*)(&vFGIndices), nFGIdx+1, uHash );
uHash = hash32( (UInt32*)(& vPrimIdices), nPrimIdx+1, uHash );
uHash = hash32( (UInt32*)(& vVertexIndices), nVtxIdx  , uHash );
```

U64: Vertex Bindings

Vertex Bindings is a collection of normal, texture coordinate, and colour binding information encoded within a single U64. All bits fields that are not defined as in use should be set to "0". For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

I32: Number of Vertex Records

Number of vertex records.

I32: Number of Unique Vertex Coordinates

Number of unique vertex coordinates values in the Compressed Vertex Coordinate Array.

VecI32{Int32CDP2} : Unique Vertex Coordinate Length List

The Unique Vertex Length List contains the number of vertex records containing each of the unique vertex coordinates and should sum to the number of vertex records. When read in the Compressed Vertex

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Coordinate Array only contains a single value for each unique vertex coordinate value and is therefore parallel to the Unique Vertex Length List. In order to expand its coordinates into the vertex record space it unique coordinate value will need to be smeared out such that each unique vertex coordinate is repeated the number of times specified in the Unique Vertex Length List. The Compressed Vertex Normal, Colour, Texture, and Flag Arrays do not require the same expansion.

I32: Unique Vertex List Map Hash

The Unique Vertex List Map Hash is the hash value of Unique Vertex Coordinate Length List. Refer to section 14.6 for a more detailed description on hashing.

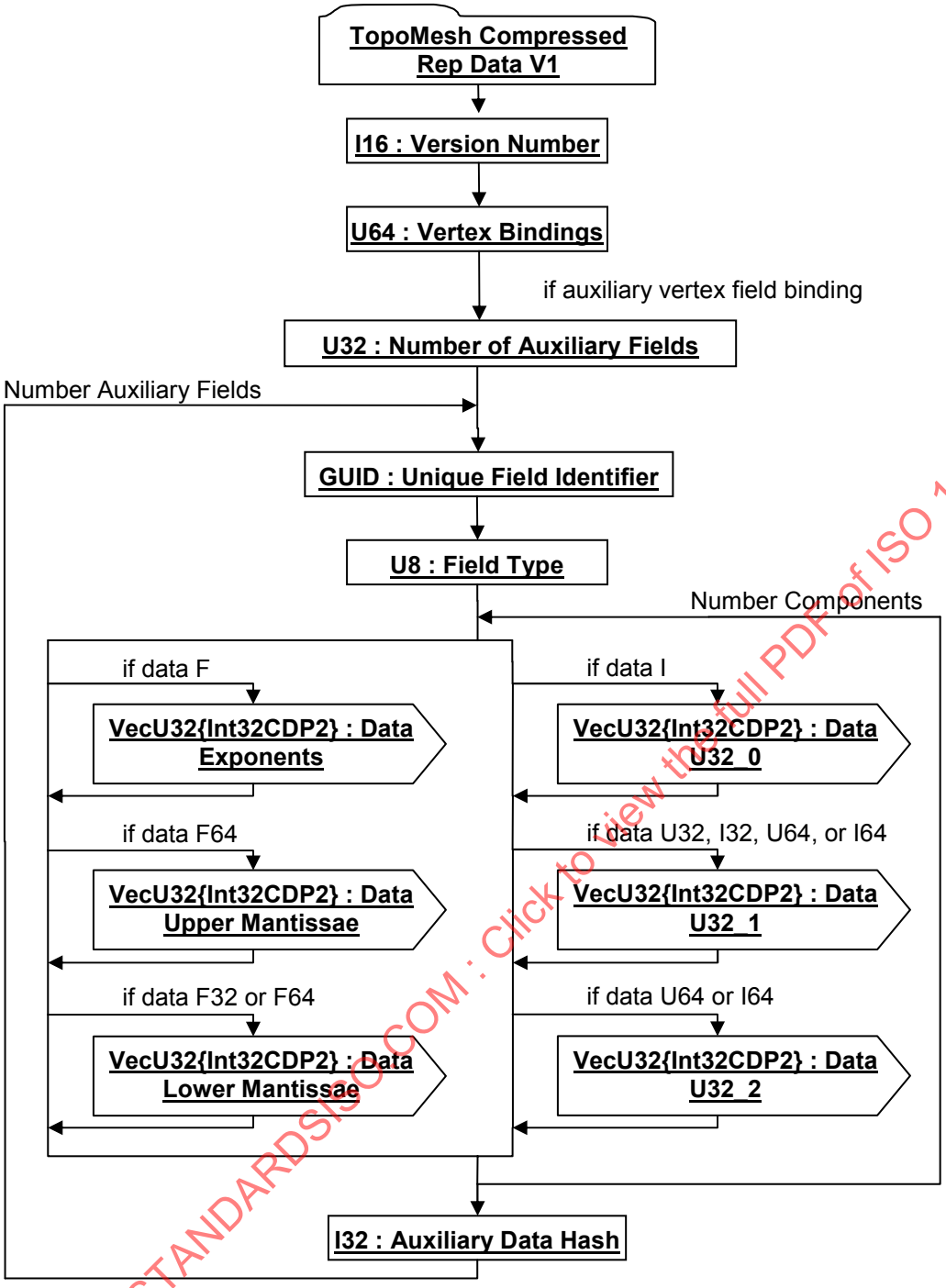
```
UInt32 uHash    = 0;
UInt32 nUniqVtx = 0;
vecF32 vUniqVtxIndices;
...
uHash = hash32( (UInt32*)(&vUniqVtxIndices), nUniqVtx, uHash );
```

6.1.2.5 TopoMesh Compressed Rep Data V2

TopoMesh Compressed Rep Data V2 data contains additional geometric shape data (auxiliary vertex fields) that were not included in V1. Each Auxiliary field contains data that is parallel to the existing vertex record fields in order capture additional information about each vertex (e.g. Vertex Identifiers, Weights, ...).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 97 — TopoMesh Compressed Rep Data V2 data collection

Complete description for TopoMesh Compressed Rep Data V1 can be found in TopoMesh Compressed Rep Data V1.

I16 : Version Number

Version Number is the version identifier for this TopoMesh Compressed Rep Data V2. For informaiton on local version numbers see best practice 14.5 Local version numbers

© ISO 2012 – All rights reserved

U64 : Vertex Bindings

Vertex Bindings is a collection of normal, texture coordinate, and colour binding information encoded within a single U64. All bits fields that are not defined as in use should be set to “0”. For more information see Vertex Shape LOD Data U64 : Vertex Bindings.

U32 : Number of Auxiliary Fields

Number of Auxiliary Vertex Fields

GUID : Unique Field Identifier

Each Auxiliary Vertex Field is associated with Unique Field Identifier to denote the usage of the contained data. This field is intended to be user defined. Any valid GUID, as defined in Table 4 - Composite Data Types, can be used as a Unique Field Identifier.

U8 : Field Type

Defines the number of components and type of data contained within the auxiliary field based upon the below table.

Table 56 — TopoMesh Compressed Rep Data V2 Field Type values

Type	Data	Components	Type	Data	Components
1	U8	1	24	I32	4
2	U8	2	25	U64	1
3	U8	3	26	U64	2
4	U8	4	27	U64	3
5	I8	1	28	U64	4
6	I8	2	29	I64	1
7	I8	3	30	I64	2
8	I8	4	31	I64	3
9	U16	1	32	I64	4
10	U16	2	33	F32	1
11	U16	3	34	F32	2
12	U16	4	35	F32	3
13	I16	1	36	F32	4
14	I16	2	37	F32	2x2
15	I16	3	38	F32	3x3
16	I16	4	39	F32	4x4
17	U32	1	40	F64	1
18	U32	2	41	F64	2
19	U32	3	42	F64	3
20	U32	4	43	F64	4
21	I32	1	44	F64	2x2
22	I32	2	45	F64	3x3
23	I32	3	46	F64	4x4

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

VecU32{Int32CDP2} : Data U32_0

Data U32_0 contains the low order bits from the data i'th data component for each vertex record in an U32 vector. For U8, I8, U16, and I16 data types this contains all bits. For U32, I32, U64, and I64 data types it contains bits 0 through 30. Data U32_0 uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Data U32_1

Data U32_1 contains the middle order bits from the data i'th data component for each vertex record in an U32 vector. For U32 and I32 data types it only contains bit 31. For U64 and U64 data types it contains bits 31 through 61. Data U32_1 uses the Int32 version of the CODEC to compress and encode data.

© ISO 2012 – All rights reserved

VecU32{Int32CDP2} : Data U32_2

Data U32_2 contains the upper order bits from the data i'th data component for each vertex record in an U32 vector. For U64 and I64 data types it contains bits 62 and 63. Data U32_2 uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Data Lower Mantissae

Vertex Coord Components is a vector of lower bits of Floating Point Mantissae for all the i'th component values of a set of vertex coordinates. For F32 data type this contains all bits of the mantissa, however for F64 data type it only contains bits 0 through 30. Data Lower Mantissae uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Data Upper Mantissae

Vertex Coord Components is a vector of upper bits of the Floating Point Mantissae for all the i'th component values of a set of vertex coordinates. For the F64 data type it contains bits 31 though 51. Data Upper Mantissae uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Data Exponents

Vertex Coord Components is a vector of Floating Point Exponents and Sign for all the i'th component values of a set of vertex coordinates. Data Exponents uses the Int32 version of the CODEC to compress and encode data.

I32 : Auxiliary Data Hash

The Auxiliary Data Hash is the combined hash of auxiliary field data arrays. Refer to section 14.6 for a more detailed description on hashing.

```
UInt32 uHash    = 0;
UInt32 nVtxRec  = 0,
       nComp    = 0;
vecU32 vU32_0, vU32_1, vU32_2, vLMANT, vUMANT, vEXP;
...
if ( bU8 || bI8 | bU16 | bI16 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vU32_0[i], nVtxRec, uHash );
  }
} else if ( bU32 || bI32 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vU32_0[i], nVtxRec, uHash );
   uHash = hash32( &vU32_1[i], nVtxRec, uHash );
  }
} else if ( bU64 || bI64 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vU32_0[i], nVtxRec, uHash );
   uHash = hash32( &vU32_1[i], nVtxRec, uHash );
   uHash = hash32( &vU32_2[i], nVtxRec, uHash );
  }
} else if ( bF32 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vLMANT[i], nVtxRec, uHash );
   uHash = hash32( &vEXP[i], nVtxRec, uHash );
  }
} else {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vLMANT[i], nVtxRec, uHash );
   uHash = hash32( &vUMANT[i], nVtxRec, uHash );
   uHash = hash32( &vEXP[i], nVtxRec, uHash );
  }
}
```

6.1.3 Tri-Strip Set Shape LOD Element

Object Type ID: 0x10dd10ab, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

A Tri-Strip Set Shape LOD Element contains the geometric shape definition data (e.g. vertices, polygons, normals, etc.) for a single LOD of a collection of independent and unconnected triangle strips. Each strip constitutes one primitive of the set and the ordering of the vertices in forming triangles, is the same as OpenGL's triangle strip definition [1].

A Tri-Strip Set Shape LOD Element is typically referenced by a Tri-Strip Set Shape Node Element using Late Loaded Property Atom Elements (see 5.1.1.10.3 Tri-Strip Set Shape Node Element and 5.2.7 Late Loaded Property Atom Element respectively).

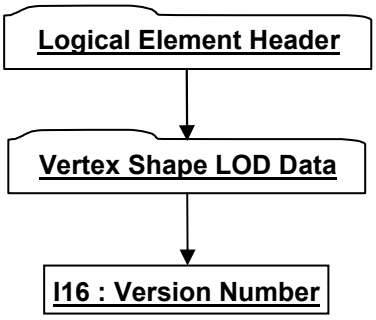


Figure 98 — Tri-Strip Set Shape LOD Element data collection

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

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

I16 : Version Number

Version Number is the version identifier for this Tri-Strip Set Shape LOD. For informaiton on local version numbers see best practice 14.5 Local version numbers

6.1.4 Polyline Set Shape LOD Element

Object Type ID: 0x10dd10a1, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A Polyline Set Shape LOD Element contains the geometric shape definition data (e.g. vertices, normals, etc.) for a single LOD of a collection of independent and unconnected polylines. Each polyline constitutes one primitive of the set.

A Polyline Set Shape LOD Element is typically referenced by a Polyline Set Shape Node Element using Late Loaded Property Atom Elements (see 5.1.1.10.5 Polyline Set Shape Node Element and 5.2.7 Late Loaded Property Atom Element respectively).

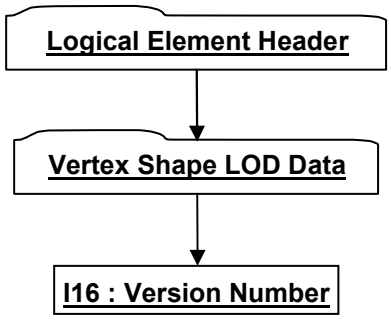


Figure 99 — Polyline Set Shape LOD Element data collection

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

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

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

I16 : Version Number

Version Number is the version identifier for this Polyline Set Shape LOD. For informaiton on local version numbers see best practice 14.5 Local version numbers

6.1.5 Point Set Shape LOD Element

Object Type ID: 0x98134716, 0x0011, 0x0818, 0x19, 0x98, 0x08, 0x00, 0x09, 0x83, 0x5d, 0x5a

A Point Set Shape LOD Element contains the geometric shape definition data (e.g. coordinates, normals, etc.) for a collection of independent and unconnected points. Each point constitutes one primitive of the set.

A Point Set Shape LOD Element is typically referenced by a Point Set Shape Node Element using Late Loaded Property Atom Elements (see 5.1.1.10.5 Point Set Shape Node Element and 5.2.7 Late Loaded Property Atom Element respectively).

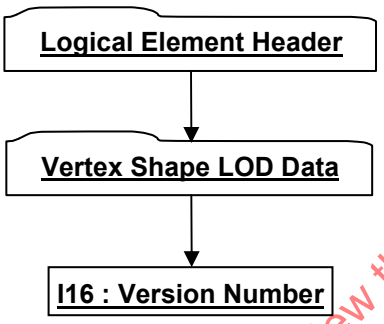


Figure 100 — Point Set Shape LOD Element data collection

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

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

I16 : Version Number

Version Number is the version identifier for this Point Set Shape LOD. For informaiton on local version numbers see best practice 14.5 Local version numbers

6.1.6 Null Shape LOD Element

Object Type ID: 0x3e637aed, 0x2a89, 0x41f8, 0xa9, 0xfd, 0x55, 0x37, 0x37, 0x3, 0x96, 0x82

A Null Shape LOD Element represents the pseudo geometric shape definition data for a NULL Shape Node Element. Although a NULL Shape Node Element has no real geometric primitive representation (i.e. is empty), its usage as a "proxy/placeholder" node within the LSG still supports the concept of having a defined bounding box and thus the existence of this Null Shape LOD Element type.

A Null Shape LOD Element is typically referenced by a NULL Shape Node Element using Late Loaded Property Atom Elements (see 5.1.1.10.7 NULL Shape Node Element and 5.2.7 Late Loaded Property Atom Element respectively).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

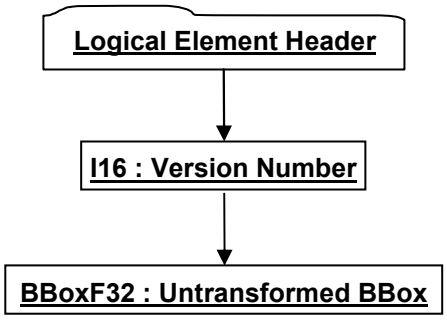


Figure 101 — Null Shape LOD Element data collection

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

I16 : Version Number

Version Number is the version identifier for this Null Shape LOD Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

BBoxF32 : Untransformed BBox

The Untransformed BBox is an axis-aligned LCS bounding box and represents the untransformed extents for this Null Shape LOD Element.

6.2 Primitive Set Shape Element

Object Type ID: 0xe40373c2, 0x1ad9, 0x11d3, 0x9d, 0xaf, 0x0, 0xa0, 0xc9, 0xc7, 0xdd, 0xc2

A Primitive Set Shape Element defines the minimum data necessary to procedurally generate LODs for a list of primitive shapes (e.g. box, cylinder, sphere, etc.). "Procedurally generate" means that the raw geometric shape definition data (e.g. vertices, polygons, normals, etc) for LODs is not directly stored; instead some basic shape information is stored (e.g. sphere centre and radius) from which LODs can be generated.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

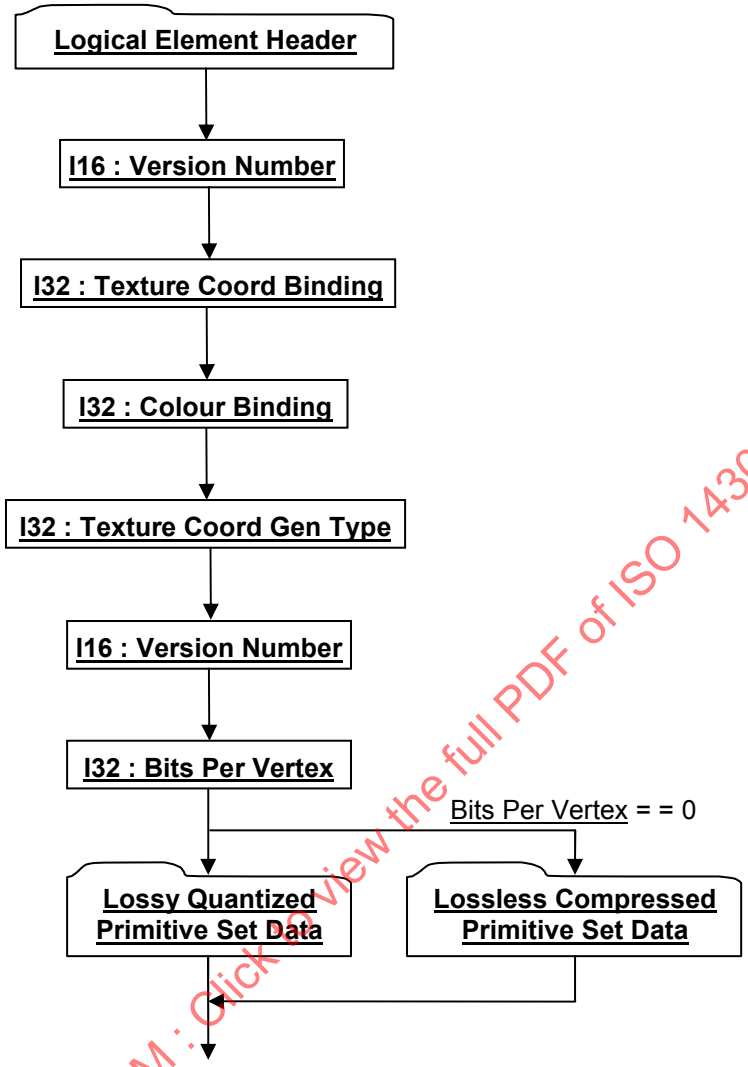


Figure 102 — Primitive Set Shape Element data collection

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

I16 : Version Number

Version Number is the version identifier for this element. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : Texture Coord Binding

Texture Coord Binding specifies how (at what granularity) texture coordinate data is supplied ("bound") for the shape. Valid values are as follows:

Table 57 — Primitive Set Shape Texture Coord Binding values

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

I32 : Colour Binding

Colour Binding specifies how (at what granularity) colour data is supplied ("bound") for the shape. Valid values are the same as documented for Texture Coord Binding data field.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Version Number

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

Table 58 — Primitive Set Shape Version Number values

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

I32 : Bits Per Vertex

Bits Per Vertex specifies the number of quantization bits per vertex coordinate component. Value shall be within range [0:32] inclusive.

I32 : Texture Coord Gen Type

Texture Coord Gen Type specifies how a texture is applied to each face of the primitive. Single tile means one copy of the texture will be stretched to fit the face, isotropic means that the texture will be duplicated on the longer dimension of the face in order to maintain the texture's aspect ratio

Table 59 — Primitive Set Shape Texture Coord Gen Type values

= 0	Single Tile…Indicates that a single copy of a texture image will be applied to significant primitive features (i.e. cube face, cylinder wall, end cap) no matter how eccentrically shaped.
= 1	Isotropic…Implies that multiple copies of a texture image may be mapped onto eccentric surfaces such that a mapped texel stays approximately square.

6.2.1 Lossless Compressed Primitive Set Data

The Lossless Compressed Primitive Set Data collection contains all the per-primitive information stored in a "lossless" compression format for all primitives in the Primitive Set. The Lossless Compressed Primitive Set Data collection is only present when the Bits Per Vertex data field equals "0" (see 6.2 Primitive Set Shape Element for complete description).

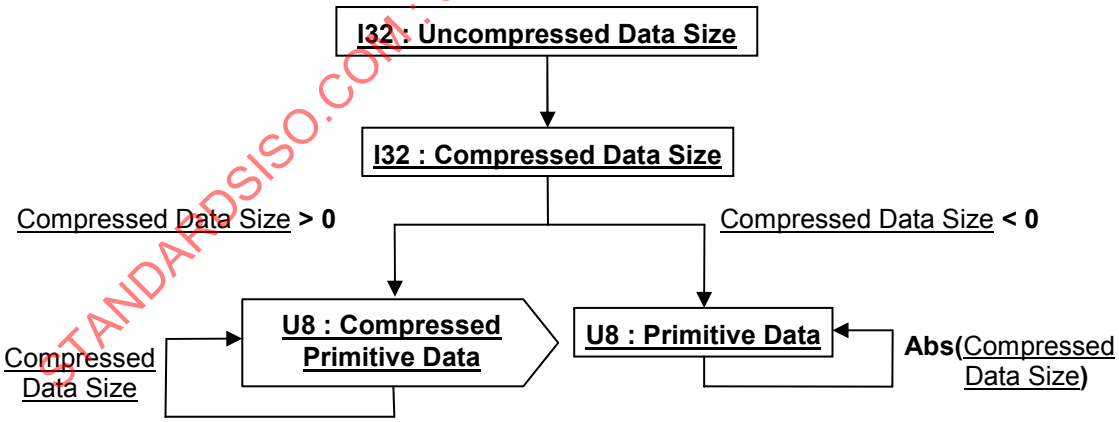


Figure 103 — Lossless Compressed Primitive Set Data collection

I32 : Uncompressed Data Size

Uncompressed Data size specifies the uncompressed size of Primitive Data or Compressed Primitive Data in bytes.

I32 : Compressed Data Size

Compressed Data Size specifies the compressed size of Primitive Data or Compressed Primitive Data in bytes. If the Compressed Data Size is negative, then the Compressed Primitive Data field is not present (i.e.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

data is not compressed) and the absolute value of Compressed Data Size should be equal to Uncompressed Data Size value.

U8 : Primitive Data

The Primitive Data field is a packed array of the raw per primitive data (i.e. reserved, params1, params2, params3, colour, type) sequentially for all primitives in the set. The Primitive Data field is only present if Compressed Data Size value is less than zero.

The per primitive data is packed into Primitive Data array using an interleaved data schema/format as follows:

{[reserved], [params1], [params2], [params3], [colour], [type]}, …, **for all primitives**

Where the data elements have the following size and meaning:

Table 60 — Lossless Compressed Primitive Set Data Field values

Element	Data Type	Description
reserved	I32	This is a field reserved for future expansion of the JT Format.
params1	CoordF32	Interpretation is Primitive Type specific (see below table)
params2	DirF32	Interpretation is Primitive Type specific (see below table)
params3	Quaternion	Interpretation is Primitive Type specific (see below table)
colour	RGB	Red, Green, Blue colour component values
type	I32	Primitive Type = 0 – Box = 1 – Cylinder = 2 – Pyramid = 3 – Sphere = 4 – Tri-Prism

Given this format of the Primitive Data, and the previously read size fields, a reader can then implicitly compute the data stride (length of one primitive entry in Primitive Data), and number of primitives.

The interpretation of the three “params#” data fields is primitive type dependent as follows:

Table 61 — Primitive Set “params#” Data Fields Interpretation

Primitive Type	params1			params2			params3			
	[0]	[1]	[2]	[0]	[1]	[2]	[0]	[1]	[2]	[3]
Box	min X	min Y	min Z	length X	length Y	length Z	orientation in Quaternion form			
Cylinder	base centre X	base centre Y	base centre Z	spine X	spine Y	spine Z	radius 1	radius 2	N/A	N/A
Pyramid	base centre X	base centre Y	base centre Z	length X	length Y	length Z	orientation in Quaternion form			
Sphere	centre X	centre Y	centre Z	radius	N/A	N/A	N/A	N/A	N/A	N/A
Tri-Prism	bottom front X	bottom front Y	bottom front Z	length X (to right)	length Y (to back)	length Z (to top)	orientation in Quaternion form			

U8 : Compressed Primitive Data

The Compressed Primitive Data field represents the same data as documented in Primitive Data field above except that the data is compressed using the general “ZLIB deflation compression” method. The Compressed

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

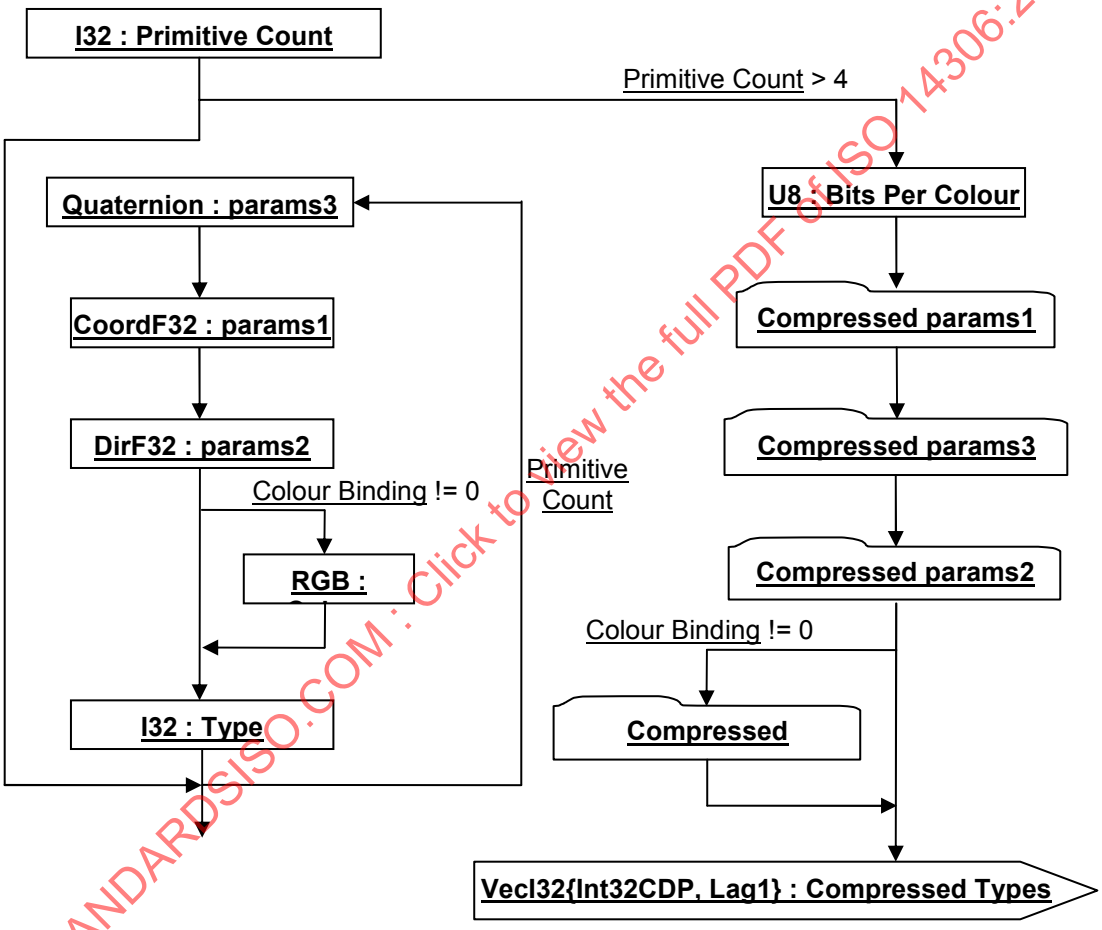
© ISO 2012 – All rights reserved

Primitive Data field is only present if Compressed Data Size value is greater than zero. See 13 Data Compression and Encoding for more details on ZLIB compression and ZLIB library version used.

6.2.2 Lossy Quantized Primitive Set Data

The Lossy Quantized Primitive Set Data collection contains all the per-primitive information (i.e. reserved, params1, params2, params3, colour, type) stored in a "lossy" encoding/compression format for all primitives in the Primitive Set. The Lossy Quantized Primitive Set Data collection is only present when the Bits Per Vertex data field is NOT equal to "0" (See 6.2 Primitive Set Shape Element for compete description).

The interpretation of the three per-primitive "params#" data fields is primitive type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the "params#" data fields.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 104 — Lossy Quantized Primitive Set Data collection

I32 : Primitive Count

Primitive Count specifies the number of primitives in the Primitive Set.

Quaternion : params3

Interpretation of params3 data field is primitive Type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the params3 data fields.

© ISO 2012 – All rights reserved

CoordF32 : params1

Interpretation of params1 data field is primitive Type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the params1 data fields.

DirF32 : params2

Interpretation of params1 data field is primitive Type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the params1 data fields.

RGB : Colour

Colour specifies the Red, Green Blue colour components for the primitive. This data field is only present if previously read Colour Binding (see 6.2 Primitive Set Shape Element) is not equal to "0".

I32 : Type

Type specifies the primitive type. See Table 60 — Lossless Compressed Primitive Set Data Field values in Lossless Compressed Primitive Set Data for valid primitive Type values.

U8 : Bits Per Colour

Bits Per Colour specifies the number of quantization bits per colour component. Value shall be within range [0:32] inclusive.

Vecl32{Int32CDP, Lag1} : Compressed Types

The Compressed Types data field is a vector of Type data for all the primitives in the Primitive Set. Compressed Types uses the Int32 version of the CODEC to compress and encode data. In an uncompressed form the valid primitive Type vales are as documented in Table 60 — Lossless Compressed Primitive Set Data Field values in Lossless Compressed Primitive Set Data.

6.2.2.1 Compressed params1

Compressed params1 is the compressed representation of the *params1* data for all the primitives in the Primitive Set. Note that the interpretation of the uncompressed *params1* data is primitive Type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the *params1* data fields

The *params1* data for all primitives in the Primitive Set is compressed/encoded on a per ordinate basis using a separate Uniform Quantizer (with Bits Per Vertex number of quantization bits) for each collection of ordinate values. Since *params1* is of type "CoordF32", it has three ordinate values (three F32 values), and thus three Uniform Quantizers (where a Uniform Quantizer is a scalar quantizer/encoder whose range is divided into levels of equal spacing). See 13 Data Compression and Encoding for more complete description of Uniform Quantizer.

The JT Format packs all the *params1* data for all primitives into a single array using an ordinate dependent order (as shown below) and then encodes each of the lists of ordinate values using a separate Uniform Quantizer per ordinate list.

```
{prim1 params1[0], prim2 params1[0],…primN params1[0],
 prim1 params1[1], prim2 params1[1],…primN params1[1],
 prim1 params1[2], prim2 params1[2],…primN params1[2]}
```

The result of the Uniform Quantizer encoding is a range min and max floating point value pairs for each ordinate value collection, and an integer array of *params1* quantization codes that corresponds to the above described "ordinate dependent order" packed array of *params1* data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

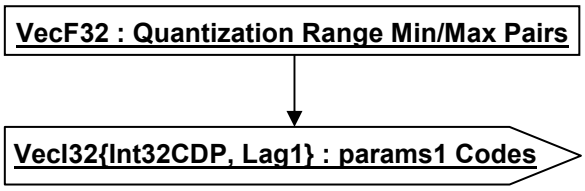


Figure 105 — Compressed params1 data collection

VecF32 : Quantization Range Min/Max Pairs

Quantization Range Min/Max Pairs is a vector of Uniform Quantizer range min/max value pairs. There shall be a min/max pair for each ordinate value collection (i.e. each Uniform Quantizer). Thus the length of this vector is "2 * num_ordinates" (so vector length would be "6" for *params1* data).

VecI32{Int32CDP, Lag1} : params1 Codes

The params1 Codes data field is a vector of quantizer "codes" for the *params1* data of all the primitives in the Primitive Set. The params1Codes also uses the Int32 version of the CODEC to compress and encode data.

6.2.2.2 Compressed params3

Compressed params3 is the compressed representation of the *params3* data for all the primitives in the Primitive Set. Note that the interpretation of the uncompressed *param31* data is primitive Type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the *params3* data fields

The *params3* data for all primitives in the Primitive Set is compressed/encoded on a per ordinate basis using a separate Uniform Quantizer (with Bits Per Vertex number of quantization bits) for each collection of ordinate values. Since *params1* is of type "Quaternion", it has four ordinate values (four F32 values), and thus four Uniform Quantizers (where a Uniform Quantizer is a scalar quantizer/encoder whose range is divided into levels of equal spacing). See 13 Data Compression and Encoding for more complete description of Uniform Quantizer.

The JT Format packs all the *params3* data for all primitives into a single array using an ordinate dependent order (as shown below) and then encodes each of the lists of ordinate values using a separate Uniform Quantizer per ordinate list.

```
{prim1 params3[0], prim2 params3[0],…primN params3[0],
 prim1 params3[1], prim2 params3[1],…primN params3[1],
 prim1 params3[2], prim2 params3[2],…primN params3[2],
 prim1 params3[3], prim2 params3[3],…primN params3[3]}
```

The result of the Uniform Quantizer encoding is a range min and max floating point value pairs for each ordinate value collection, and an integer array of *params3* quantization codes that corresponds to the above described "ordinate dependent order" packed array of *params3* data.

The storage format of Compressed params3 is exactly the same as that documented in Figure 105 — Compressed params1 data collection.

6.2.2.3 Compressed params2

Compressed params2 is the compressed representation of the *params2* data for all the primitives in the Primitive Set. Note that the interpretation of the uncompressed *params2* data is primitive Type dependent. See Table 61 — Primitive Set "params#" Data Fields Interpretation in Lossless Compressed Primitive Set Data for per-primitive type description of the *params2* data fields

The *params2* data for all primitives in the Primitive Set is compressed/encoded on a per ordinate basis using a separate Uniform Quantizer (with Bits Per Vertex number of quantization bits) for each collection of ordinate

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

values. Since *params2* is of type “DirF32”, it has three ordinate values (three F32 values), and thus three Uniform Quantizers (where a Uniform Quantizer is a scalar quantizer/encoder whose range is divided into levels of equal spacing). See 13 Data Compression and Encoding for more complete description of Uniform Quantizer.

The JT Format packs all the *params2* data for all primitives into a single array using an ordinate dependent order (as shown below) and then encodes each of the lists of ordinate values using a separate Uniform Quantizer per ordinate list.

```
{prim1 params2[0], prim2 params2[0],…primN params2[0],
 prim1 params2[1], prim2 params2[1],…primN params2[1],
 prim1 params2[2], prim2 params2[2],…primN params2[2]}
```

The result of the Uniform Quantizer encoding is a range min and max floating point value pairs for each ordinate value collection, and an integer array of *params2* quantization codes that corresponds to the above described “ordinate dependent order” packed array of *params2* data.

The storage format of Compressed params2 is exactly the same as that documented in Figure 105 — Compressed params1 data collection.

6.2.2.4 Compressed Colours

Compressed Colours is the compressed representation of the *colour* data for all the primitives in the Primitive Set. This data collection is only present if previously read Colour Binding (see 6.2 Primitive Set Shape Element) is not equal to “0”.

The *colour* data for all primitives in the Primitive Set is compressed/encoded on a per ordinate basis using a separate Uniform Quantizer (with Bits Per Colour number of quantization bits) for each collection of ordinate values. Since *colour* is of type “RGB”, it has three ordinate values (three F32 values), and thus three Uniform Quantizers (where a Uniform Quantizer is a scalar quantizer/encoder whose range is divided into levels of equal spacing). See 13 Data Compression and Encoding for more complete description of Uniform Quantizer.

The JT Format packs all the *colour* data for all primitives into a single array using an ordinate dependent order (as shown below) and then encodes each of the lists of ordinate values using a separate Uniform Quantizer per ordinate list.

```
{prim1 colour[0], prim2 colour[0],…primN colour[0],
 prim1 colour[1], prim2 colour[1],…primN colour[1],
 prim1 colour[2], prim2 colour[2],…primN colour[2]}
```

The result of the Uniform Quantizer encoding is a range min and max floating point value pairs for each ordinate value collection, and an integer array of *colour* quantization codes that corresponds to the above described “ordinate dependent order” packed array of *colour* data.

The storage format of Compressed Colours is exactly the same as that documented in Figure 105 — Compressed params1 data collection.

7 JT B-Rep Segment

JT B-Rep Segment contains an Element that defines the precise geometric Boundary Representation data for a particular Part in JT B-Rep format. Note that there is also another Boundary Representation format (i.e. XT B-Rep) supported by the JT file format within a different file Segment Type. Complete description for the XT B-Rep can be found in 8 XT B-Rep Segment.

JT B-Rep Segments are typically referenced by Part Node Elements (see 5.1.1.5 Part Node Element) using Late Loaded Property Atom Elements (see 5.2.7 Late Loaded Property Atom Element). The JT B-Rep Segment type supports ZLIB compression on all element data, so all elements in JT B-Rep Segment use the Logical Element Header ZLIB form of element header data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

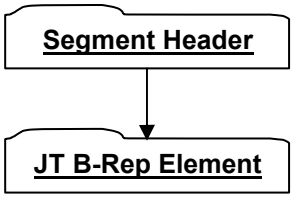


Figure 106 — JT B-Rep Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

7.1 JT B-Rep Element

Object Type ID: 0x873a70c0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

JT B-Rep Element represents a particular Part's precise data in JT boundary representation format. Much of the "heavyweight" data contained within a JT B-Rep Element is compressed and/or encoded. The compression and/or encoding state is indicated through other data stored in each JT B-Rep Element.

Two important aspects of a Part are its geometry and its topology. The geometry describes the shape of a Part: this Surface is a plane, that Surface is a cylinder, this Curve is an arc, etc. The topology describes the connectivity of the Part: this Point is inside the Part, these Surfaces are next to each other, etc. The 0, 1, and 2 dimensional building blocks of geometry are Points, Curves, and Surfaces. The corresponding topological building blocks are Vertices, Edges, and Faces. Topology also uses Shells and Regions to conceptually divide up the three dimensional space.

Parts may have the same topology, but wildly different geometry. Imagine the Surfaces of a Part being composed of rubber. The topology of the Part does not change as we deform the Part by bending or stretching the surfaces, as long as we do not cut or glue them (we call this a "nice" deformation). A Part's topology can be classified as being "manifold" or "non-manifold"; where "manifold" implies that the Part has the property that each Edge, excluding seams and poles, has exactly two faces using it.

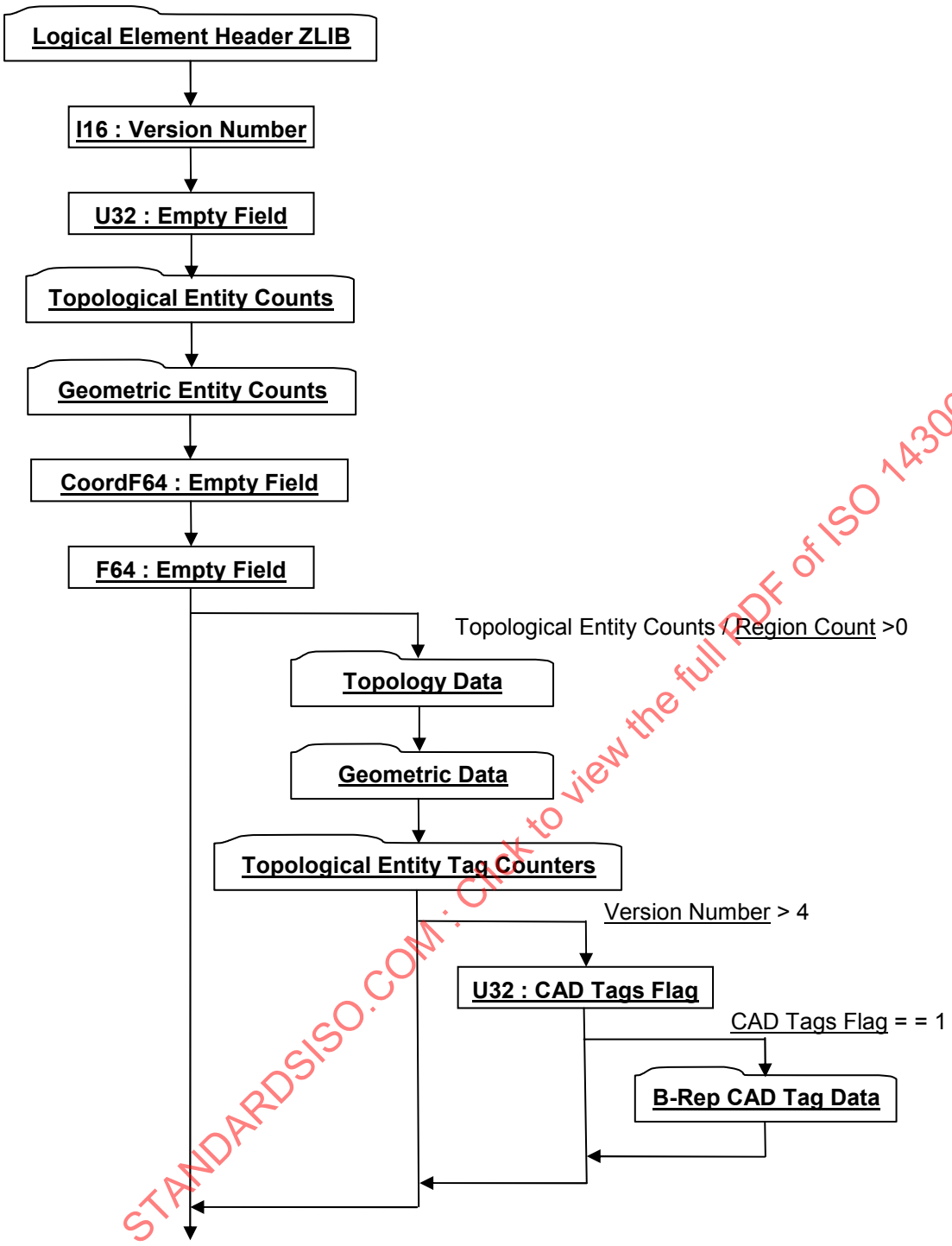
Similarly, Parts may have nearly identical geometry but different topology. The topology of a Part depends on how the geometry is put together. A Part may be manifold or non-manifold simply depending on how the geometry is put together. In addition to describing connectivity in space, topology is used to describe areas of interest (active areas) on Surfaces. These active Surface areas are used in defining a complex Part. The areas are specified by oriented Loops and often referred to as trimmed Surfaces which are exactly the 2-dimensional topological building block called a Face.

Readers desiring/needing a more in-depth exploration of boundary representation theory in order to understand the significance/meaning of some of the JT B-Rep data fields are referred to references [5] and [6] listed in the bibliography section of this International Standard.

Since the topology is a convenient way to describe or organize the Part, it is also convenient to store the geometry of the Part in the topological structures. The following sub-sections document the JT B-Rep format for storing the topology and geometry of a Part in a JT file.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 107 — JT B-Rep Element data collection

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

I16 : Version Number

Version Number is the version identifier for this JT B-Rep Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

U32 : Empty Field

Refer to best practice 14.4 Empty Field

© ISO 2012 – All rights reserved

CoordF64 : Empty Field

Refer to best practice 14.4 Empty Field

F64 : Empty Field

Refer to best practice 14.4 Empty Field

U32 : CAD Tags Flag

CAD Tags Flag is a flag indicating whether CAD Tag data exist for the JT B-Rep.

7.1.1 Topological Entity Counts

Topological Entity Counts data collection defines the counts for each of the various topological entities within a B-Rep.

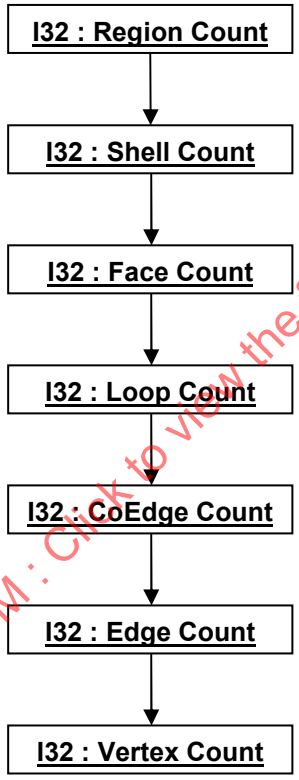


Figure 108 — Topological Entity Counts data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

I32 : Region Count

Region Count indicates the number of topological region entities in the B-Rep.

I32 : Shell Count

Shell Count indicates the number of topological shell entities in the B-Rep

I32 : Face Count

Face Count indicates the number of topological face entities in the B-Rep

I32 : Loop Count

Loop Count indicates the number of topological loop entities in the B-Rep

© ISO 2012 – All rights reserved

I32 : CoEdge Count

CoEdge Count indicates the number of topological coedge entities in the B-Rep

I32 : Edge Count

Edge Count indicates the number of topological edge entities in the B-Rep

I32 : Vertex Count

Vertex Count indicates the number of topological vertex entities in the B-Rep

7.1.2 Geometric Entity Counts

Geometric Entity Counts data collection defines the counts for each of the various geometric entities within a B-Rep.

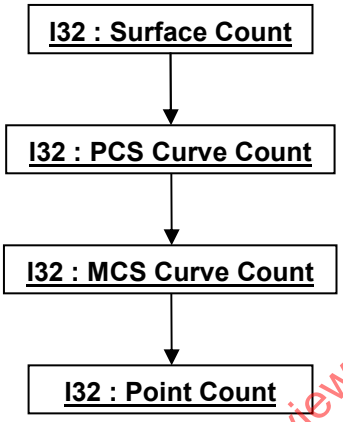


Figure 109 — Geometric Entity Counts data collection

I32 : Surface Count

Surface Count indicates the number of distinct geometric surface entities in the B-Rep

I32 : PCS Curve Count

PCS Curve Count indicates the number of distinct geometric Parameter Coordinate Space curves (i.e. UV curve) entities in the B-Rep

I32 : MCS Curve Count

MCS Curve Count indicates the number of distinct geometric (Model Coordinate Space) curves (i.e. XYZ curve) entities in the B-Rep.

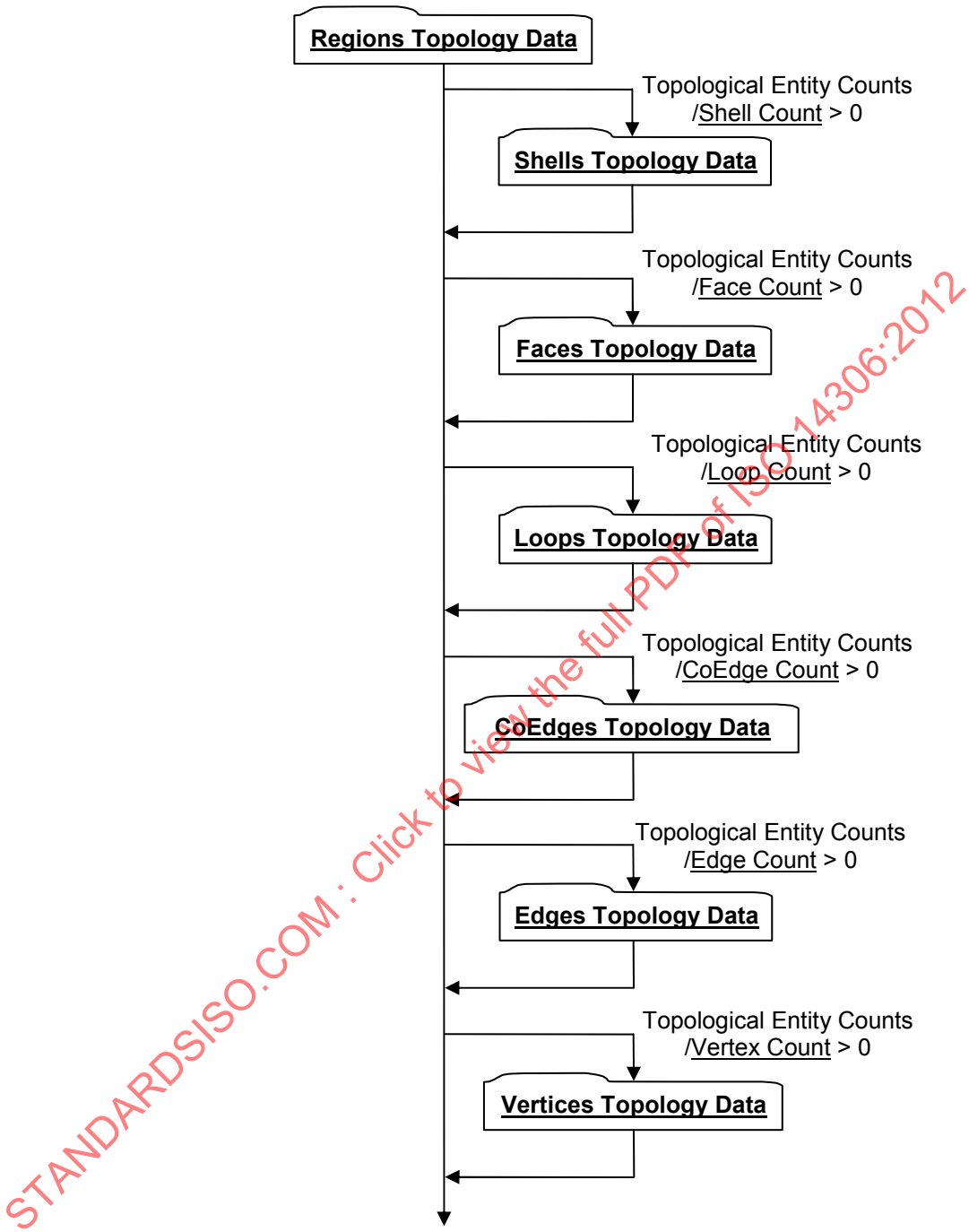
I32 : Point Count

Point Count indicates the number of distinct geometric point entities in the B-Rep.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

7.1.3 Topology Data



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 110 — Topology Data collection

7.1.3.1 Regions Topology Data

Regions Topology Data defines the set of non-overlapping Shells comprising each Region. The volume of a Region is that volume lying inside each “anti-hole Shell” and outside each simply-contained “hole Shell” belonging to the particular Region. A Region is analogous to a dimensionally elevated face where Region corresponds to Face and Shell corresponds to Trim Loop.

Each Region’s defining Shells are identified in a list of Shells by an index for both the first Shell and the last Shell in each Region (i.e. all Shells inclusive between the specified first and last Shell list index define the particular Region).

© ISO 2012 – All rights reserved

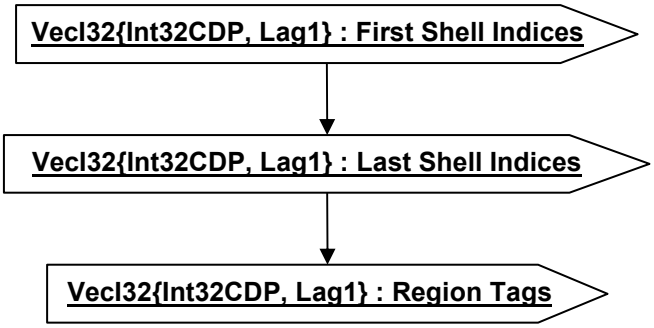


Figure 111 — Regions Topology Data collection

Vecl32{Int32CDP, Lag1} : First Shell Indices

First Shell Indices is a vector of indices representing the index of the first Shell in each Region. First Shell Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : Last Shell Indices

Last Shell Indices is a vector of indices representing the index of the last Shell in each Region. Last Shell Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : Region Tags

Each Region has an identifier tag. Region Tags is a vector of identifier tags for a set of Regions. Region Tags uses the Int32 version of the CODEC to compress and encode data.

7.1.3.2 Shells Topology Data

Shells Topology Data defines the set of topological adjacent Faces making up each Shell. A Shell's set of topological adjacent Faces define a single (usually closed) two manifold solid that in turn defines the boundary between the finite volume of space enclosed within the Shell and the infinite volume of space outside the Shell. Additional, each Shell has a flag that denotes whether the Shell refers to the finite interior volume (i.e. a "hole Shell") or the infinite exterior volume (i.e. an "anti-hole Shell").

Each Shell's defining Faces are identified in a list of Faces by an index for both the first Face and the last Face in each Shell (i.e. all Faces inclusive between the specified first and last Face list index define the particular Shell).

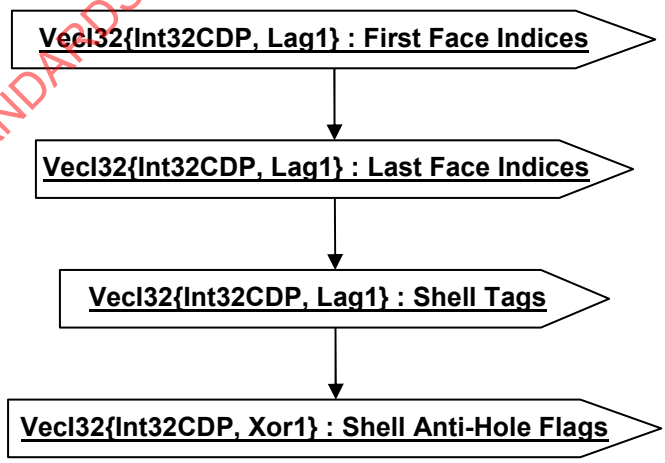


Figure 112 — Shells Topology Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecI32{Int32CDP, Lag1} : First Face Indices

First Face Indices is a vector of indices representing the index of the first Face in each Shell. First Face Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Last Face Indices

Last Face Indices is a vector of indices representing the index of the last Face in each Shell. Last Face Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Shell Tags

Each Shell has an identifier tag. Shell Tags is a vector of identifier tags for a set of Shells. Shell Tags uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Xor1} : Shell Anti-Hole Flags

Each Shell has a flag identifying whether the Shell is an anti-hole Shell. Shell Anti-Hole Flags is a vector of anti-hole flags for a set of Shells.

In an uncompressed/decoded form the flag values have the following meaning:

Table 62 — JT B-Rep Shell Topology Anti-Hole Flag values

= 0	Shell is not an anti-hole Shell
= 1	Shell is an anti-hole Shell

Shell Anti-Hole Flags uses the Int32 version of the CODEC to compress and encode data.

7.1.3.3 Faces Topology Data

A Face is a two-dimensional topological building block defined as the active (that portion to be used in the model) regions/areas of a Geometric Surface; where active regions/areas of a Geometric Surface are indicated using oriented Trim Loops. Faces Topology Data specifies the underlying Geometric Surface and Trim Loops making up each Face along with a “reverse normal” flag and identifier tag for each Face.

A Face shall be trimmed with at least one “anti-hole” Trim Loop and zero or more “hole” Trim Loops. Thus the area of the Geometric Surface defined as the Face, is the area inside the “anti-hole” Trim Loops and outside each “hole” Trim Loop. No Trim Loops (“hole’ or “anti-hole”) may intersect/cross or be tangent at any point. “Anti-Hole” Trim Loops shall be defined with a counter-clockwise orientation in the underlying surface's parameter space whereas “hole” Trim Loops shall be defined with a clockwise orientation. With this Trim Loop orientation definition, as one traverses a Trim Loop of a Face, the material or “active region” is always to one’s left. Figure 113 gives an example in parameter space of proper trim loop definition and orientation (as indicated by the arrows on the loop’s CoEdges) for a face with two holes. “L1” represents the face “anti-hole” Trim Loop while “L2” and L3” represent the two “hole” Trim Loops. Note that each hole is always represented by a separate distinct “hole” Trim Loop.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

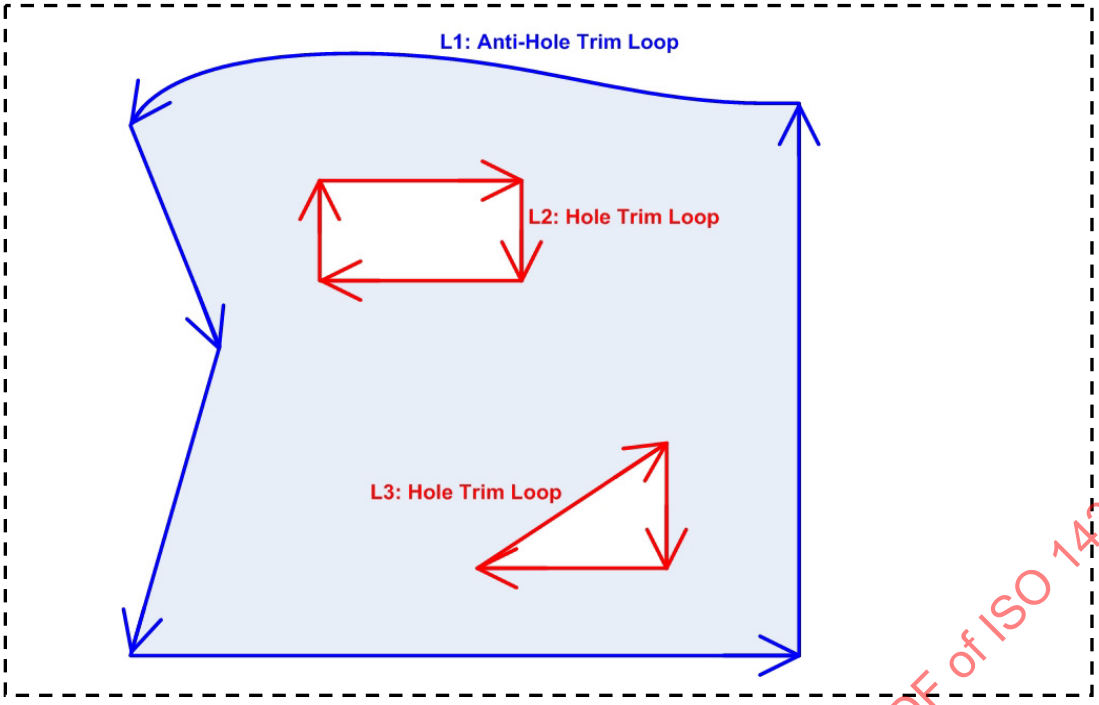


Figure 113 — Trim Loop example in parameter Space - One Face with 2 Holes

Each Face's underlying Geometric Surface is identified by an index into a list of Geometric Surfaces. Each Face's defining Trim Loops are identified in a list of trim Loops by an index for both the first Trim Loop and the last Trim Loop in each Face (i.e. all Trim Loops inclusive between the specified first and last Trim Loop list index define the particular Face).

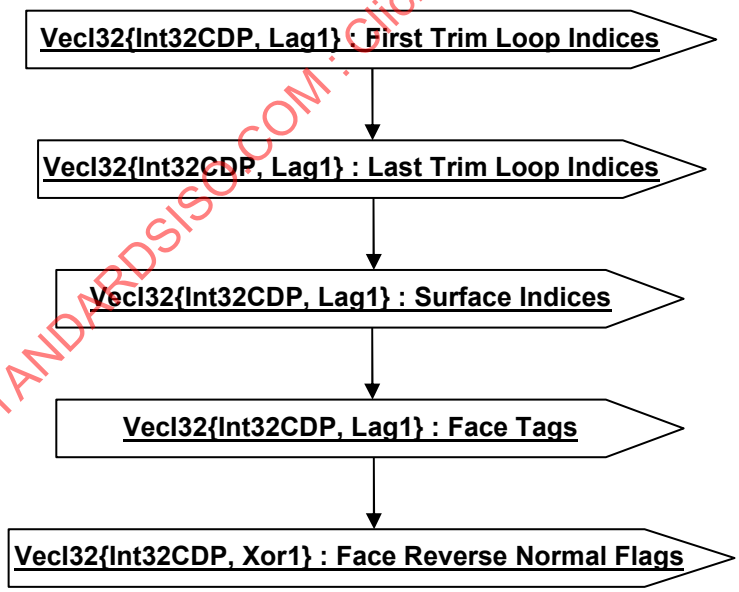


Figure 114 — Faces Topology Data collection

Vecl32{Int32CDP, Lag1} : First Trim Loop Indices

First Trim Loop Indices is a vector of indices representing the index of the first Trim Loop in each Face. First Trim Loop Indices uses the Int32 version of the CODEC to compress and encode data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecI32{Int32CDP, Lag1} : Last Trim Loop Indices

Last Trim Loop Indices is a vector of indices representing the index of the last Trim Loop in each Face. Last Trim Loop Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Surface Indices

Surface Indices is a vector of indices representing the index of the underlying Geometric Surface for each Face. Surface Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Face Tags

Each Face has an identifier tag. Face Tags is a vector of identifier tags for a set of Faces. Face Tags uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Xor1} : Face Reverse Normal Flags

Each Face has a flag identifying whether the Face's normal(s) should be interpreted to point in the direction opposite of the usual U cross V normal (note that these flags do not imply any sort of parameter reversal, the flag only implies that the material is on the other side of the surface).

Face Reverse Normal Flags is a vector of reverse-normal flags for a set of Faces.

In an uncompressed/decoded form the flag values have the following meaning:

Table 63 — JT B-Rep Face Reverse Normal Flag values

= 0	Face normal is not reversed
= 1	Face normal is reversed.

Face Reverse Normal Flags uses the Int32 version of the CODEC to compress and encode data.

7.1.3.4 Loops Topology Data

A Loop (often called Trimming Loop) defines in parameter space a 1D boundary around which geometric surfaces are trimmed to form a Face. Loops Topology Data specifies the CoEdges making up each Loop along with an anti-hole flag and identifier tag for each Loop.

A Loop is composed of one or more CoEdges and the Loop shall be closed and non-self-intersecting.

Each Loop's defining CoEdges are identified in a list of CoEdges by an index for both the first CoEdge and the last CoEdge in each Loop (i.e. all CoEdges inclusive between the specified first and last CoEdge list index define the particular Loop).

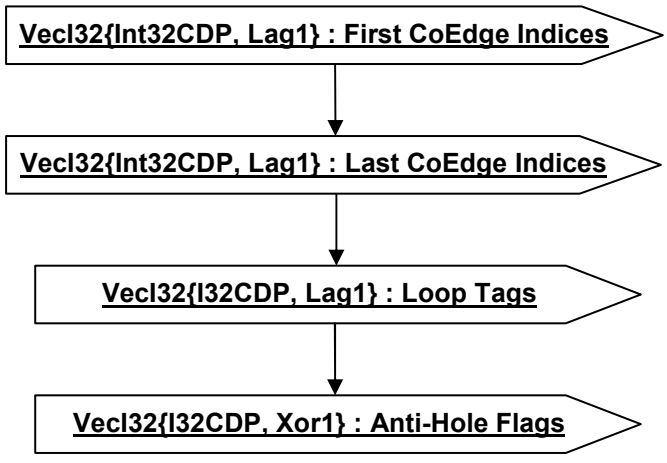


Figure 115 — Loops Topology Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Vecl32{Int32CDP, Lag1} : First CoEdge Indices

First CoEdge Indices is a vector of indices representing the index of the first CoEdge in each Loop. First CoEdge Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : Last CoEdge Indices

Last CoEdge Indices is a vector of indices representing the index of the last CoEdge in each Loop. Last CoEdge Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{I32CDP, Lag1} : Loop Tags

Each Loop has an identifier tag. Loop Tags is a vector of identifier tags for a set of Loops. Loop Tags uses the Int32 version of the CODEC to compress and encode data.

Vecl32{I32CDP, Xor1} : Anti-Hole Flags

Each Loop has a flag identifying whether the Loop is an anti-hole Loop. Anti-Hole Flags is a vector of anti-hole flags for a set of Loops

In an uncompressed/decoded form the flag values have the following meaning:

Table 64 — JT B-Rep Loops Topology Data Anti-Hole Flag values

= 0	Loop is not an anti-hole Loop
= 1	Loop is an anti-hole Loop

Anti-Hole Flags uses the Int32 version of the CODEC to compress and encode data.

7.1.3.5 CoEdges Topology Data

A CoEdge defines a parameter space edge trim Loop segment (i.e. the projection of an Edge into the parameter space of the Face). CoEdges Topology Data specifies the underlying Edge and PCS Curve making up each CoEdge along with a MCS curve reversed flag and tag for each CoEdge.

The “Co” portion of the CoEdge name derives from the manifold topology definition that each Edge has exactly two Faces containing it; thus a CoEdge defines one Face’s “use” of an Edge and the adjoining Face also has a CoEdge (“edge use” in some other terminologies) for the same underlying Edge. This sharing of the same underlying Edge by two adjoining Faces requires an “MCS Curve Reversed Flag” on each CoEdge to indicate the edge traversal direction (i.e. for a proper manifold topology definition each CoEdge shall traverse the Edge in opposite directions).

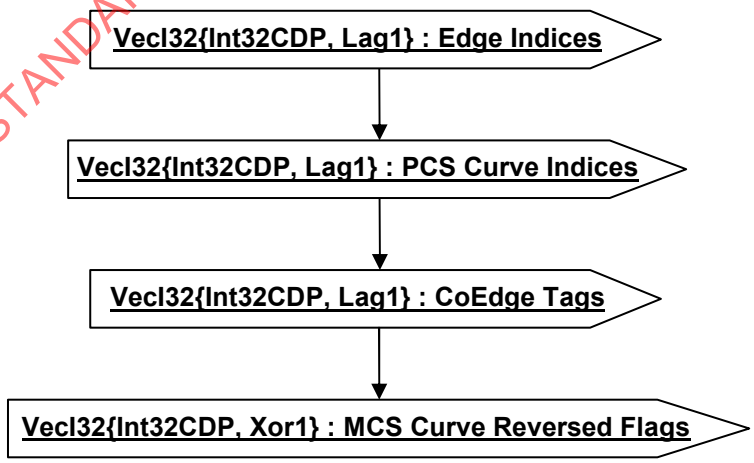


Figure 116 — CoEdges Topology Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Vecl32{Int32CDP, Lag1} : Edge Indices

Edge Indices is a vector of indices representing the index of the underlying Edge for each CoEdge. Edge Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : PCS Curve Indices

PCS Curve Indices is a vector of indices representing the index of the PCS Curve (UV Curve) for each CoEdge. PCS Curve Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : CoEdge Tags

Each CoEdge has an identifier tag. CoEdge Tags is a vector of identifier tags for a set of CoEdges. CoEdge Tags uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Xor1} : MCS Curve Reversed Flags

Each CoEdge has a flag indicating whether the directional sense of the associated Edge's MCS curve should be interpreted as opposite the direction its parameterization implies. MCS Curve Reversed Flags is a vector of reverse flags for a set of CoEdges.

In an uncompressed/decoded form the flag values have the following meaning:

Table 65 — JT B-Rep MCS Curve Reversed Flag values

= 0	Directional sense of associated edges MCS curve should not be interpreted as opposite the direction its parameterization implies.
= 1	Directional sense of associated edges MCS curve should be interpreted as opposite the direction its parameterization implies.

MCS Curve Reversed Flags uses the Int32 version of the CODEC to compress and encode data.

7.1.3.6 Edges Topology Data

An Edge defines a model space trim Loop segment. Edges Topology Data specifies the underlying MCS Curve and start and end Vertex making up each Edge along with an identification tag for each Edge.

If manifold topology, then two faces join at a single model Edge and thus an edge is shared/referenced by two CoEdges (one per Face).

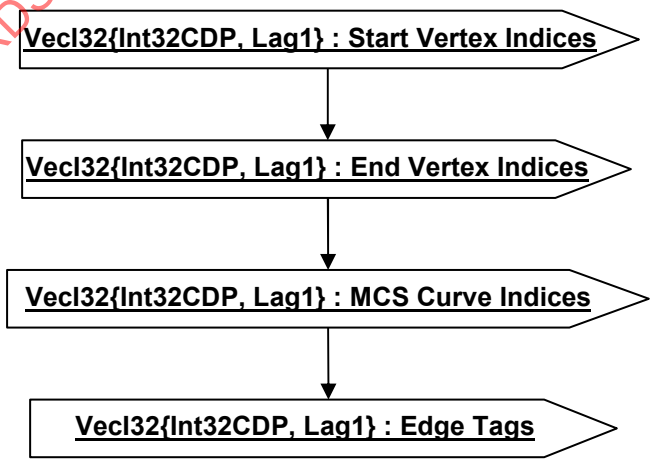


Figure 117 — Edges Topology Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecI32{Int32CDP, Lag1} : Start Vertex Indices

Start Vertex Indices is a vector of indices representing the index of the start Vertex in each Edge. Start Vertex Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : End Vertex Indices

End Vertex Indices is a vector of indices representing the index of the end Vertex in each Edge. End Vertex Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : MCS Curve Indices

MCS Curve Indices is a vector of indices representing the index of the MCS Curve (Model Space curve) for each Edge. MCS Curve Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Edge Tags

Each Edge has an identifier Tag. Edge Tags is a vector of identifier Tags for a set of Edges. Edge Tags uses the Int32 version of the CODEC to compress and encode data.

7.1.3.7 Vertices Topology Data

A Vertex is the simplest topological entity and is basically made up of a geometric Point. Vertices Topology Data specifies the underlying geometric Point making up each Vertex along with an identification tag for each Vertex.

The presence of Vertices Topology Data in a JT B-Rep topology definition is optional. Vertex data is optional because unlike most topological entities, no connectivity information is contained in a Vertex structure and Vertex data is also not necessary for performing operations such as tessellation or mass properties calculations.

A Vertex is usually shared/referenced by two or more Edges (e.g. if the corners of four rectangular Faces touches at a common point, this point is represented by a Vertex and is shared by four Edges).

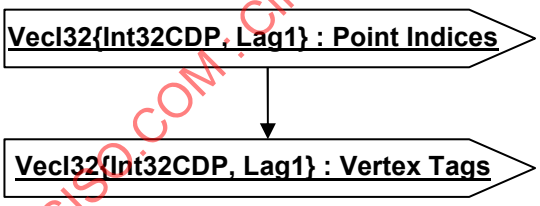


Figure 118 — Vertices Topology Data collection

VecI32{Int32CDP, Lag1} : Point Indices

Point Indices is a vector of indices representing the index of the geometric point for each Vertex. Point Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Vertex Tags

Each Vertex has an identifier Tag. Vertex Tags is a vector of identifier Tags for a set of Vertices. Vertex Tags uses the Int32 version of the CODEC to compress and encode data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

7.1.4 Geometric Data

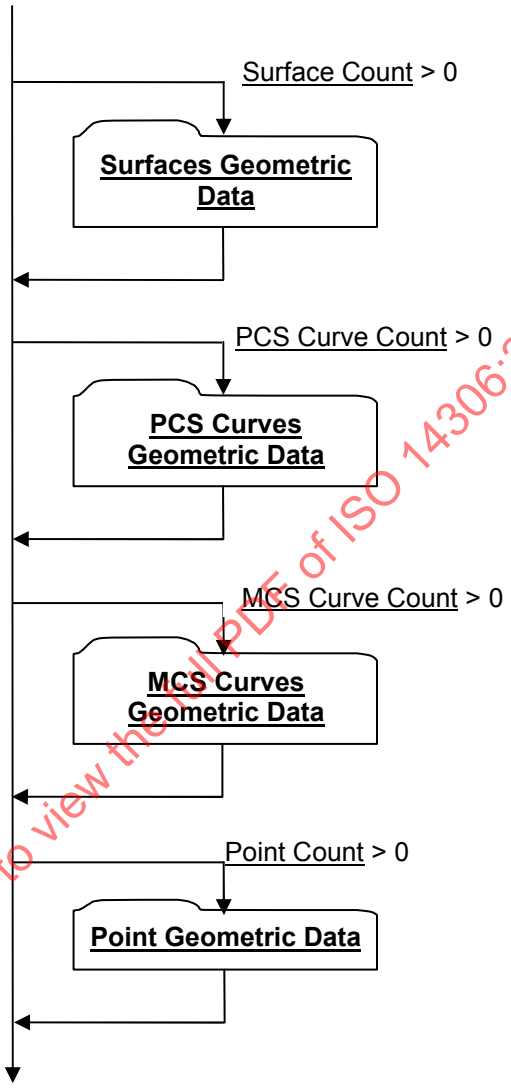


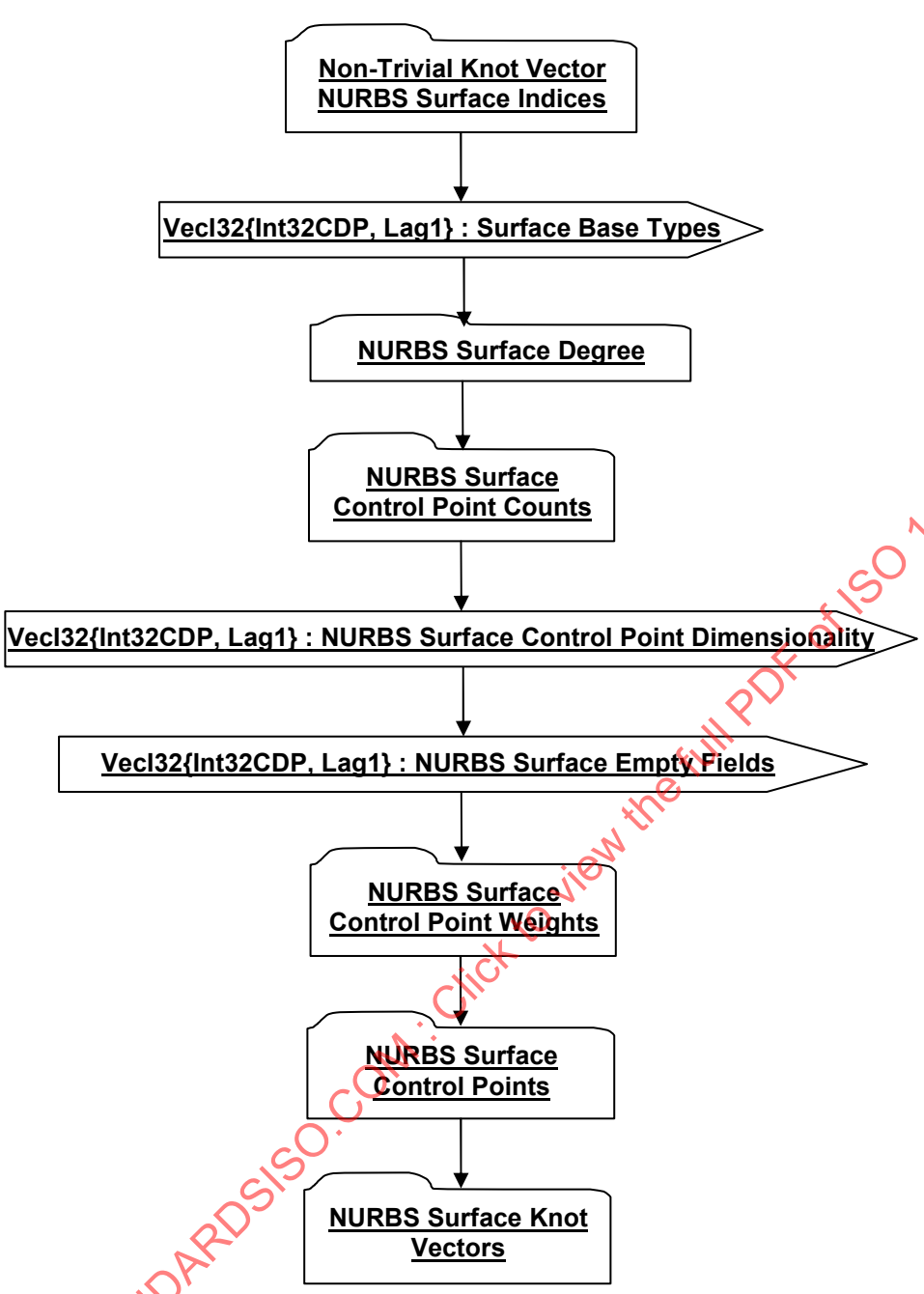
Figure 119 — Geometric Data collection

7.1.4.1 Surfaces Geometric Data

Surfaces Geometric Data collection contains the JT B-Rep's geometric Surface data. Currently only NURBS Surface types are supported within a JT B-Rep. The count/number of Surfaces within a JT B-Rep is indicated by data field Surface Count documented in 7.1.2 Geometric Entity Counts.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 120 — Surfaces Geometric Data collection

Vecl32{Int32CDP, Lag1} : Surface Base Types

Each Surface is assigned a base type identifier. Surface Base Types is a vector of base type identifiers for each Surface in a list of Surfaces. Currently only NURBS Surface Base Type is supported, but a type identifier is still included in the specification to allow for future expansion of the JT Format to support other surface types within a JT B-Rep.

In an uncompressed/decoded form the Surface base type identifier values have the following meaning:

Table 66 — JT B-Rep Surface Base Type value

= 1	Surface is a NURBS surface

© ISO 2012 – All rights reserved

Surface Base Types uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : NURBS Surface Control Point Dimensionality

NURBS Surface Control Point Dimensionality is a vector of control point dimensionality values for each NURBS Surface in a list of Surfaces (i.e. there is a stored values for each NURBS Surface in the list).

In an uncompressed/decoded form dimensionality values have the following meaning:

Table 67 — JT B-Rep NURBS Surface Control Point Demensionality values

= 3	Non-Rational (each control point has 3 coordinates)
= 4	Rational (each control point has 4 coordinates)

NURBS Surface Control Point Dimensionality uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : NURBS Surface Empty Fields

NURBS Surface Empty Fields is a vector of data. Each NURBS Surface in a list of Surfaces has one empty data field entry in this NURBS Surface Empty Fields vector. NURBS Surface Empty Fields uses the Int32 version of the CODEC to compress and encode data. Refer to best practice 14.4 Empty Field.

7.1.4.1.1 Non-Trivial Knot Vector NURBS Surface Indices

Non-Trivial Knot Vector NURBS Surface Indices data collection specifies for both U and V directions the Surface index identifiers (i.e. indices to particular NURBS Surfaces within a list of Surfaces) for all NURBS Surfaces containing non-trivial knot vectors. A description/definition for “non-trivial knot vector” can be found in 13.1.13 Compressed Entity List for Non-Trivial Knot Vector.

This Surface index data is stored in a compressed format.

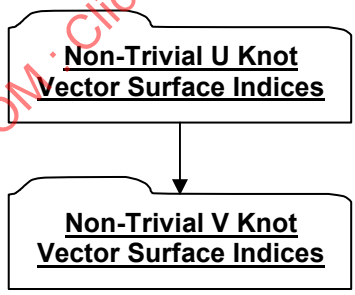


Figure 121 — Non-Trivial Knot Vector NURBS Surface Indices data collection

Both Non-Trivial U Knot Vector Surface Indices and Non-Trivial V Knot Vector Surface Indices have the same data format as that documented for data collection 13.1.13 Compressed Entity List for Non-Trivial Knot Vector.

7.1.4.1.2 NURBS Surface Degree

NURBS Surface Degree data collection defines the Surface degree in both U and V directions for each NURBS Surface in a list of Surfaces (i.e. there are stored values for each NURBS Surface in the list). This degree data for the list of Surfaces is stored in a compressed format.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

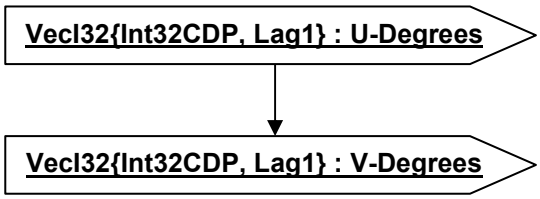


Figure 122 — NURBS Surface Degree data collection

Vecl32{Int32CDP, Lag1} : U-Degrees

U-Degrees is a vector of Surface degree values in U direction for each NURBS Surface in a list of Surfaces. U-Degrees uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : V-Degrees

V -Degrees is a vector of Surface degree values in V direction for each NURBS Surface in a list of Surfaces. V-Degrees uses the Int32 version of the CODEC to compress and encode data.

7.1.4.1.3 NURBS Surface Control Point Counts

NURBS Surface Control Point Counts defines the number of NURBS Surface control points for both U and V directions for each NURBS Surface in a list of Surfaces (i.e. there are stored values for each NURBS Surface in the list). The control point count data for the list of Surfaces in stored in a compressed format.

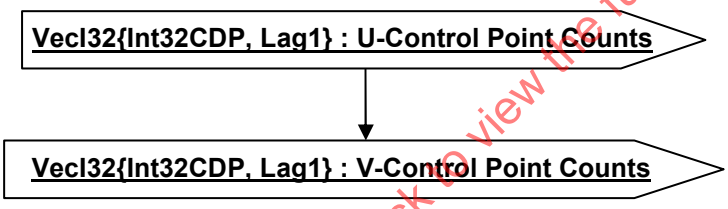


Figure 123 — NURBS Surface Control Point Counts data collection

Vecl32{Int32CDP, Lag1} : U-Control Point Counts

U-Control Point Counts is a vector of control point counts in U direction for each NURBS Surface in a list of Surfaces. U-Control Point Counts uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : V-Control Point Counts

V-Control Point Counts is a vector of control point counts in V direction for each NURBS Surface in a list of Surfaces. V-Control Point Counts uses the Int32 version of the CODEC to compress and encode data.

7.1.4.1.4 NURBS Surface Control Point Weights

NURBS Surface Control Point Weights data collection defines the Weight values for a conditional set of Control Points for a list of NURBS Surfaces. The storing of the Weight value for a particular Control Point is conditional, because if NURBS Surface Control Point Dimension is “non-rational” or the actual Control Point’s Weight value is “1”, then no Weight value is stored for the Control Point (i.e. Weight value can be inferred to be “1”).

The NURBS Surface Control Point Weights data is stored in a compressed format.

Figure 124 — NURBS Surface Control Point Weights data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Complete description for Compressed Control Point Weights Data can be found in 13.1.14 Compressed Control Point Weights Data.

7.1.4.1.5 NURBS Surface Control Points

NURBS Surface Control Points is the compressed and/or encoded representation of the Control Point coordinates for each NURBS Surface in a list of Surfaces (i.e. there are stored values for each NURBS Surface in the list). Note that these are non-homogeneous coordinates (i.e. Control Point coordinates have been divided by the corresponding Control Point Weight values).

Figure 125 — NURBS Surface Control Points data collection

VecF64{Float64CDP, NULL} : Control Points

Control Points is a vector of Control Point coordinates for all the NURBS Surfaces in a list of Surfaces. All the NURBS Surfaces Control Point coordinates are cumulated into this single vector in the same order as the Surface appears in the Surface list (i.e. Surface-1 U Control Points, Surface-1 V Control Points, Surface-2 U Control Points, Surface-2 V Control Points, etc.). Control Points uses the Float64 version of the CODEC to compress and encode data in a “lossless” manner.

7.1.4.1.6 NURBS Surface Knot Vectors

NURBS Surface Knot Vectors defines the knot vectors for both U and V directions for each NURBS Surface having non-trivial knot vectors in a list of Surfaces (i.e. there are stored values for each non-trivial knot vector NURBS Surface in the list). The NURBS Surfaces for which knot vectors are stored (i.e. those containing non-trivial knot vectors) are identified in data collection Non-Trivial Knot Vector NURBS Surface Indices documented in 7.1.4.1.1 Non-Trivial Knot Vector NURBS Surface Indices.

The knot vector data for the list of Surfaces is stored in a compressed format.

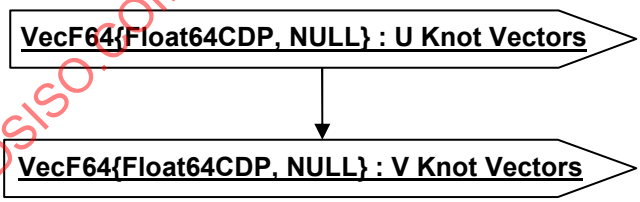


Figure 126 — NURBS Surface Knot Vectors data collection

VecF64{Float64CDP, NULL} : U Knot Vectors

U Knot Vectors is a list of knot vector values in U direction for each NURBS Surface having non-trivial knot vectors in a list of Surfaces. All these NURBS Surface U direction non-trivial knot vectors are cumulated into this single list in the same order as the Surface appears in the Surface list (i.e. Surface-N Non-Trivial U Knot Vector, Surface-M Non-Trivial U Knot Vector, etc.). U Knot Vectors uses the Float64 version of the CODEC to compress and encode data.

VecF64{Float64CDP, NULL} : V Knot Vectors

V Knot Vectors is a list of knot vector values in V direction for each NURBS Surface having non-trivial knot vectors in a list of Surfaces. All these NURBS Surface V direction non-trivial knot vectors are cumulated into this single list in the same order as the Surface appears in the Surface list (i.e. Surface-N Non-Trivial V Knot Vector, Surface-M Non-Trivial V Knot Vector, etc.). V Knot Vectors uses the Float64 version of the CODEC to compress and encode data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

7.1.4.2 PCS Curves Geometric Data

PCS Curves Geometric Data collection contains the JT B-Rep's Parameter Coordinate Space geometric Curve data (i.e. UV Curve data). This geometric PCS Curve data is divided up into two collection types; one data collection for what are considered "Trivial" PCS curves and one data collection for compressed/encoded PCS NURBS Curve data.

"Trivial" PCS Curves are those UV Curves whose definition is such that the actual UV Curve definition can be derived from the parametric domain definition by storing a limited amount of descriptive data for each UV curve (i.e. do not have to store the complete NURBS UV Curve definition).

The count/number of PCS Curves within a JT B-Rep is indicated by data field PCS Curve Count documented in 7.1.2 Geometric Entity Counts.

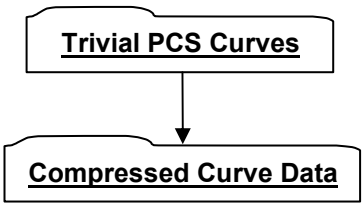


Figure 127 — PCS Curves Geometric Data collection

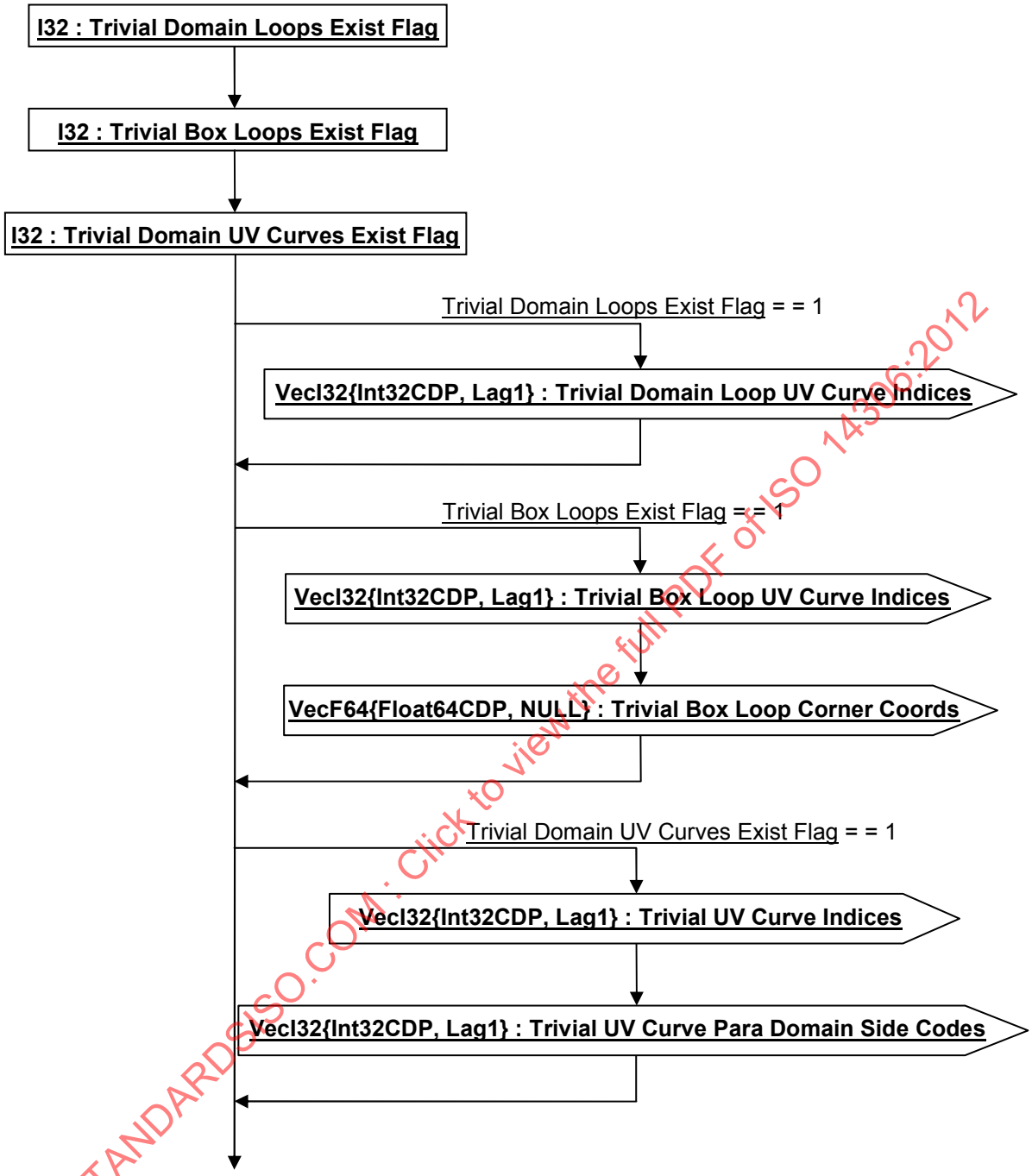
Complete description for Compressed Curve Data can be found in 13.1.15 Compressed Curve Data.

7.1.4.2.1 Trivial PCS Curves

Trivial PCS Curves data collection represents those UV curves whose definition is such (i.e. "trivial" enough) that the actual UV curve definition can be derived from the parametric domain definition by storing a limited amount of descriptive data for each UV curve (i.e. do not have to store the complete UV curve definition). These Trivial PCS Curves are grouped into three classifications (Trivial Domain Loop, Trivial Box Loop, or Trivial Domain UV Curve) and stored as described in the following sub-sections.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 128 — Trivial PCS Curves data collection

I32 : Trivial Domain Loops Exist Flag

Trivial Domain Loops Exist Flag is a flag indicating whether “trivial” domain loops exist/follow. A Trivial Domain Loop is a Loop that encloses the entire parametric domain. (i.e. all UV Curves of the Loop span the entire length of the Surface parametric domain). Given this criteria a Trivial Domain Loop shall always be made up of four Trivial Domain UV curves.

Table 68 — Trivial Domain Loops Exist Flag values

= 0	Trivial Domain Loops do not exist.
= 1	Trivial Domain Loops exist.

© ISO 2012 – All rights reserved

I32 : Trivial Box Loops Exist Flag

Trivial Box Loops Exist Flag is a flag indicating whether “trivial” box loops exist/follow. A trivial Box Loop is a Loop that forms a rectangle (i.e. corresponding curve end coordinates of opposite sides of the box are equal). Given this criteria a Trivial Box Loop shall always be made up of four UV curves

Table 69 — Trivial Box Loops Exist Flag values

= 0	Trivial Box Loops do not exist.
= 1	Trivial Box Loops exist.

“Equality of corresponding curve end coordinates of opposite sides of the box” is represented graphically in Figure 129.

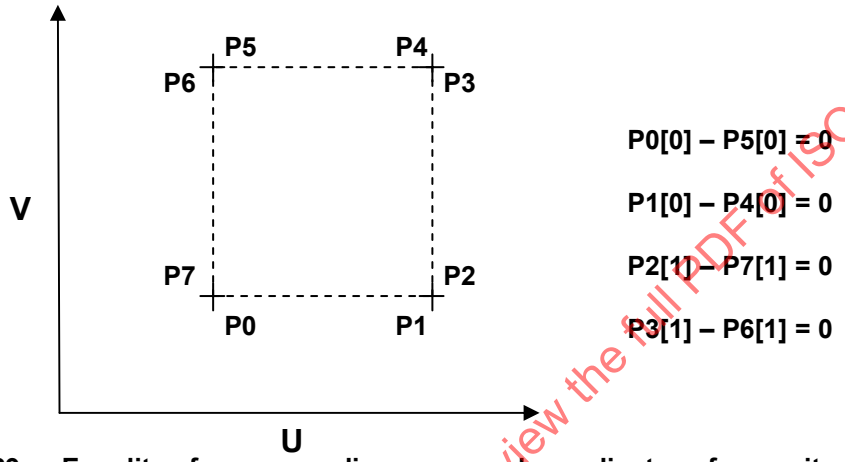


Figure 129 — Equality of corresponding curve end coordinates of opposite sides of the box

I32 : Trivial Domain UV Curves Exist Flag

Trivial Domain UV Curves Exist Flag is a flag indicating whether “trivial” domain UV curves (Loop CoEdges) exist/follow that are not part of a Trivial Domain Loop or Trivial Box Loop (i.e. a Loop contains some UV curves that span the entire length of the Surface parametric domain but not all the Loop UV curves meet this criteria and thus not captured as part of the Trivial Domain Loop data).

Table 70 — Trivial Domain UV Curves Exist Flag values

= 0	Trivial Domain UV Curves do not exist.
= 1	Trivial Domain UV Curves exist.

VecI32{Int32CDP, Lag1} : Trivial Domain Loop UV Curve Indices

Trivial Domain Loop UV Curve Indices is a vector of all UV curve indices that are part of a Trivial Domain Loop. Note that each Trivial Domain Loop is always made up of four UV curves (thus four UV curve indices per Loop). Trivial Domain Loop UV Curve Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : Trivial Box Loop UV Curve Indices

Trivial Box Loop UV Curve Indices is a vector of all UV Curve indices that are part of a Trivial Box Loop. Note that each Trivial Box Loop is always made up of four UV Curves (thus four UV Curve indices per Loop). Trivial Box Loop UV Curve Indices uses the Int32 version of the CODEC to compress and encode data.

VecF64{Float64CDP, NULL} : Trivial Box Loop Corner Coords

Trivial Box Loop Corner Coords is a vector of box corner coordinates for all Trivial Box Loops (i.e. each Box Loop will store two box coroner coordinates). A Box Loop’s set of “box corner coordinates” are the

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

coordinates of the two min/max diagonally opposite corners of the box. Note that if the Box Loop is a “hole”, then the max and min corners are the other ends of the respective box sides that contain the max and min corners. Trivial Box Loop Corner Coords uses the Float64 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : Trivial UV Curve Indices

Trivial UV Curve Indices is a vector of all Loop UV Curve indices that are not part of a Trivial Domain Loop or Trivial Box Loop. Trivial UV Curve Indices uses the Int32 version of the CODEC to compress and encode data.

Vecl32{Int32CDP, Lag1} : Trivial UV Curve Para Domain Side Codes

Trivial UV Curve Para Domain Side Codes is a vector containing a “side code” for each Trivial UV Curve indicating which parametric domain side the UV Curve lies on.

In an uncompressed/decoded form the parametric domain side values have the following meaning:

Table 71 — Trivial UV Curve Para Domain Side Codes values

= 0	Bottom side of parametric domain
= 1	Right side of parametric domain
= 2	Top side of parametric domain
= 3	Left side of parametric domain

Trivial UV Curve Para Domain Side Codes uses the Int32 version of the CODEC to compress and encode data.

7.1.4.2.2 MCS Curves Geometric Data

MCS Curves Geometric Data collection contains the JT B-Rep’s Model Coordinate System geometric Curve data (i.e. XYZ Curve data). Currently only NURBS Curve types are supported within a JT B-Rep. The count/number of MCS Curves within a JT B-Rep is indicated by data field MCS Curve Count documented in 7.1.2 Geometric Entity Counts.

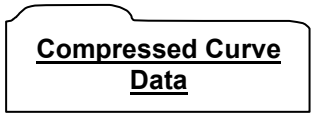


Figure 130 — MCS Curves Geometric Data collection

Complete description for Compressed Curve Data can be found in 13.1.15 Compressed Curve Data.

7.1.4.2.3 Point Geometric Data

Point Geometric Data collection contains the JT B-Rep’s geometric Point data. Each Point is simply represented by a CoordF32 for the Point’s coordinate components. The count/number of Points within a JT B-Rep is indicated by data field Point Count documented in 7.1.2 Geometric Entity Counts.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

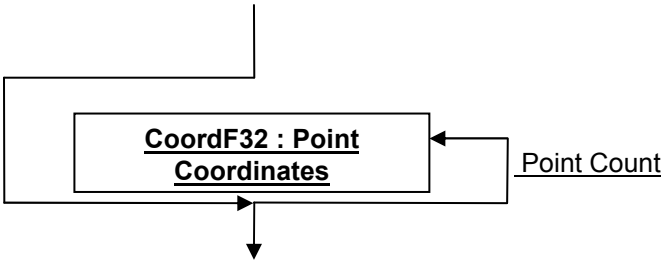


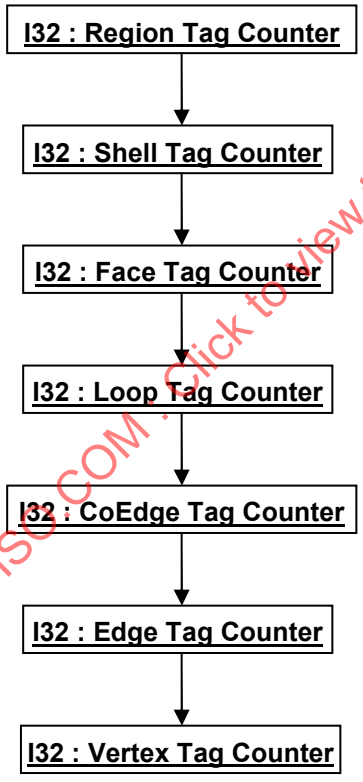
Figure 131 — Point Geometric Data collection

CoordF32 : Point Coordinates

Point Coordinates specifies the XYZ coordinate components for a Point.

7.1.5 Topological Entity Tag Counters

Topological Entity Tag Counters data collection specifies the next available “unique” tag value for each entity type in a JT B-Rep. These are rolling tag counters that are meant to be used for assigning a unique tag when a new entity is added to a JT B-Rep.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 132 — Topological Entity Tag Counters data collection

I32 : Region Tag Counter

Region tag Counter specifies the next available “unique’ tag value for Region entity.

I32 : Shell Tag Counter

Shell Tag Counter specifies the next available “unique’ tag value for Shell entity.

I32 : Face Tag Counter

Face Tag Counter specifies the next available “unique’ tag value for Face entity.

© ISO 2012 – All rights reserved

I32 : Loop Tag Counter

Loop Tag Counter specifies the next available “unique’ tag value for Loop entity.

I32 : CoEdge Tag Counter

CoEdge Tag Counter specifies the next available “unique’ tag value for CoEdge entity.

I32 : Edge Tag Counter

Edge Tag Counter specifies the next available “unique’ tag value for Edge entity.

I32 : Vertex Tag Counter

Vertex Tag Counter specifies the next available “unique’ tag value for Vertex entity.

7.1.6 B-Rep CAD Tag Data

The B-Rep CAD Tag Data collection contains the list of persistent IDs, as defined in the CAD System, to uniquely identify individual Faces and Edges in the JT B-Rep. The existence of this B-Rep CAD Tag Data collection is dependent upon the value of previously read data field CAD Tags Flag as documented in 7.1 JT B-Rep Element.

If B-Rep CAD Tag Data collection is present, there will be a CAD Tag for every Face and every Edge in the JT B-Rep and the list order will be Face CAD Tags followed by Edge CAD Tags. Therefore the total number of CAD Tags in the list should be equal to “Face Count + Edge Count” as documented in 7.1.1 Topological Entity Counts.

Figure 133 — B-Rep CAD Tag Data collection

Complete description for Compressed CAD Tag Data can be found in 13.1.16 Compressed CAD Tag Data.

8 XT B-Rep Segment

XT B-Rep Segment contains an Element that defines the precise geometric Boundary Representation data for a particular Part in XT boundary representation format. Note that there is also another Boundary Representation format (i.e. JT B-Rep) supported by the JT file format within a different file Segment Type. Complete description for the JT B-Rep can be found in 7 JT B-Rep Segment.

XT B-Rep Segments are typically referenced by Part Node Elements (see 5.1.1.5 Part Node Element) using Late Loaded Property Atom Elements (see 5.2.7 Late Loaded Property Atom Element). The XT B-Rep Segment type supports ZLIB compression on all element data, so all elements in XT B-Rep Segment use the Logical Element Header ZLIB form of element header data.

8.1 XT B-Rep Element

Object Type ID: 0x873a70e0, 0x2ac9, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

XT B-Rep Element represents a particular part’s precise data in XT boundary representation) format.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

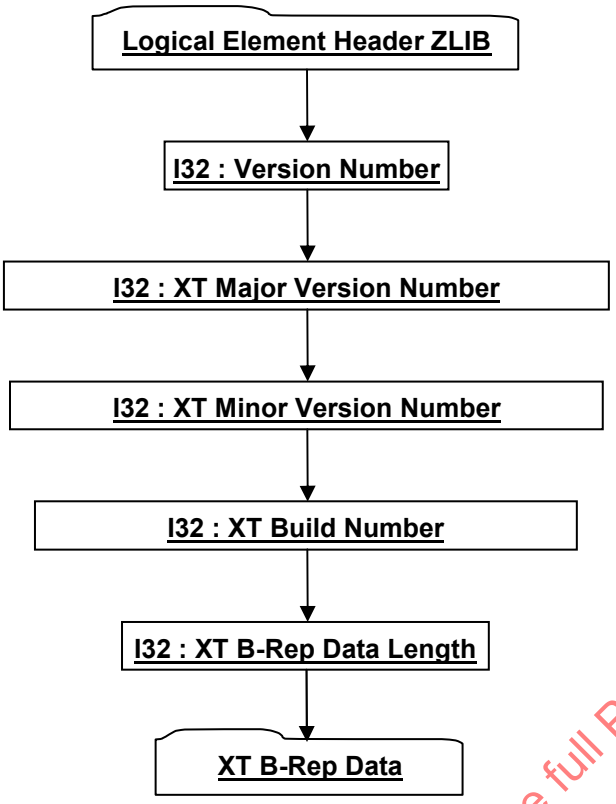


Figure 134 — XT B-Rep Element data collection

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

I32 : Version Number

Version Number is the version identifier for this XT B-Rep Element. Version number is an informative field which can be set to 0 without negative impact to implementation.

I32 : XT Major Version Number

XT Major Version Number specifies the major version number for the for the XT B-Rep data in the JT File. Major version number is an informative field which can be set to 0 without negative impact to implementation.

I32 : XT Minor Version Number

XT Minor Version Number specifies the minor version number for the the XT B-Rep data in the JT File. Minor version number is an informative field which can be set to 0 without negative impact to implementation.

I32 : XT Build Number

XT Build Number specifies the build number for the XT B-Rep data in the JT File. XT build number is an informative field which can be set to 0 without negative impact to implementation.

I32 : XT B-Rep Data Length

XT B-Rep Data Length specifies the length in bytes of the XT B-Rep Data collection. A JT file loader/reader may use this information to compute the end position of the XT B-Rep Data within the JT file and thus skip (for whatever reason) reading the remaining XT B-Rep Data.

8.1.1 XT B-Rep Data

The XT B-Rep Data collection specifies the raw stream of bytes used to represent a Part's XT B-Rep Body(s) Complete documentation for the XT "Neutral Binary" encoding format can be found in Annex E.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

9 Wireframe Segment

Wireframe Segment contains an Element that defines the precise 3D wireframe data for a particular Part. A Wireframe Segment is typically referenced by a Part Node Element (see 5.1.1.5 Part Node Element) using a Second specifies the date Second value. Valid values are [0, 59] inclusive.

Late Loaded Property Atom Element (see 5.2.7 Late Loaded Property Atom Element). The Wireframe Segment type supports ZLIB compression on all element data, so all elements in Wireframe Segment use the Logical Element Header ZLIB form of element header data.

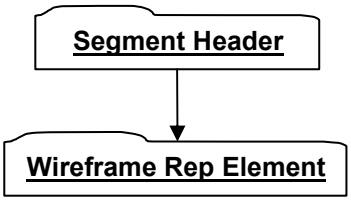


Figure 135 — Wireframe Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

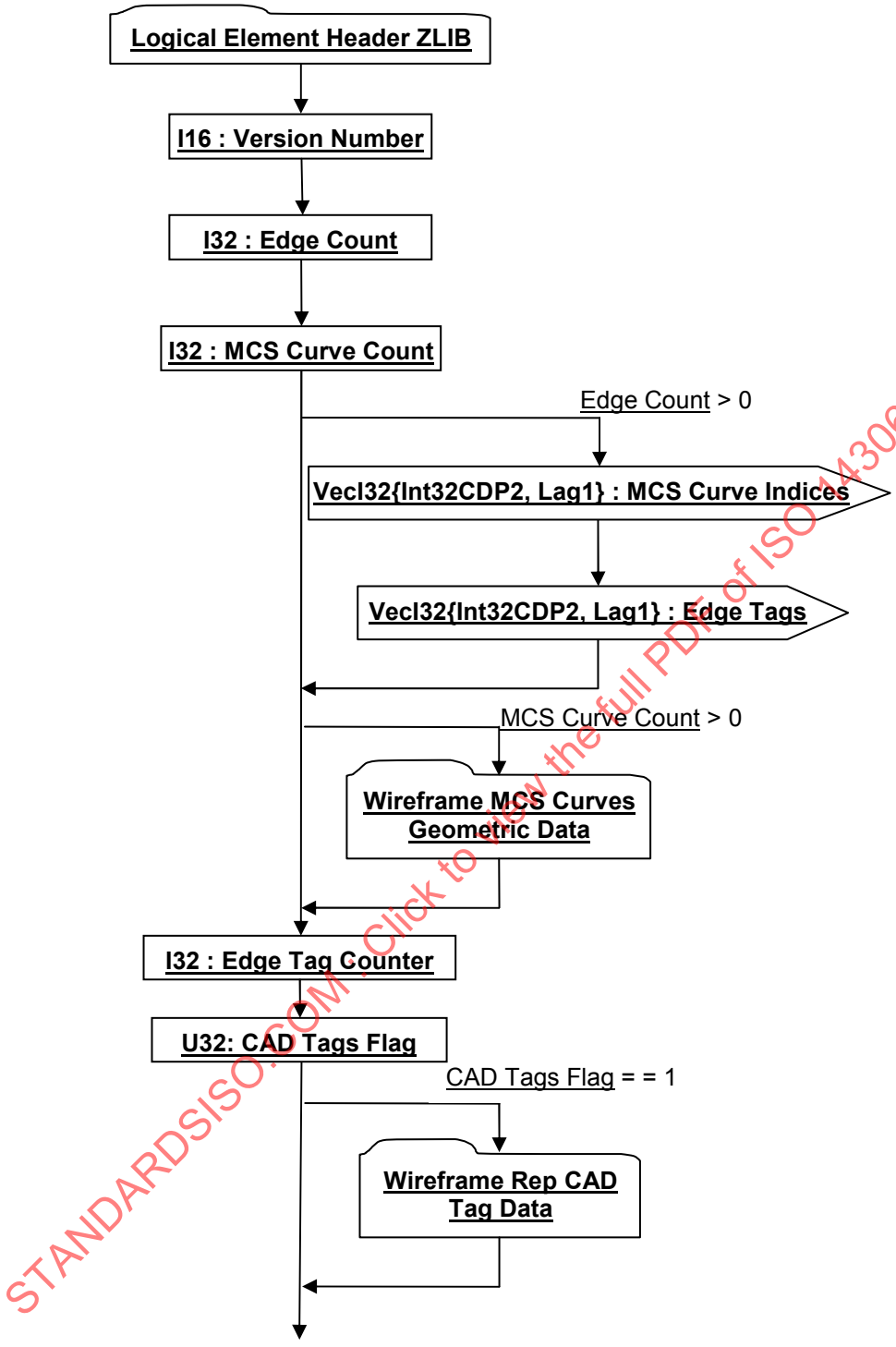
9.1 Wireframe Rep Element

Object Type ID: 0x873a70d0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A Wireframe Rep Element represents a particular Part's precise 3D wireframe data (e.g. reference curves, section curves). Much of the "heavyweight" data contained within a Wireframe Rep Element is compressed and/or encoded. The compression and/or encoding state is indicated through other data stored in each Wireframe Rep Element.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 136 — Wireframe Rep Element data collection

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

I16 : Version Number

Version Number is the version identifier for this JT Wireframe Rep Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

© ISO 2012 – All rights reserved

I32 : Edge Count

Edge Count indicates the number of topological Edge entities in the Wireframe Rep

I32 : MCS Curve Count

MCS Curve Count indicates the number of distinct geometric (Model Coordinate Space) curves (i.e. XYZ curve) entities in the Wireframe Rep.

VecI32{Int32CDP2, Lag1} : MCS Curve Indices

MCS Curve Indices is a vector of indices representing the index of the MCS Curve (Model Space curve) for each Edge. MCS Curve Indices uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP2, Lag1} : Edge Tags

Each Edge has an identifier Tag. Edge Tags is a vector of identifier Tags for a set of Edges. Edge Tags uses the Int32 version of the CODEC to compress and encode data.

I32 : Edge Tag Counter

Edge Tag Counter specifies the next available "unique' tag value for Edge entity.

U32: CAD Tags Flag

CAD Tags Flag is a flag indicating whether CAD Tag data exist for the Wireframe Rep.

9.1.1 Wireframe MCS Curves Geometric Data

Wireframe MCS Curves Geometric Data collection contains the Wireframe Rep's Model Coordinate System geometric Curve data (i.e. XYZ Curve data). Currently only NURBS Curve types are supported within a Wireframe Rep. The count/number of MCS Curves within a Wireframe Rep is indicated by data field MCS Curve Count documented in 9.1 Wireframe Rep Element.

Figure 137 — Wireframe MCS Curves Geometric Data collection

Complete description for Compressed Curve Data can be found in 13.1.15 Compressed Curve Data.

9.1.2 Wireframe Rep CAD Tag Data

The Wireframe Rep CAD Tag Data collection contains the list of persistent IDs, as defined in the CAD System, to uniquely identify individual Edges in the Wireframe Rep. The existence of this Wireframe Rep CAD Tag Data collection is dependent upon the value of previously read data field CAD Tags Flag as documented in 9.1 Wireframe Rep Element.

If Wireframe Rep CAD Tag Data collection is present, there will be a CAD Tag for every Edge in the Wireframe Rep. Therefore the total number of CAD Tags in the list should be equal to "Edge Count" as documented in 9.1 Wireframe Rep Element.

Figure 138 — Wireframe Rep CAD Tag Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Complete description for Compressed CAD Tag Data can be found in 13.1.16 Compressed CAD Tag Data.

10 Meta Data Segment

Meta Data Segments are 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 (i.e. JT file reader can be structured to support the "best practice" of delaying the loading/reading of the referenced meta-data segment until it is actually needed).

Meta Data Segments are typically referenced by Part Node Elements (see 5.1.1.5 Part Node Element) using Late Loaded Property Atom Elements (see 5.2.7 Late Loaded Property Atom Element).

The Meta Data Segment type supports ZLIB compression on all element data, so all elements in Meta Data Segment use the Logical Element Header ZLIB form of element header data.

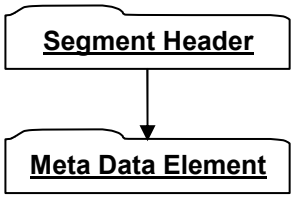


Figure 139 — Meta Data Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

The following sub-sections document the various I32 : Texture Coord Channel types.

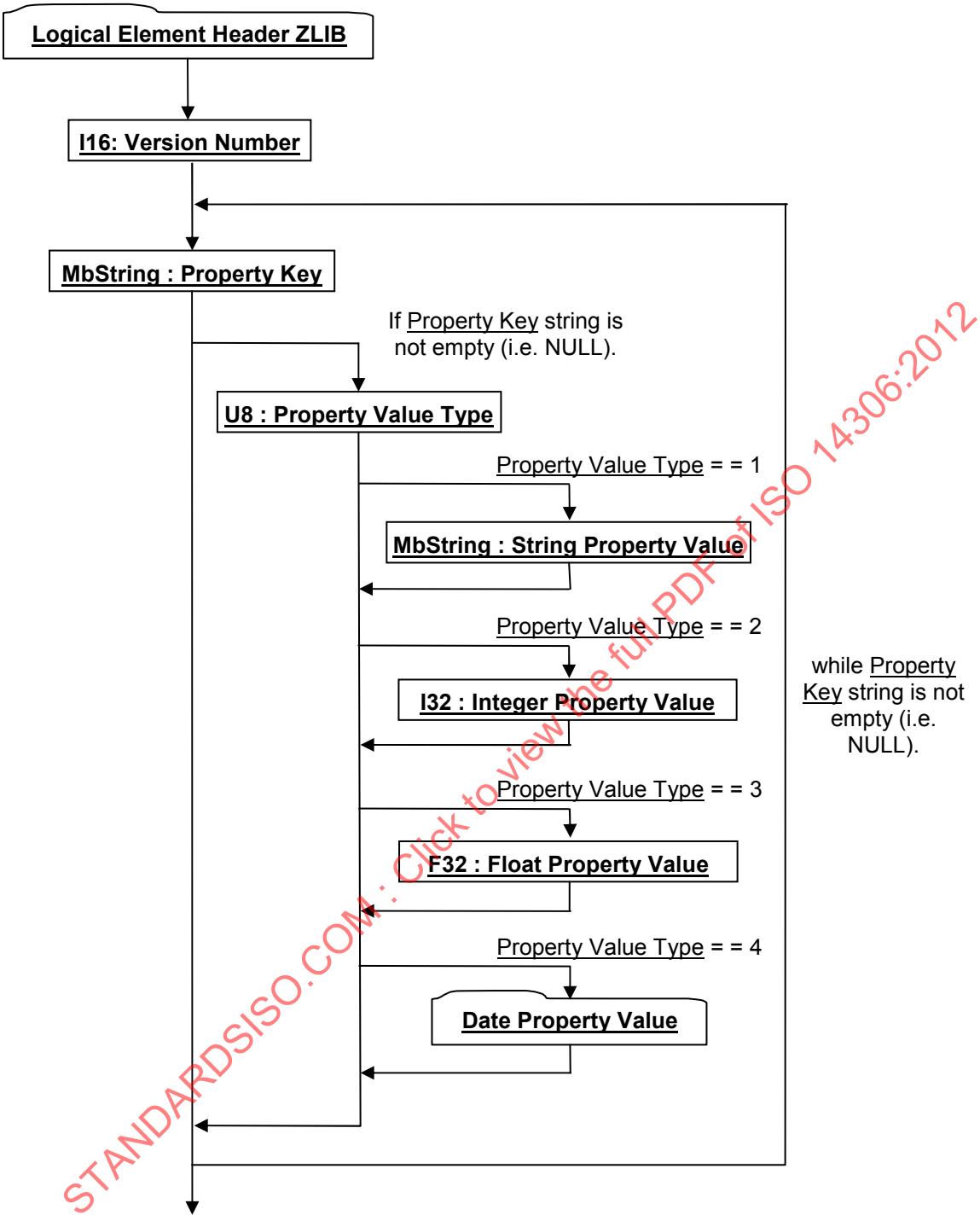
10.1 Property Proxy Meta Data Element

Object Type ID: 0xce357247, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1

A Property Proxy Meta Data Element serves as a "proxy" for all meta-data properties associated with a particular Meta Data Node Element (see 5.1.1.6 Meta Data Node Element). The proxy is in the form of a list of key/value property pairs where the *key* identifies the type and meaning of the *value*. Although the property *key* is always in the form of a String data type, the *value* can be one of several data types.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 140 — Meta Data Segment data collection

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

I16: Version Number

Version Number is the version identifier for this data collection. For informaiton on local version numbers see best practice 14.5 Local version numbers

MbString : Property Key

Property Key specifies the *key* string for the property.

© ISO 2012 – All rights reserved

U8 : Property Value Type

Property Value Type specifies the data type for the Property Value. If the type equals “0” then no Property Value is written. Valid types include the following:

Table 72 — Property Proxy Meta Data Property Value Type values

= 0	Unknown
= 1	MbString data type value
= 2	I32 data type value
= 3	F32 data type value
= 4	Date value

MbString : String Property Value

String Property Value represents the property value when Property Value Type = = 1.

I32 : Integer Property Value

Integer Property Value represents the property value when Property Value Type = = 2.

F32 : Float Property Value

Float Property Value represents the property value when Property Value Type = = 3.

10.1.1 Date Property Value

Date Property Value represents the property value when Property Value Type = = 4. Date Property Value data collection represents a date as a combination of year, month, day, hour, minute, and second data fields.

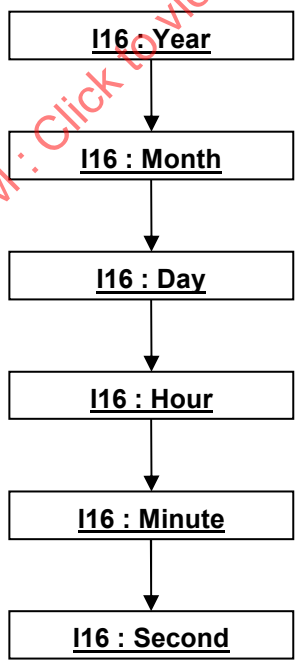


Figure 141 — Date Property Value data collection

I16 : Year

Year specifies the date year value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Month

Month specifies the date month value.

I16 : Day

Day specifies the date day value.

I16 : Hour

Hour specifies the date hour value.

I16 : Minute

Minute specifies the date minute value.

I16 : Second

Second specifies the date Second value.

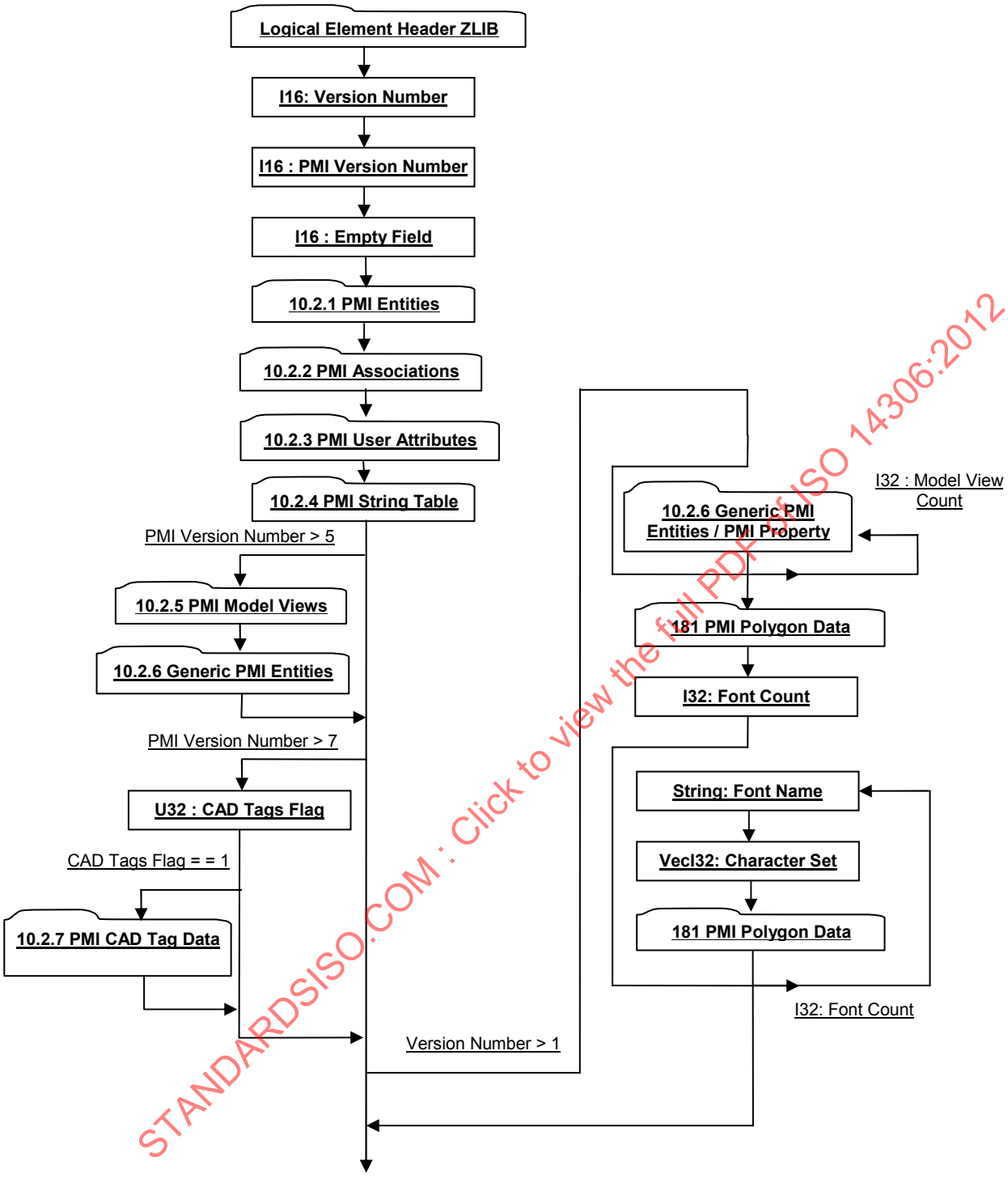
10.2 PMI Manager Meta Data Element

Object Type ID: 0xce357249, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1

The PMI Manager Meta Data Element data collection is a type of I32.Texture Coord Channel which contains the Product and Manufacturing Information for a part/assembly.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 142 — PMI Manager Meta Data Element data collection

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

I16: Version Number

Version Number is the version identifier for this PMI Manager Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

I16 : PMI Version Number

PMI Version Number is the identifier that defines which classes of PMI data will be available in the PMI Manager Meta Data Element. PMI Version Number data is additive.

© ISO 2012 – All rights reserved

The valid PMI version numbers for V9.5 JT are as follows:

Table 73 — PMI Manager Meta Data Version Number values

= 3	Version 3 includes; • I32 : Attribute Count, Design Group Attribute (see Figure 165 — PMI Design Group Entities data collection)
= 4	Version 4 includes version 3 plus; • CoordF32 : Weld Point, DirF32 : Approach Direction, DirF32 : Clamping Direction, DirF32 : Normal Direction (see Figure 159 — PMI Spot Weld Entities data collection) • CoordF32 : Location, DirF32 : Measurement Direction, DirF32 : Coordinate Direction, DirF32 : Normal Direction (see Figure 162 — PMI Measurement Point Entities data collection)
= 5	Version 5 Includes version 4 plus; • U8 : Symbol Valid Flag (see Figure 146 — PMI Base Data collection) I32 : Polyline Type Count, I16 : Polyline Type (see Figure 153 — Non-Text Polyline Data collection)
= 6	Version 6 includes version 5 plus; • 10.2.5 PMI Model Views, 10.2.6 Generic PMI Entities (see Figure 142 — PMI Manager Meta Data Element data collection) • U32 : URL Flag (see Figure 154 — PMI Note Entities data collection) • I32 : Source Owning Entity String ID, I32 : Destination Owning Entity String ID (see Figure 168 — PMI Associations data collection)
= 7	**Version 7 includes version 6 plus;** • U16 : User Flags (see Figure 172 — Generic PMI Entities data collection) • U32 : Hidden Flag (Figure 174 — PMI Property Atom data collectionsee)
= 8	Version 8 Includes version 7 plus; • U32 : CAD Tags Flag (see Figure 142 — PMI Manager Meta Data Element data collection)

I16 : Empty Field

Refer to best practice 14.4 Empty Field

U32 : CAD Tags Flag

CAD Tags Flag is a flag indicating whether CAD Tag data exist for the PMI.

I32: Font Count

Number of sets of glyph definitions. Each set of glyphs represents a single font definition that consists of a name, a character set and polygonal glyph definition for each character in the set.

String: Font Name

Font name specifies a representative name for the font set.

VecI32: Character Set

Integer identifiers for each character whose symbol is defined in the ensuing PolygonData segment.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

10.2.1 PMI Entities

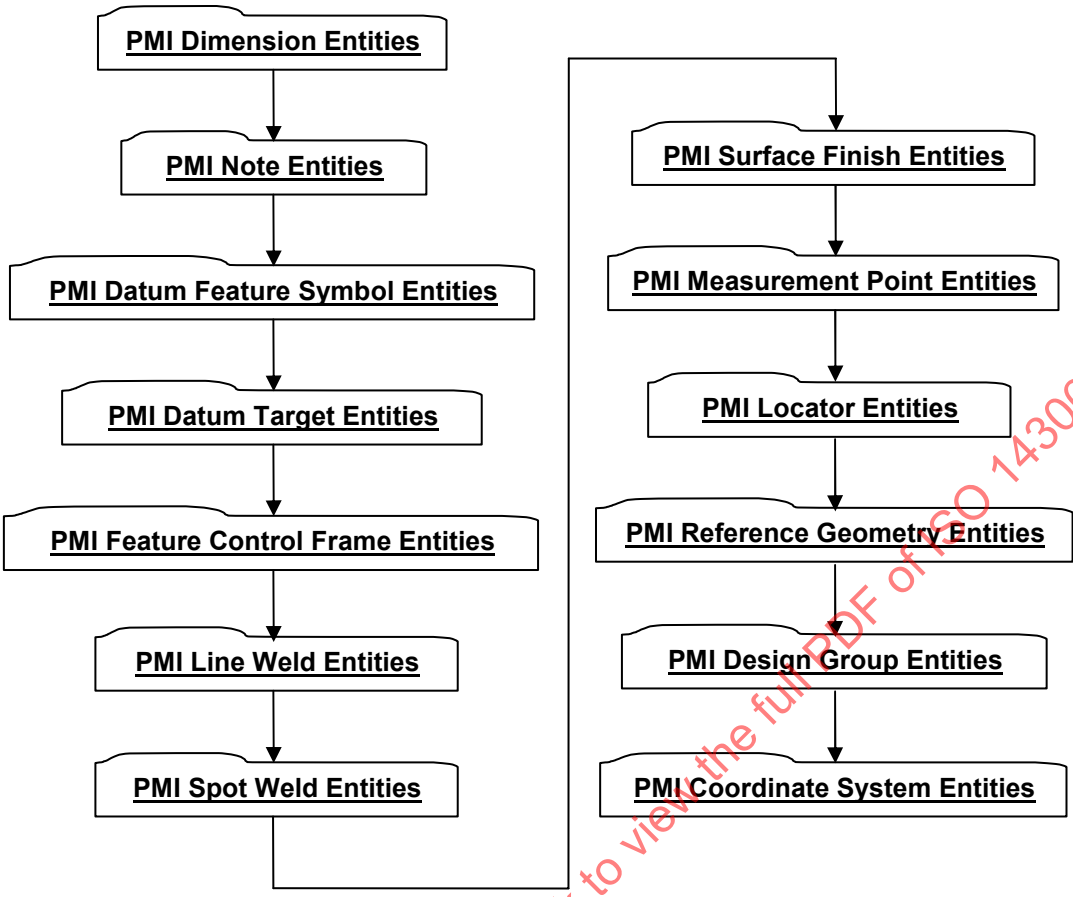


Figure 143 — PMI Entities data collection

10.2.1.1 PMI Dimension Entities

The PMI Dimension Entities data collection defines data for a list of Dimensions.

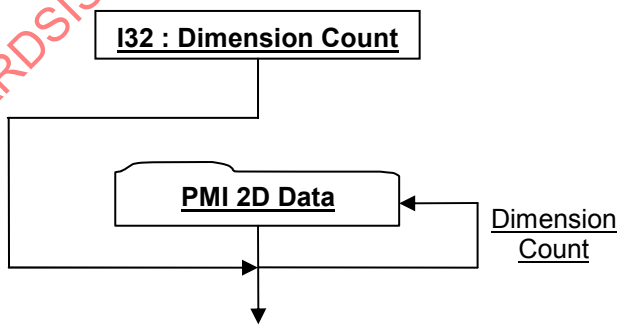


Figure 144 — PMI Dimension Entities data collection

I32 : Dimension Count

Dimension Count specifies the number of Dimension entities.

10.2.1.1.1 PMI 2D Data

The PMI 2D Data collection defines a data format common to all 2D based PMI entities.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

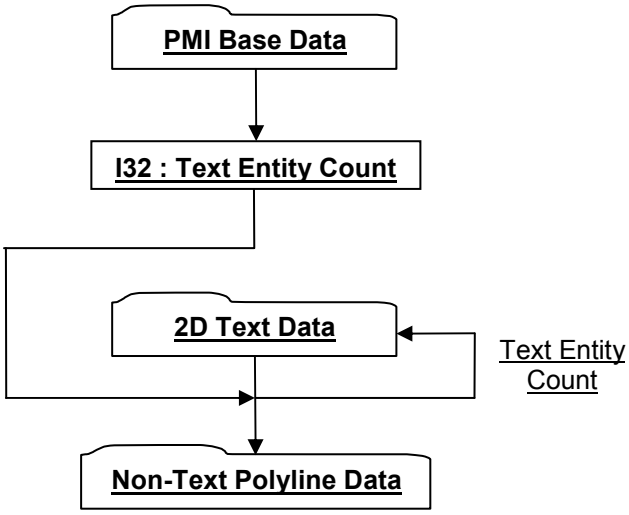


Figure 145 — PMI 2D Data collection

I32 : Text Entity Count

Text Entity Count specifies the number of Text entities in the particular PMI entity.

PMI Base Data

The PMI Base Data collection defines the basic/common data that every 2D and 3D PMI entity contains.

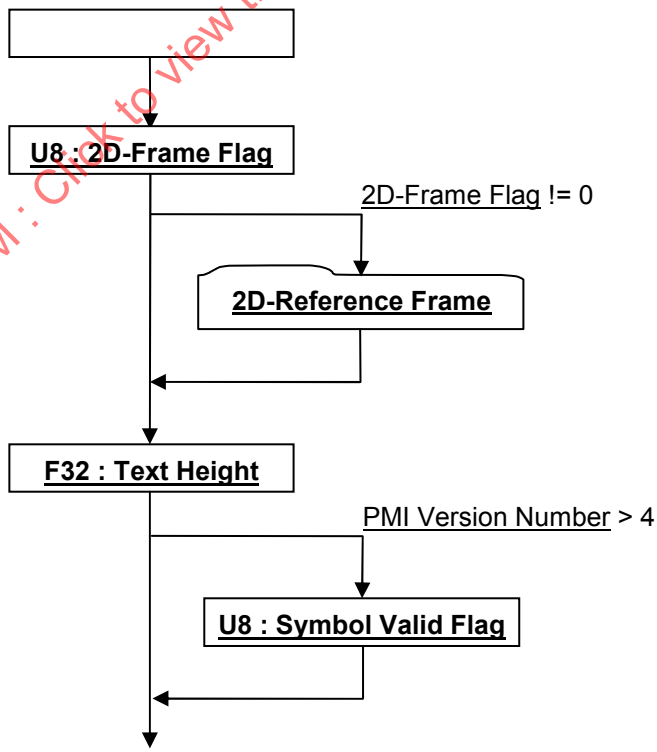


Figure 146 — PMI Base Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I32 : User Label

User Label specifies the particular PMI entity identifier.

U8 : 2D-Frame Flag

2D-Frame Flag is a flag specifying whether 2D-Reference Frame data is stored. If 2D-Frame Flag has a non-zero value then 2D-Reference Frame data is included. If 2D-Frame Flag has a value of "2", then dummy (i.e. all zeros) 2D-Reference Frame data is written. The "2D-Frame Flag = = 2" case is used by 10.2.6 Generic PMI Entities because for Generic PMI Entities all the Non-Text Polyline Data is already in 3D form (i.e. XYZ coordinate data).

F32 : Text Height

Text Height specifies the PMI text height in WCS.

U8 : Symbol Valid Flag

Symbol Valid Flag is a flag specifying whether the particular PMI entity is valid. If Symbol Valid Flag has a non-zero value then PMI entity is valid. This flag is only stored if the Version Number as defined in 10.2 PMI Manager Meta Data Element is greater than "4."

2D-Reference Frame

The 2D-Reference Frame data collection defines a reference frame (2D coordinate system) where the PMI entity is displayed in 3D space. All the PMI entity's 2D and 3D polyline data is assumed to lie on the defined plane.

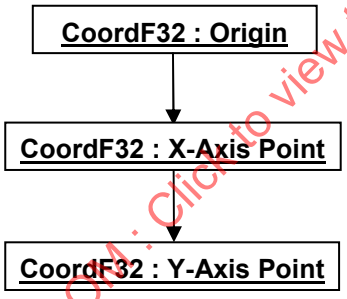


Figure 147 — 2D-Reference Frame data collection

CoordF32 : Origin

Origin defines the origin (min-corner) of the 2D coordinate system

CoordF32 : X-Axis Point

X-Axis Point defines a point along the X-Axis of the 2D coordinate system.

CoordF32 : Y-Axis Point

Y-Axis Point defines a point along the Y-Axis of the 2D coordinate system.

2D Text Data

The 2D Text Data collection defines a 2D text entity/primitive.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

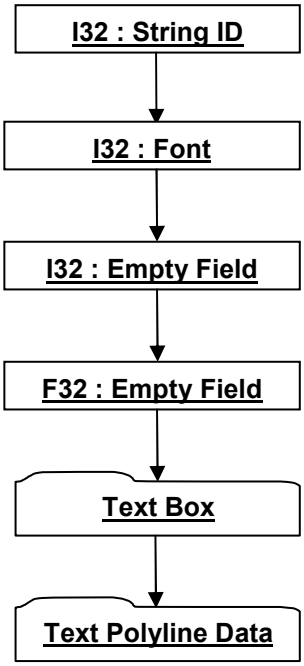


Figure 148 — 2D Text Data collection

I32 : String ID

String ID specifies the identifier for the character string. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

I32 : Font

Font identifies the font to be used for this text. Valid values include the following:

Table 74 — PMI 2D Base Data Font values

= 1	Simplex
= 2	Din
= 3	Military
= 4	ISO
= 5	Lightline
= 6	IGES 1001
= 7	Century
= 8	IGES 1002
= 9	IGES 1003
= 101	Japanese JISX 0208 coded character set
= 102	Japanese Extended Unix Codes JISX 0208 coded character set
= 103	Chinese GB 2312.1980 Simplified coded character set
= 104	Korean KSC 5601 coded character set
= 105	Chinese Big5 Traditional coded character set

I32 : Empty Field

Refer to best practice 14.4 Empty Field

F32 : Empty Field

Refer to best practice 14.4 Empty Field

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text Box

The Text Box data collection specifies a 2D box that particular text fits within. All values are with respect to 2D-Reference Frame documented in 0 2D-Reference Frame.

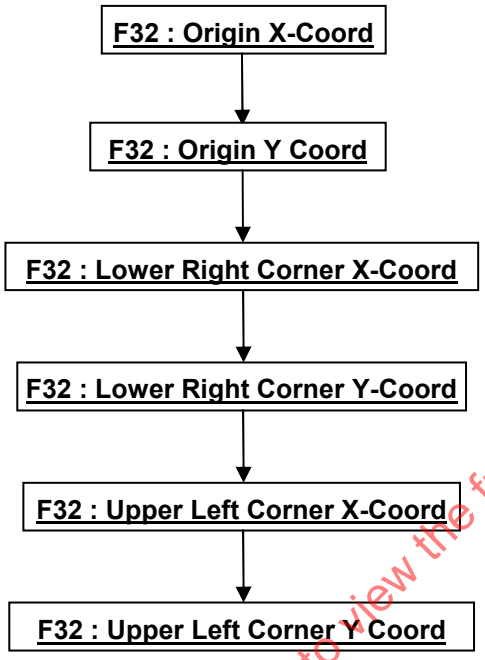


Figure 149 — Text Box data collection

F32 : Origin X-Coord

Origin X-Coord defines the 2D X-coordinate of the text origin with respect to 2D-Reference Frame.

F32 : Origin Y Coord

Origin Y-Coord defines the 2D Y-coordinate of the text origin with respect to 2D-Reference Frame.

F32 : Lower Right Corner X-Coord

Lower Right Corner X-Coord defines the 2D X-coordinate of the lower right corner of the text with respect to 2D-Reference Frame.

F32 : Lower Right Corner Y-Coord

Lower Right Corner Y-Coord defines the 2D Y-coordinate of the lower right corner of the text with respect to 2D-Reference Frame.

F32 : Upper Left Corner X-Coord

Upper Left Corner X-Coord defines the 2D X-coordinate of the upper left corner of the text with respect to 2D-Reference Frame.

F32 : Upper Left Corner Y Coord

Upper Left Corner Y-Coord defines the 2D Y-coordinate of the upper left corner of the text with respect to 2D-Reference Frame.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text Polyline Data

The Text Polyline Data collection defines any polyline segments which are part of the text representation. This existence of this polyline data is conditional (i.e. not all text has it) and is made up of an array of indices into an array of polyline segments packed as 2D vertex coordinates, specifying where each polyline segment begins and ends. Polylines are constructed from these arrays of data as follows:

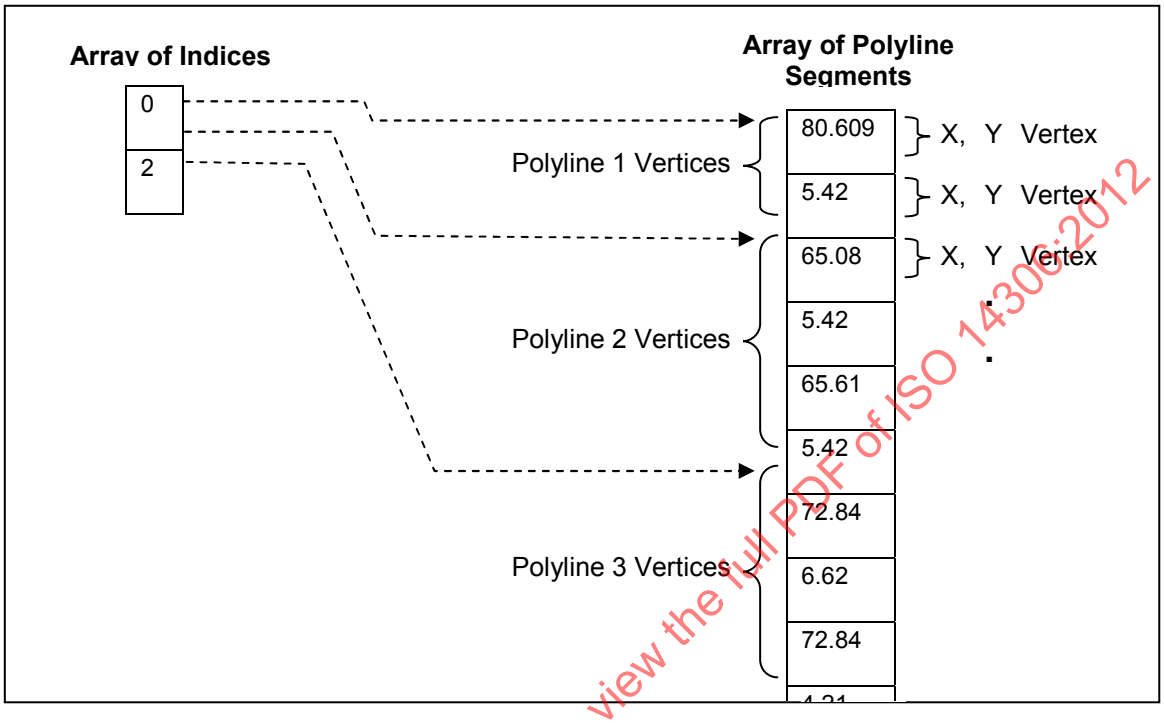


Figure 150 — Constructing Text Polylines from data arrays

This data is represented in JT file in the following format:

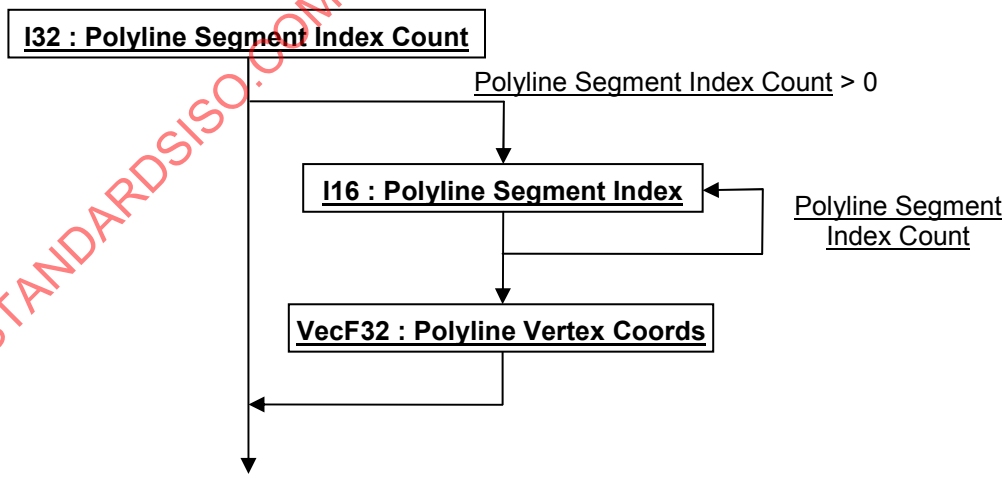


Figure 151 — Text Polyline Data collection

I32 : Polyline Segment Index Count

Polyline Segment Index Count specifies the number of polyline segment indices.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I16 : Polyline Segment Index

Polyline Segment Index is an index into the Polyline Vertex Coords array specifying where polyline segment begins or ends. This index is a vertex coordinate index so the absolute index into the Polyline Vertex Coords array is computed by multiplying the index value by "2" (i.e. for 2D coordinates).

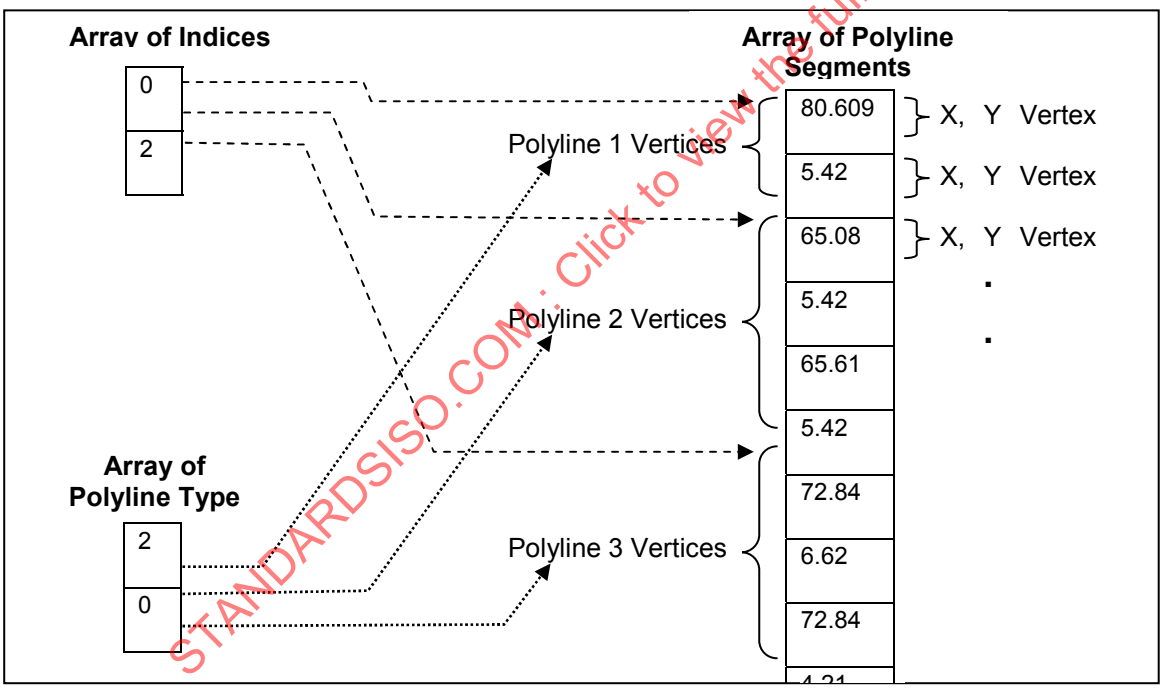
VecF32 : Polyline Vertex Coords

Polyline Vertex Coords is an array of polyline segments packed as 2D point coordinates. These 2D point coordinates are with respect to the 2D-Reference Frame documented in 0 2D-Reference Frame.

Non-Text Polyline Data

The Non-Text Polyline Data collection contains all the non-text polylines making up the particular PMI entity. Examples of non-text polylines include line attachments, text boxes, symbol box dividers, etc. The Non-Text Polyline Data collection is made up of an array of indices into an array of polyline segments packed as either 2D or 3D vertex coordinates, specifying where each polyline segment begins and ends. Whether vertex coordinates are 2D or 3D is dependent upon the PMI entity type using this data collection. If it is a 10.2.6 Generic PMI Entities type then the packed coordinate data is 3D; for all other PMI entity types the packed coordinate data is 2D. Also for Version Number, as defined in 10.2 PMI Manager Meta Data Element, greater than "4" an array of values that sequentially specify the polyline type in the polyline segments array is included.

Figure 152 below shows how Polylines are constructed from these arrays of data for the packed 2D coordinates case. The packed 3D coordinates case is interpreted the same except that the coordinates array includes a Z component and is thus packed as "[XYZ][XYZ][XYZ]…"



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 152 — Constructing Non-Text Polylines from packed 2D data arrays

This data is represented in the JT format as follows:

© ISO 2012 – All rights reserved

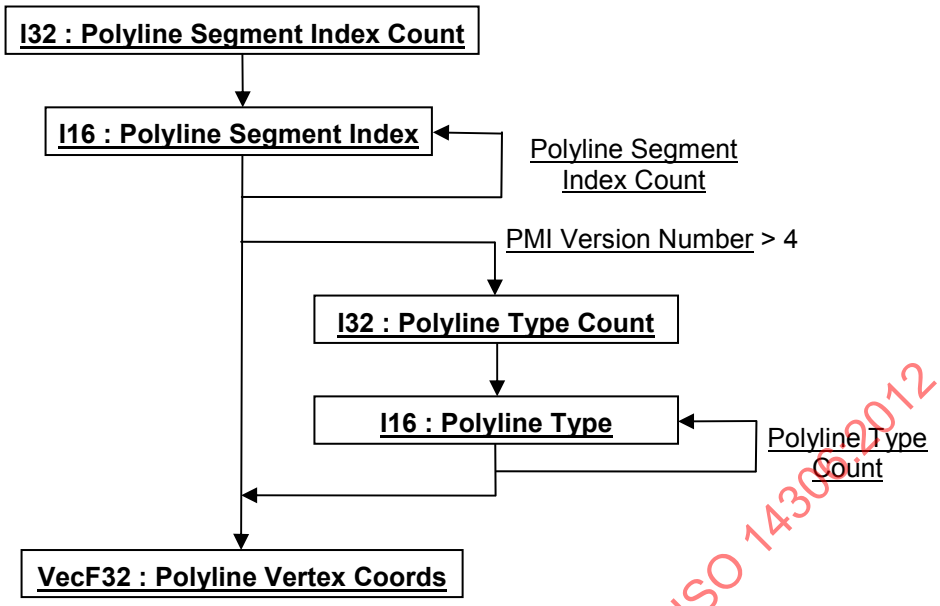


Figure 153 — Non-Text Polyline Data collection

I32 : Polyline Segment Index Count

Polyline Segment Index Count specifies the number of polyline segment indices.

I16 : Polyline Segment Index

Polyline Segment Index is an index into the Polyline Vertex Coords array specifying where polyline segment begins or ends. This index is a vertex/coordinate index so the absolute index into the Polyline Vertex Coords array is computed by multiplying the index value by "2" (i.e. for 2D coordinates).

I32 : Polyline Type Count

Polyline Type Count specifies the number of polyline type values.

I16 : Polyline Type

Polyline Type specifies the type of polyline segment in Polyline Vertex Coords array. See Figure 152 — Constructing Non-Text Polylines from packed 2D data arrays for interpretation of this array of type values relative to the defined polylines. Valid values include the following:

Table 75 — PMI 2D Non-Text Polyline Type values

= 0	General line
= 1	General arrow
= 2	General circle
= 3	General arc
= 4	Extended line 1
= 5	Extended line 2
= 6	Extended arc
= 7	Extended circle
= 8	Text line (used in text boxes and symbol box dividers)
= 9	Text string

VecF32 : Polyline Vertex Coords

Polyline Vertex Coords is an array of polyline segments packed as 2D point coordinates. These 2D point coordinates are with respect to the 2D-Reference Frame documented in 0 2D-Reference Frame.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

10.2.1.2 PMI Note Entities

The PMI Note Entities data collection defines data for a list of Notes. Notes are used to connect textual information to specific Part entities.

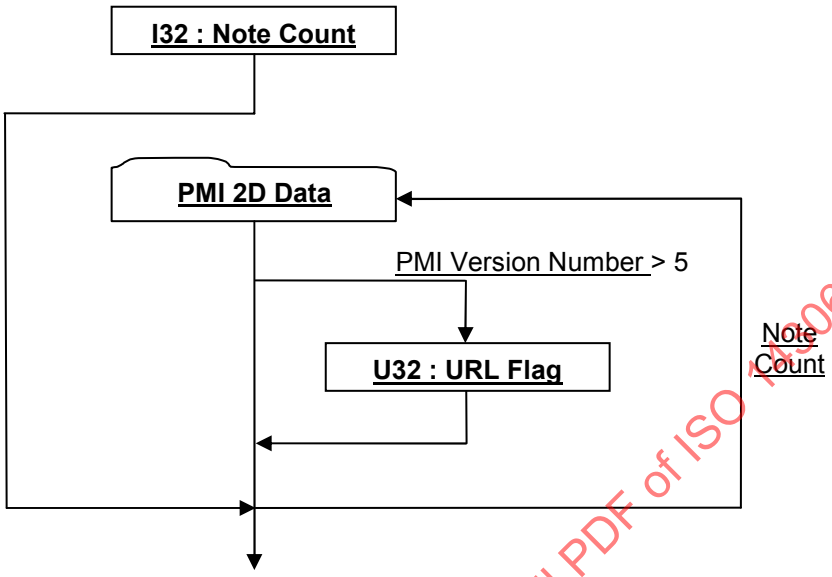


Figure 154 — PMI Note Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : Note Count

Note Count specifies the number of Note entities.

U32 : URL Flag

URL Flag specifies whether Note is an URL. This data field is only present if Version Number, as defined in 10.2 PMI Manager Meta Data Element, is greater than "5". The URL is the actual text of the note as specified in PMI 2D Data.

10.2.1.3 PMI Datum Feature Symbol Entities

The PMI Datum Feature Symbol Entities data collection defines data for a list of Datum Feature Symbols. A Datum Feature Symbol is a Geometric Dimensioning and Tolerancing (GD&T) symbol that provides a "label" for a part feature which is referenced by a Feature Control Frame.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

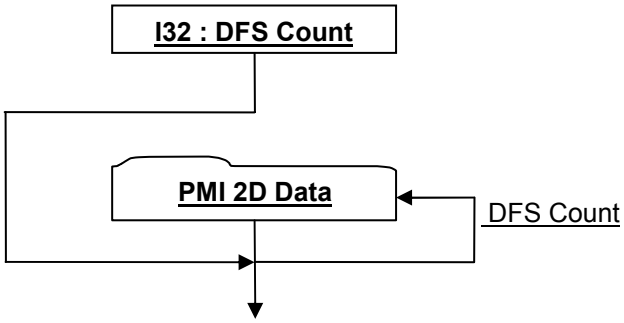


Figure 155 — PMI Datum Feature Symbol Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : DFS Count

DFS Count specifies the number of Datum Feature Symbol entities.

10.2.1.4 PMI Datum Target Entities

The PMI Datum Target Entities data collection defines data for a list of Datum Targets. A Datum Target is a Geometric Dimensioning and Tolerancing (GD&T) symbol that specifies a point, a line, or an area on a part to define a “datum” for manufacturing and inspection operations.

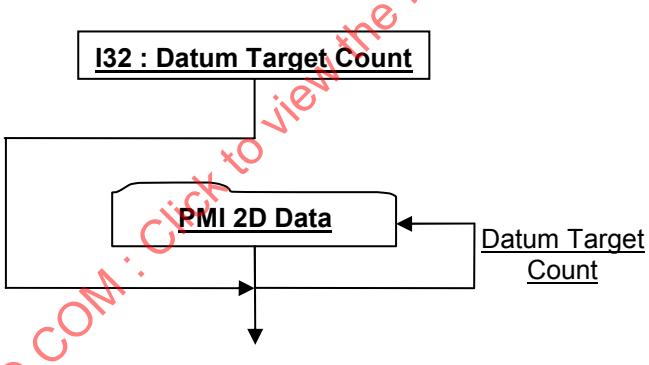


Figure 156 — PMI Datum Target Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : Datum Target Count

Datum Target Count specifies the number of Datum Target entities.

10.2.1.5 PMI Feature Control Frame Entities

The PMI Feature Control Frame Entities data collection defines data for a list of Feature Control Frames. A Feature Control Frame is a Geometric Dimensioning and Tolerancing (GD&T) symbol used for expressing the geometric characteristics, form tolerance, runout or location tolerance, and relationships between the geometric features of a part. If necessary, Datum Feature and/or Datum Target references may be included in the Feature Control Frame symbol.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

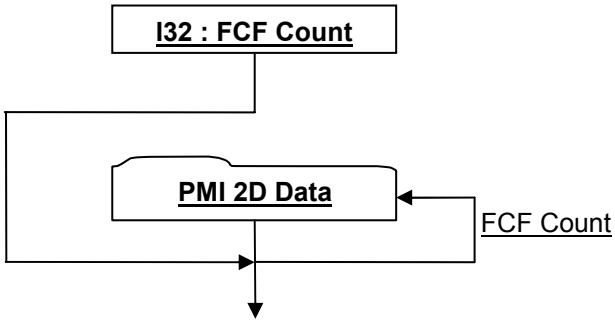


Figure 157 — PMI Feature Control Frame Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : FCF Count

FCF Count specifies the number of Feature Control Frame entities.

10.2.1.6 PMI Line Weld Entities

The PMI Line Weld Entities data collection defines data for a list of Line Weld symbols. A Line Weld symbol describes the specifications for welding a joint.

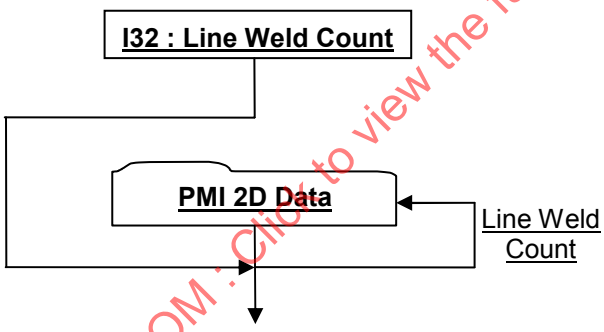


Figure 158 — PMI Line Weld Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : Line Weld Count

Line Weld Count specifies the number of Line Weld entities.

10.2.1.7 PMI Spot Weld Entities

The PMI Spot Weld Entities data collection defines data for a list of Spot Weld Symbols. Spot Weld symbols describe the specifications for welding sheet metal.

Several data fields of the PMI Spot Weld Entities data collection are only present if Version Number, as defined in 10.2 PMI Manager Meta Data Element, is greater than or equal to “4”.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

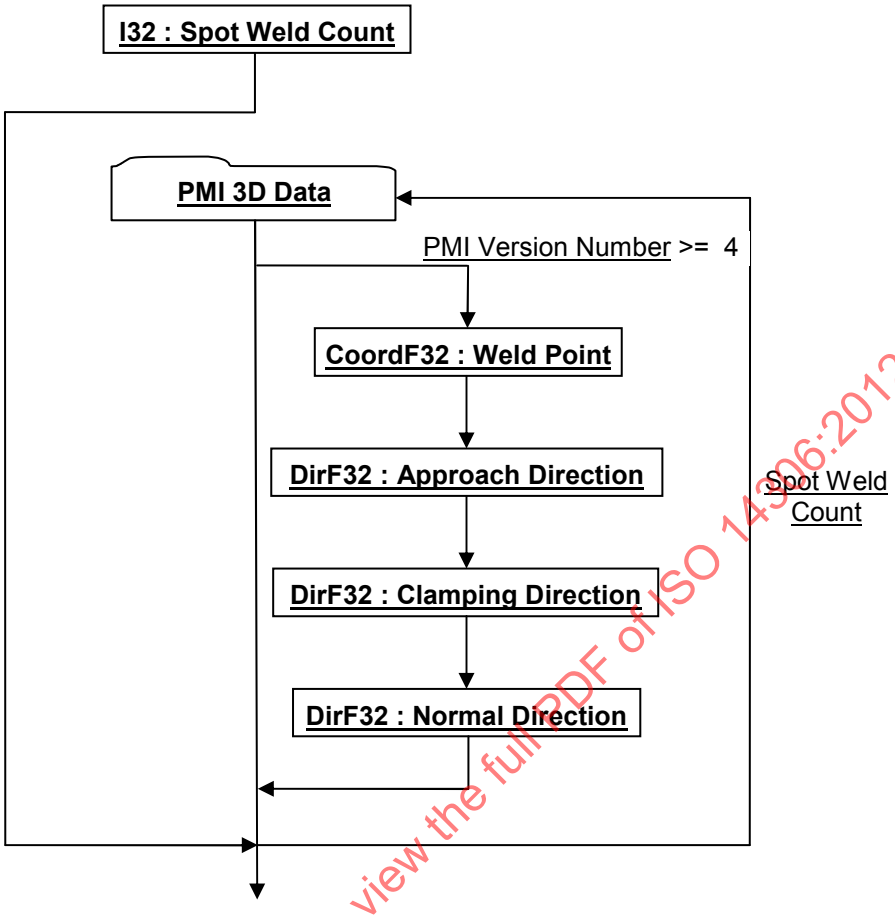


Figure 159 — PMI Spot Weld Entities data collection

I32 : Spot Weld Count

Spot Weld Count specifies the number of Spot Weld entities.

CoordF32 : Weld Point

Weld Point specifies the coordinates of the weld point.

DirF32 : Approach Direction

Approach Direction specifies the components of the direction vector from which the weld gun approaches the part.

DirF32 : Clamping Direction

Clamping Direction specifies the components of the clamping force direction vector.

DirF32 : Normal Direction

Normal Direction specifies the components of the direction vector normal to the actual spot weld.

10.2.1.7.1 PMI 3D Data

The PMI 3D Data collection defines a data format common to all 3D based PMI entities.

Along with the PMI Base Data and String identifier, this data collection also includes non-text polyline data defined by an array of indices into an array of polyline segments packed as 2D/3D vertex coordinates,

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

specifying where each polyline segment begins and ends. How polylines are constructed from this index array and packed vertex coordinates array is the same as that illustrated in Figure 150 of 0 Text Polyline Data.

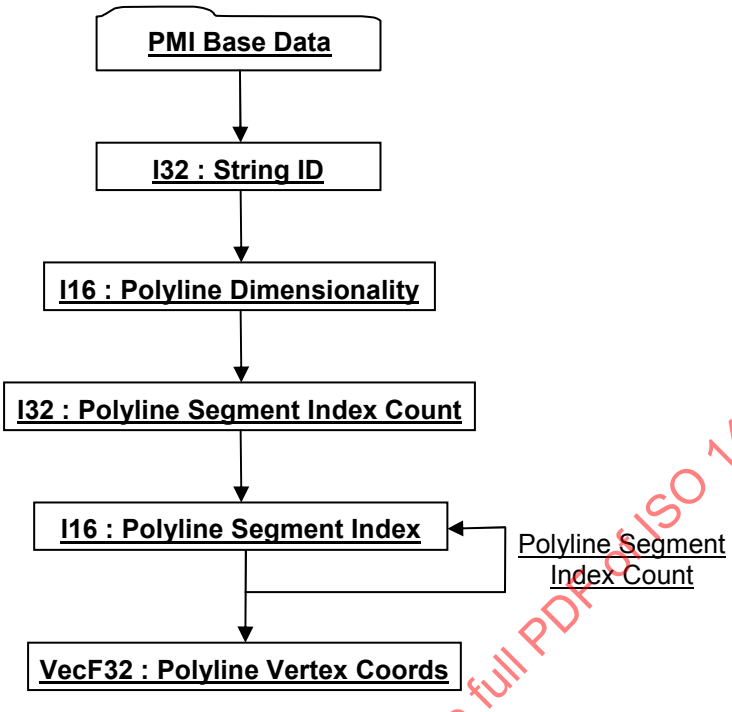


Figure 160 — PMI 3D Data collection

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

I32 : String ID

String ID specifies the identifier for the character string. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

I16 : Polyline Dimensionality

Polyline Dimensionality specifies the dimensionality of the polyline coordinates packed in Polyline Vertex Coords. Valid values include the following:

Table 76 — PMI 3D Data Polyline Dimensionality values

= 2	Indicates 2-dimensioanl (xyxy…) data packing..
= 3	Indicates 3-dimensional (xyzxyz…) data packing.

I32 : Polyline Segment Index Count

Polyline Segment Index Count specifies the number of polyline segment indices.

I16 : Polyline Segment Index

Polyline Segment Index is an index into the Polyline Vertex Coords array specifying where polyline segment begins or ends. This index is a vertex coordinate index so the absolute index into the Polyline Vertex Coords array is computed by multiplying the index value by Polyline Dimensionality.

VecF32 : Polyline Vertex Coords

Polyline Vertex Coords is an array of polyline segments packed as Polyline Dimensionality point coordinates.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

10.2.1.8 PMI Surface Finish Entities

The PMI Surface Finish Entities data collection defines data for a list of Surface Finish symbols. Surface Finish symbols indicate surface quality and generally are only specified where finish quality affects function (e.g. bearings, pistons, gears).

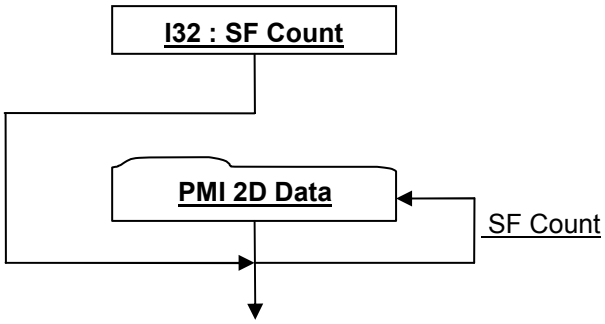


Figure 161 — PMI Surface Finish Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : SF Count

SF Count specifies the number of Surface Finish symbol entities.

10.2.1.9 PMI Measurement Point Entities

The PMI Measurement Point Entities data collection defines data for a list of Measurement Point symbols. Measurement Points are predefined locations (i.e. geometric entities or theoretical, but measurable points, such as surface locations) which are measured on manufactured parts to verify the accuracy of the manufacturing process.

Several data fields of the PMI Measurement Point Entities data collection are only present if Version Number, as defined in 10.2 PMI Manager Meta Data Element, is greater than or equal to “4”.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

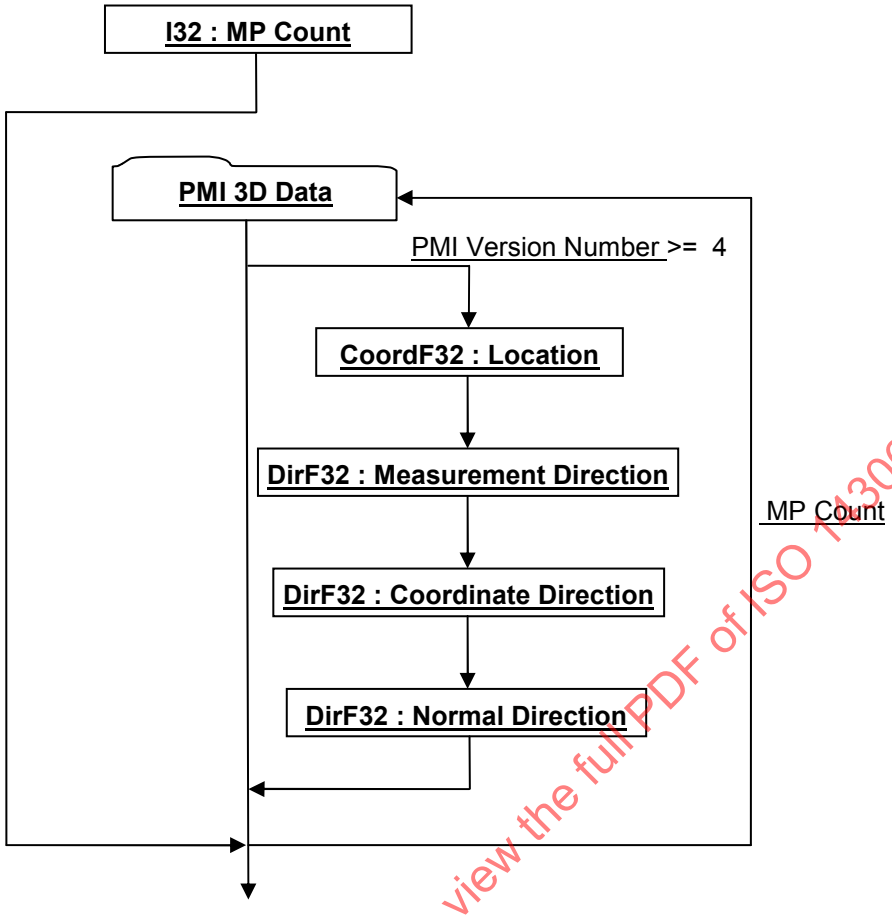


Figure 162 — PMI Measurement Point Entities data collection

Complete description for PMI 3D Data can be found in 10.2.1.7.1 PMI 3D Data.

I32 : MP Count

MP Count specifies the number of Measurement Point entities.

CoordF32 : Location

Location specifies the coordinates of the Measurement Point.

DirF32 : Measurement Direction

Measurement Direction specifies the components of the direction vector from which a CCM (Coordinate Measuring Machine) approaches when taking a measurement.

DirF32 : Coordinate Direction

Coordinate Direction specifies the components of the direction vector another Measurement Point on a mating part would like to align with a Measurement Point on the first part.

DirF32 : Normal Direction

Normal Direction specifies the components of the direction vector normal to the actual Measurement Point.

10.2.1.10 PMI Locator Entities

The PMI Locator Entities data collection defines data for a list of Locator symbols. Locator symbols are used to accurately locate components with respect to each other and the manufacturing tooling.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

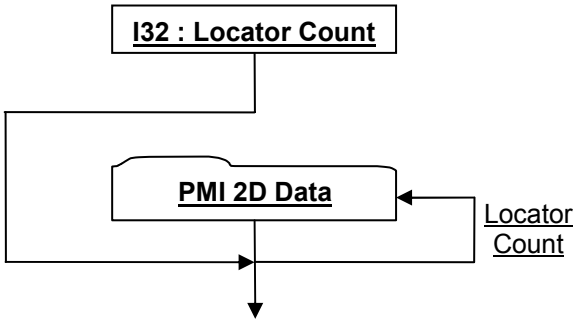


Figure 163 — PMI Locator Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : Locator Count

Locator Count specifies the number of Locator symbol entities.

10.2.1.11 PMI Reference Geometry Entities

The PMI Reference Geometry Entities data collection defines data for a list of Reference Geometry. Reference Geometry can be thought of as user-definable datums, which are positioned relative to the topology of an existing entity. Each reference geometry type (point, polyline, polygon) can be implicitly determined by the value of Polyline Segment Index[1] (see 10.2.1.7.1 PMI 3D Data) as follows:

Table 77 — PMI Reference Geometry Entity values

Polyline Segment Index[1]	Implied Reference Geometry Type
= 1	Point
= 2	Polyline
> 2	Polygon

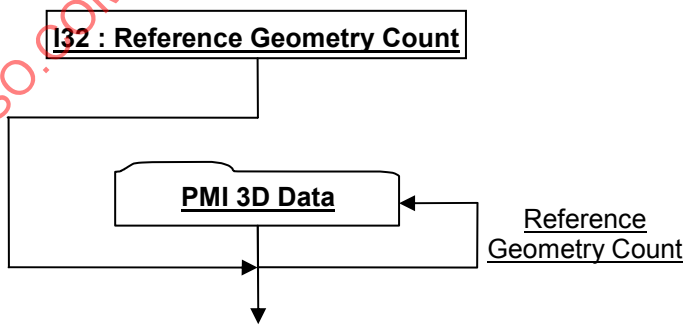


Figure 164 — PMI Reference Geometry Entities data collection

Complete description for PMI 3D Data can be found in 10.2.1.7.1 PMI 3D Data.

I32 : Reference Geometry Count

Reference Geometry Count specifies the number of Reference Geometry entities.

10.2.1.12 PMI Design Group Entities

The PMI Design Group Entities data collection defines data for a list of Design Groups. Design Groups are collections of PMI created to organize a model into smaller subsets of information. This organization is

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

achieved via PMI Associations (see 10.2.2 PMI Associations), where specific PMI entities are associated as "destinations" to a "source" PMI Design Group.

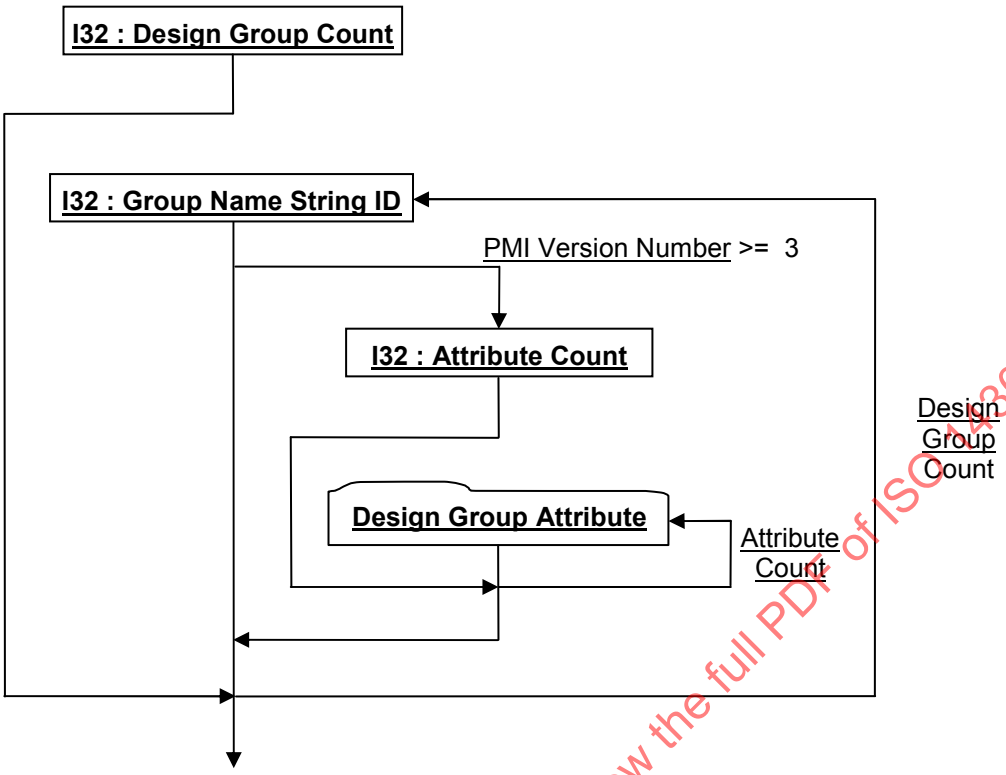


Figure 165 — PMI Design Group Entities data collection

I32 : Design Group Count

Design Group Count specifies the number of Design Group entities.

I32 : Group Name String ID

Group Name String ID specifies the identifier for the group name character string. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

I32 : Attribute Count

Attribute Count specifies the number of Design Group Attribute data collections

10.2.1.12.1 Design Group Attribute

The Design Group Attribute data collection defines a group property/attribute.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

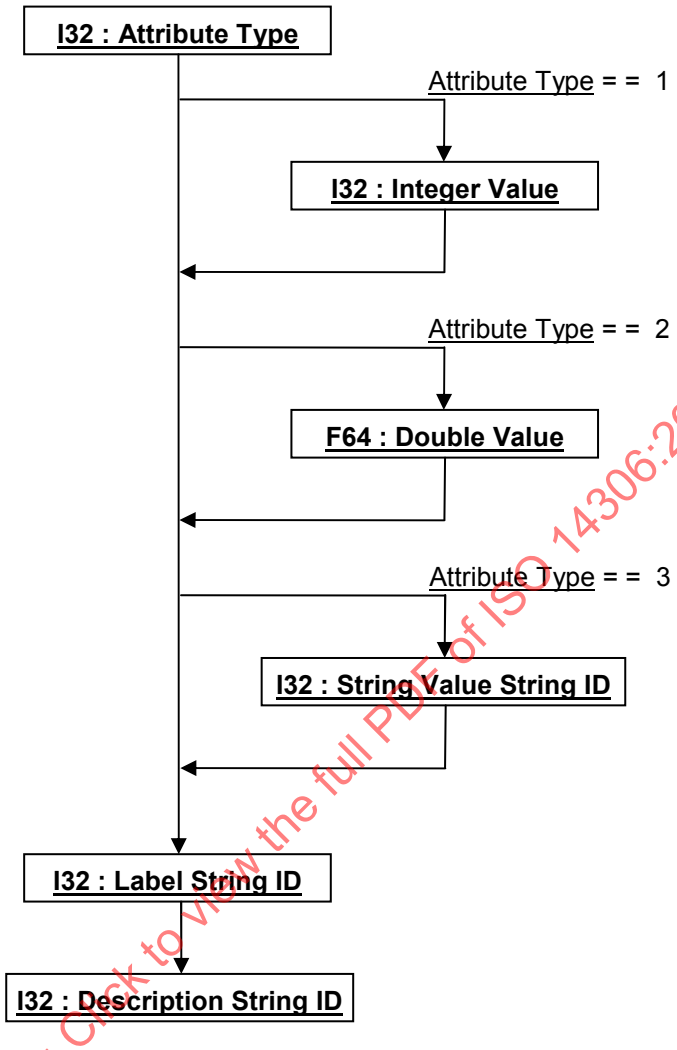


Figure 166 — Design Group Attribute data collection

I32 : Attribute Type

Attribute Type specifies the attribute type. Valid types include the following:

Table 78 — PMI Design Group Attribute Type values

= 1	Integer
= 2	Double
= 3	String

I32 : Integer Value

Integer Value specifies the value for “integer” Attribute Types.

F64 : Double Value

Double Value specifies the value for “double” Attribute Types.

I32 : String Value String ID

String Value String ID specifies the string identifier value for “string” Attribute Types. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of “-1” indicates no string.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I32 : Label String ID

Label String ID specifies the string identifier for the attribute label. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

I32 : Description String ID

Description String ID specifies the string identifier for the attribute description. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

10.2.1.13 PMI Coordinate System Entities

The PMI Coordinate System Entities data collection defines data for a list of Coordinate Systems.

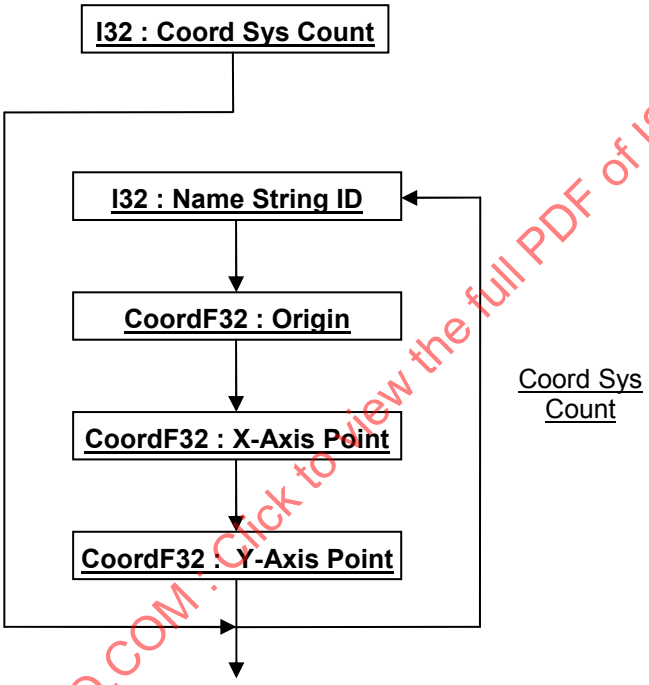


Figure 167 — PMI Coordinate System Entities data collection

I32 : Coord Sys Count

Coord Sys Count specifies the number of Coordinate System entities.

I32 : Name String ID

Name String ID specifies the string identifier for the Coordinate System name. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

CoordF32 : Origin

Origin defines the origin of the coordinate system.

CoordF32 : X-Axis Point

X-Axis Point defines a point along the X-Axis of the coordinate system.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CoordF32 : Y-Axis Point

Y-Axis Point defines a point along the Y-Axis of the coordinate system.

10.2.2 PMI Associations

The PMI Associations data collection defines data for a list of associations. An association defines a link ("relationship") between two PMI, B-Rep, or Wireframe Rep entities where one entity is defined as the "source" and the other entity is defined as the "destination".

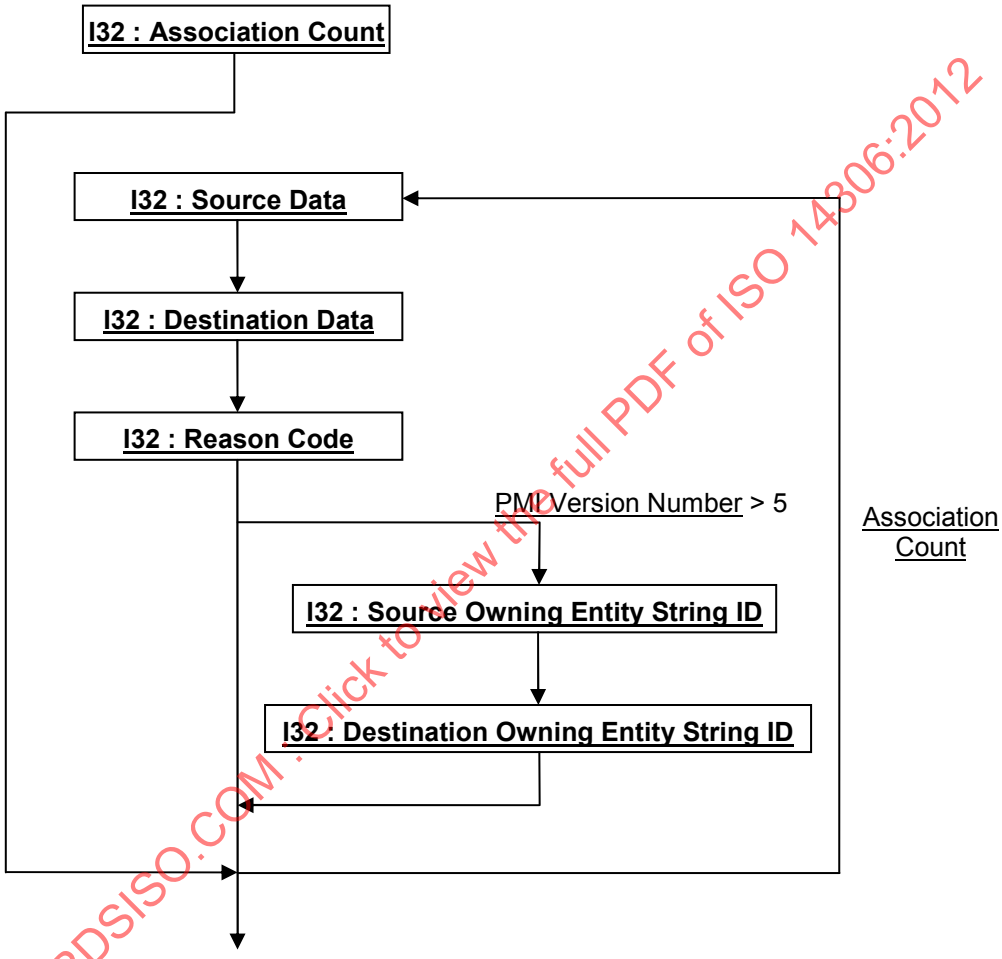


Figure 168 — PMI Associations data collection

I32 : Association Count

Association Count specifies the number of associations.

I32 : Source Data

Source Data is a collection of source entity information encoded/packed within a single I32 using the following bit allocation. All bits fields that are not defined as in use should be set to "0".

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 79 — PMI Associations Source Data values

Bits 0 - 23	Source Entity Identifier. The interpretation of this identifier data is dependent upon the value of Bit 31 documented below.
Bits 24 -30	Source Entity PMI or B-Rep type. Valid types include the following: = 0 PMI - Dimension = 1 PMI - Note = 2 PMI - Datum Feature Symbol = 3 PMI - Datum Target = 4 PMI - Feature Control Frame = 5 PMI - Line Weld = 6 PMI - Spot Weld = 7 PMI - Measurement Point = 8 PMI - Surface Finish = 9 PMI - Locator Designator = 10 PMI - Reference Geometry = 11 PMI - Coordinate System = 12 PMI - Design Group = 13 PMI - User Attribute = 14 B-Rep - Vertex = 15 B-Rep - Edge = 16 B-Rep - Face = 17 PMI - Model View = 18 PMI - Generic = 19 Wireframe Rep - Edge = 20 PMI - Unspecified type = 21 Part Instance
Bit 31	Indirect Identifier Flag = 0 – Value in Bits 0-23 is not the actual CAD identifier, instead Bits 0-23 is an index into the source type's PMI array or index of the edge/face in B-Rep or Wireframe Rep for the source entity. = 1 – Value in Bits 0-23 is not the actual CAD identifier; instead Bits 0-23 is an index into the list of CAD Tags (as documented in 10.2.7 PMI CAD Tag Data) identifying the CAD Tag belonging to the particular source entity.

I32 : Destination Data

Destination Data is a collection of destination entity information encoded/packed within a single I32. The encoding schema and interpretation of this data is the same as that documented in Source Data.

I32 : Reason Code

Reason Code specifies the "reason" for the association. Valid Reason Codes include the following:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 80 — PMI Associations Reason Code values

= 0	Association is to the primary entity being dimensioned
= 1	Association is to the secondary entity being dimensioned
= 2	Association is to the dimension plane
= 5	Association is to the entity used to specify the Z-Axis of a coordinate system
= 10	Association is to an entity "associated" to or "included in" a PMI symbol
= 11	Association is to an entity used to "attach" a PMI symbol.
= 12	Association is to first entity used to “attach” a PMI symbol
= 13	Association is to second entity used to “attach” a PMI symbol
= 14	Specifying PMI grouping, source is PMI/B-Rep entity and destination is design group.
= 15	Association is to a weld line entity
= 16	Association is to a “hot spot”
= 17	Association is to a child in a PMI stack
= 72	Association is for PMI miscellaneous relation.
= 73	Association is for PMI related entity.
= 98	Association is to show the PMI when associated Model View is selected. Source is the PMI, and destination is Model View.
= 99	Association is to show/select PMI B, if showing/selecting PMI A. Source is PMI A, and destination is PMI B. This is different from an “attached” PMI , where the convention is to show the PMI visibly linked to one another.
= 100	Association is to show all parts except the associated part instance. Source is the part instance, and destination is Model View

I32 : Source Owning Entity String ID

Source Owning Entity String ID specifies the string identifier for the string which contains the unique CAD identifier of the component (part or assembly) that owns the source PMI or B-Rep entity. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of “-1” indicates no string and implies that the entity is to be found on the current node‘s PMI/B-Rep/Wireframe-Rep segment. It is valid for the source owning entity to be the same as the destination owning entity (i.e. an association between two PMI or B-Rep entities in the same part/assembly). This data field is only present if Version Number, as defined in 10.2 PMI Manager Meta Data Element, is greater than “5”.

I32 : Destination Owning Entity String ID

Destination Owning Entity String ID specifies the string identifier for the string which contains the unique CAD identifier of the component (part or assembly) that owns the destination PMI or B-Rep entity. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of “-1” indicates no string and implies that the entity is to be found on the current node‘s PMI/B-Rep/Wireframe-Rep segment. It is valid for the source owning entity to be the same as the destination owning entity (i.e. an association between two PMI or B-Rep entities in the same part/assembly). This data field is only present if Version Number, as defined in 10.2 PMI Manager Meta Data Element, is greater than “5”.

10.2.3 PMI User Attributes

The PMI User Attributes collection defines data for a list of user attributes. PMI User Attributes are used to add attribute data to a part/assembly. Each user attribute is composed of key/value pair of strings.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

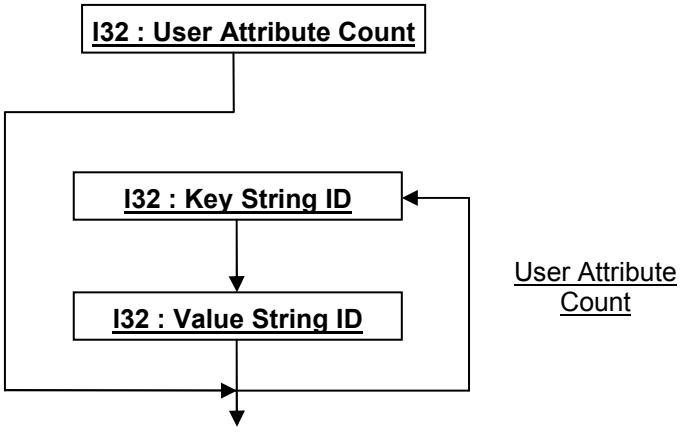


Figure 169 — PMI User Attributes data collection

I32 : User Attribute Count

User Attribute Count specifies the number of user attributes.

I32 : Key String ID

Key String ID specifies the string identifier for the user attribute key. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

I32 : Value String ID

Value String ID specifies the string identifier for the user attribute value. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

10.2.4 PMI String Table

The PMI String Table data collection defines data for a list of character strings and serves as a central repository for all character strings used by other PMI Entities within the same PMI Manager Meta Data Element. PMI Entities reference into this list/array of character strings to define usage of a particular character string using a simple list/array "index" (i.e. String ID).

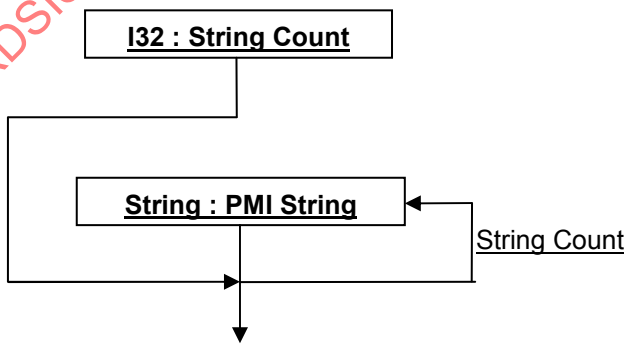


Figure 170 — PMI String Table data collection

I32 : String Count

String Count specifies the number of character strings in the string table.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

String : PMI String

PMI String specifies the character string.

10.2.5 PMI Model Views

The PMI Model Views data collection defines data for a list of Model Views. A fully annotated part/assembly may contain so much PMI information, that it becomes very difficult to interpret the design intent when viewing a 3D Model (with PMI visible) of the part/assembly. Model Views provide a means to capture and organize PMI information about a 3D model so that the design intent can be clearly interpreted and communicated to others in later stages of the Product Lifecycle Management (PLM) process (e.g. manufacturing, inspection, assembly). This organization is achieved via PMI Associations (see 10.2.2 PMI Associations), where specific PMI entities are associated as "destinations" to a "source" PMI Model View.

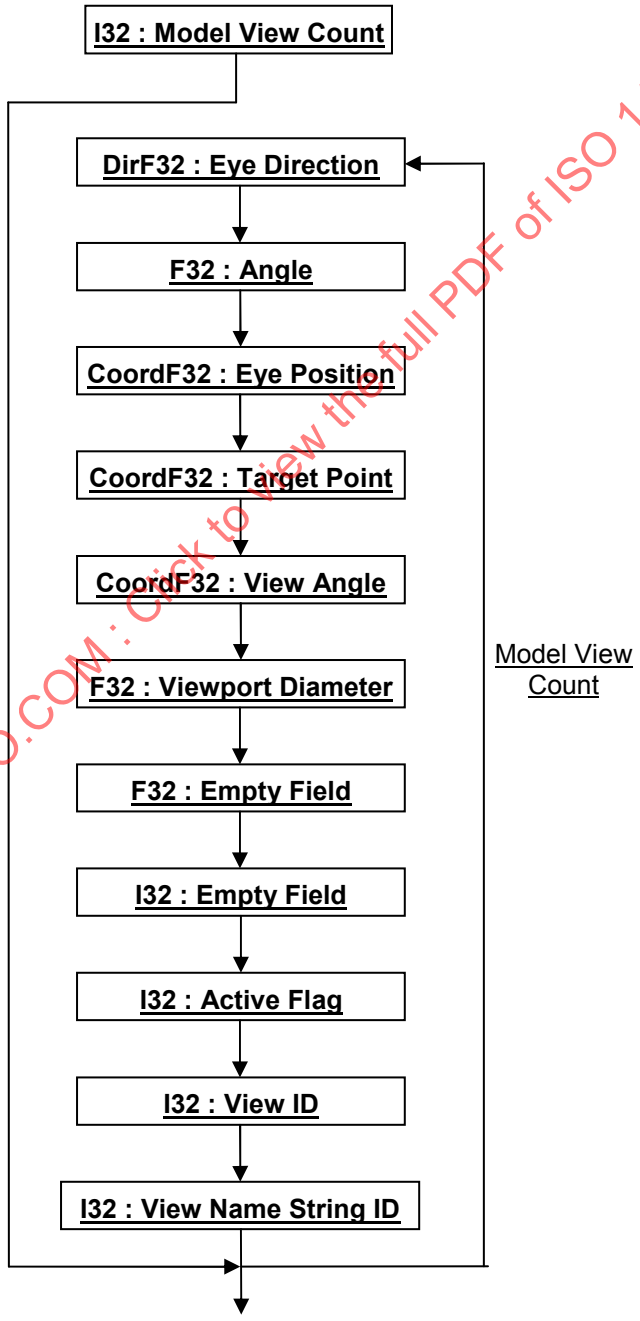


Figure 171 — PMI Model Views data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

I32 : Model View Count

Model View Count specifies the number of Model Views.

DirF32 : Eye Direction

Eye Direction specifies the camera direction vector.

F32 : Angle

Angle specifies the camera rotation angle (in degrees where positive is counter-clockwise) about the Eye Direction. So this Angle in combination with the Eye Direction is equivalent to specifying a rotation using axis-angle representation.

CoordF32 : Eye Position

Eye Position specifies the WCS coordinates of the eye/camera “look from” position.

CoordF32 : Target Point

Target Point specifies the WCS coordinates of the eye/camera “look at” position.

CoordF32 : View Angle

View angle specifies the X, Y, Z rotation angles (in degrees) of the model’s axis. The rotations are defined with respect to an initial orientation where the model’s axis are aligned with the screen’s axis (i.e. +X axis points to right, +Y axis points up, +Z axis points out at you).

F32 : Viewport Diameter

Viewport Diameter specifies the diameter in WCS coordinates of the largest possible circle that could be inscribed within viewport. If a large diameter value is specified, the model appears very small in relation to the viewport; whereas if a small diameter value is specified a close-up (“zoomed-in)” view of the model results.

F32 : Empty Field

Refer to best practice 14.4 Empty Field

I32 : Empty Field

Refer to best practice 14.4 Empty Field

I32 : Active Flag

Active Flag is a flag specifying whether this Model View is the “active” view. Valid values include the following:

Table 81 — PMI Model Views Active Flag values

= 0	Is not the active Model View.
= 1	Is the active Model View

I32 : View ID

View ID specifies the Model View unique identifier.

I32 : View Name String ID

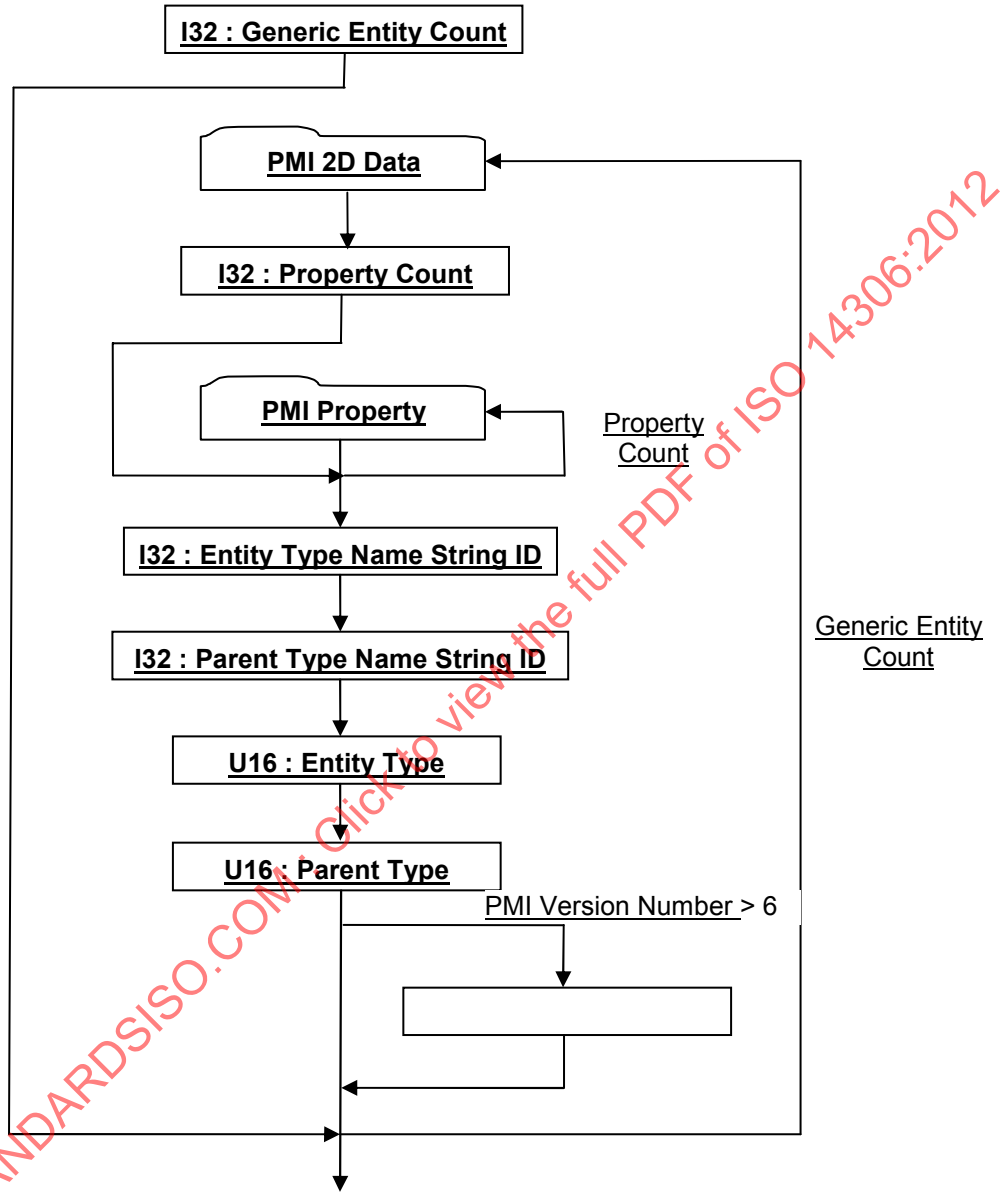
View Name String ID specifies the string identifier for the Model View’s name. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of “-1” indicates no string.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

10.2.6 Generic PMI Entities

The Generic PMI Entities data collection provides a “generic” format for defining various PMI entity types, including user defined types. The generic format defines the data making up the PMI Entity through a combination of the PMI 2D Data collection and a list of PMI Property data collections.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 172 — Generic PMI Entities data collection

Complete description for PMI 2D Data can be found in 10.2.1.1.1 PMI 2D Data.

I32 : Generic Entity Count

Generic Entity Count specifies the number of Generic PMI Entities.

I32 : Property Count

Property Count specifies the number of PMI Properties.

© ISO 2012 – All rights reserved

I32 : Entity Type Name String ID

Entity Type Name String ID specifies the string identifier for the name of the Generic PMI Entity Type. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

I32 : Parent Type Name String ID

Parent Type Name String ID specifies the string identifier for the name of the parent Generic PMI Entity Type. This identifier is an index to a particular character string in the PMI String Table as defined in 10.2.4 PMI String Table. An identifier value of "-1" indicates no string.

U16 : Entity Type

Entity Type specifies the Generic PMI Entity Type. The valid Entity Type values (in hexadecimal format) are documented in the following table. Note that for "user defined" Generic PMI Entities a hexadecimal value of "0x0114" (as documented in table below) should be used.

Table 82 — Generic PMI Entity Type values

0x0001	PMI (generally only used as a Parent Type)
0x0002	Weld
0x0004	Spot Weld
0x0008	Line Weld
0x0010	Groove Weld
0x0011	Fillet Weld
0x0012	Slot Weld
0x0014	Edge Weld
0x0018	Arc Spot Weld
0x0020	Resistance Spot Weld
0x0021	Resistance Seam Weld
0x0022	Structural Adhesive Bead Shaped
0x0024	Structural Adhesive Tape Shaped
0x0028	Structural Adhesive Dollop Shaped
0x0040	Mechanical Clinch Connector
0x0041	Surface Finish
0x0042	Measurement Point
0x0044	Datum Locator
0x0048	Certification Point
0x0080	Geometric Dimensioning and Tolerancing
0x0081	Feature Control Frame
0x0082	Dimension
0x0084	Datum Feature Symbol
0x0088	Datum Target
0x0100	Note
0x0101	Face Attribute Note
0x0102	Model View Label Note
0x0104	Coordinate System
0x0108	Reference Geometry
0x0110	Reference Point
0x0111	Reference Axis
0x0112	Reference Plane
0x0114	User Defined
0x0118	Measurement Locator
0x0120	Datum Point

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

0x0121	Surface Vector Measurement Point
0x0122	Hole Vector Measurement Point
0x0124	Trimmed Sheet Vector Measurement Point
0x0128	Hem Vector Measurement Point
0x0230	Fastener PMI
0x0231	Material specification
0x0232	Process specification
0x0233	Part specification
0x0235	Balloon Note
0x0238	Circle Centre
0x0239	Coordinate Note
0x0240	AttributeNote
0x0241	Bundle or Dressing Note
0x0242	Cutting Plane Symbol
0x0243	Crosshatch
0x0244	E Marking (Note)
0x0245	Organization
0x0246	Region
0x0305	Section
0x0306	Centreline
0x0307	Fit Designation
0x0308	Composite Feature Control Frame

U16 : Parent Type

Parent Type specifies the parent Generic PMI Entity Type. The valid Parent Type values are the same as that documented above for Entity Type. The Parent Type is used to create a class hierarchy of PMI when presenting the PMI contents from a JT file.

U16 : User Flags

User Flags is a collection of flags. The flags are combined using the binary OR operator and store various state information for the Generic PMI Entity. All bits fields that are not defined as in use should be set to "0".

Table 83 — Generic PMI User Flag values

0x0001	Show PMI Entity "flat to screen only" flag = 0 – Allow PMI display plane to rotate with model. = 1 – Display PMI entity in the plane of the screen, so that it does not rotate with model.

PMI Property

A PMI Property data collection consists of a key/value pair and is used to describe attributes of Generic PMI Entity or other specific data.

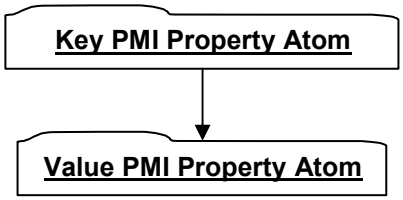


Figure 173 — PMI Property data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Both Key and Value have the same format as that documented in PMI Property Atom .

Although there is no reference compliant requirements for what the PMI Property key/value pairs shall be for each Generic PMI Entity type, there are some common PMI Property keys and corresponding value formats that appear in JT File. The below table documents these common PMI Property keys (i.e. the keys encoded string value) and what the format of the value data is in the values encoded string (see PMI Property Atom for an explanation of what is meant by "encoded string value" for the "key" and "value" data, see [3] for an explaination of the "%f" format).

Table 84 — Common Property Keys and Their Value Encoding formats

"Key" Property Atom Value String	"Value" Property Atom Value String Encoding Format	Decoding Notes
"PMI_PROP_ANCHOR_POINT"	"Px Py Pz"	Each Px, Py, Pz is a F32 value using "%f" format
"PMI_PROP_NOTE_HAS_URL"	"0" or "1"	0 = = False; 1 = = True
"PMI_PROP_NORMAL_DIR"	"Dx Dy Dz"	Each Dx, Dy, Dz is a F32 value using "%f" format
"PMI_PROP_APPROACH_DIR"	"Dx Dy Dz"	Each Dx, Dy, Dz is a F32 value using "%f" format
"PMI_PROP_CLAMPING_DIR"	"Dx Dy Dz"	Each Dx, Dy, Dz is a F32 value using "%f" format
"PMI_PROP_MEAS_DIR"	"Dx Dy Dz"	Each Dx, Dy, Dz is a F32 value using "%f" format
"PMI_PROP_COORD_DIR"	"Dx Dy Dz"	Each Dx, Dy, Dz is a F32 value using "%f" format
"PMI_PROP_MEAS_LEVEL"	"#"	Integer representing level number
"PMITextForegroundColour"	"#"	Hexadecimal integer representing RGB colour where value has "0x00bbggrr" form. The low-order byte contains a value for the relative intensity of red; the second byte contains a value for the relative intensity of green; and the third byte contains a value for the relative intensity of blue. The high-order byte shall be zero. The maximum value for a single byte is 0xFF (i.e. intensity value is in the range [0:255]).
"PMITextBackgroundColour"	"#"	Same as "PMITextForegroundColour"
"PMITextBackgroundOpacity"	"#"	Unsigned decimal integer representing opacity percentage. Actual opacity is: decoded# / 100.0
"PMITextShowBorder"	"#"	Unsigned decimal integer: 0 = = False; 1 = = True
"PMITextSize"	"#"	Unsigned decimal integer representing text size in units of pixels.
"PMITextInPlane"	"#"	Unsigned decimal integer: 0 = = False; 1 = = True where "1" indicates that text should be displayed in the plane of the entity so that it rotates with view.
"PMIGeometryColour"	"#"	Same as "PMITextForegroundColour"
"PMIGeometryWidth"	"#"	Unsigned decimal integer representing line width in units of pixels.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

"Key" Property Atom Value String	"Value" Property Atom Value String Encoding Format	Decoding Notes
CLIP_NORMAL	"#,#,#"	Used for Entity Type = "0x0114" and Entity Type Name String = "Section" to specify the normal to the clipping plane. The clipping normal points toward the piece of the model that will be clipped away. Each # is a F64 value using "%lf" format.
CLIP_POSITION	"#,#,#"	Used for Entity Type = "0x0114" and Entity Type Name String = "Section" to specify one point on the clipping plane. Each # is a F64 value using "%lf" format.
TRANSFORMATION_MATRIX	"#,#,#,#,#,#,#,#, #,#,#,#,#,#,#,#"	Used for Entity Type = "0x0114" and Entity Type Name String = "Part Transform" to specify a transformation matrix. Each # is a F32 value using "%f" format.

10.2.6.1.1 PMI Property Atom

PMI Property Atom data collection represents the data format for both the key and value data of a PMI Property key/value pair.

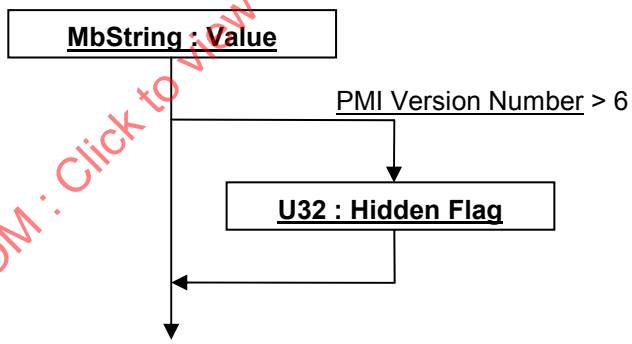


Figure 174 — PMI Property Atom data collection

MbString : Value

Value specifies the property atom value encoded into a String. See Table 84 — Common Property Keys and Their Value Encoding formats above for encoding formats of the Value string.

U32 : Hidden Flag

Hidden Flag specifies if the property is "hidden" or not. A JT file reader could use this flag to control whether read properties should be exposed to the end user of the application reading the JT file. Valid values include the following:

Table 85 — PMI Property Atom Hidden Flag values

= 0	Property is not hidden.
= 1	Property is hidden.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

10.2.7 PMI CAD Tag Data

The PMI CAD Tag Data collection contains the list of persistent IDs, as defined in the CAD System, to uniquely identify individual PMI entities. The existence of this PMI CAD Tag Data collection is dependent upon the value of previously read data field CAD Tags Flag as documented in 10.2 PMI Manager Meta Data Element.

If PMI CAD Tag Data collection is present, there will be a CAD Tag for each PMI entity as specified by the below documented CAD Tag Index Count formula.

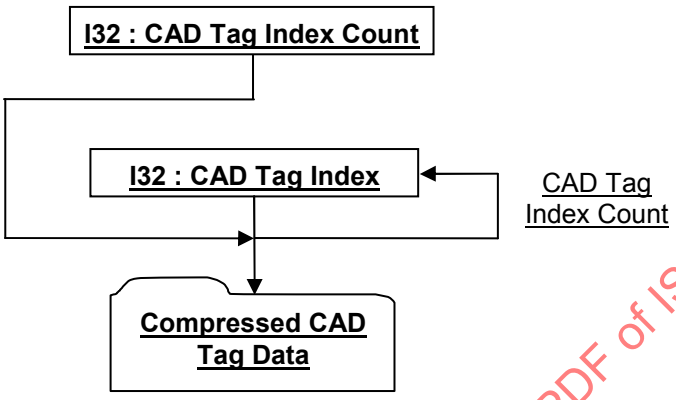


Figure 175 — PMI CAD Tag Data collection

Complete description for Compressed CAD Tag Data can be found in 13.1.16 Compressed CAD Tag Data.

I32 : CAD Tag Index Count

CAD Tag Index Count specifies the total number of CAD Tag indices. This value shall be equal to the summation of the previously read count values for all the PMI entities supporting CAD Tags. The formula is the sum of the following:

- Line Weld Count
- Spot Weld Count
- SF Count
- MP Count
- Reference Geometry Count
- Datum Target Count
- FCF Count
- Locator Count
- Dimension Count
- DFS Count
- Note Count
- Model View Count

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

— Design Group Count

— Coord Sys Count

— Generic Entity Count

I32 : CAD Tag Index

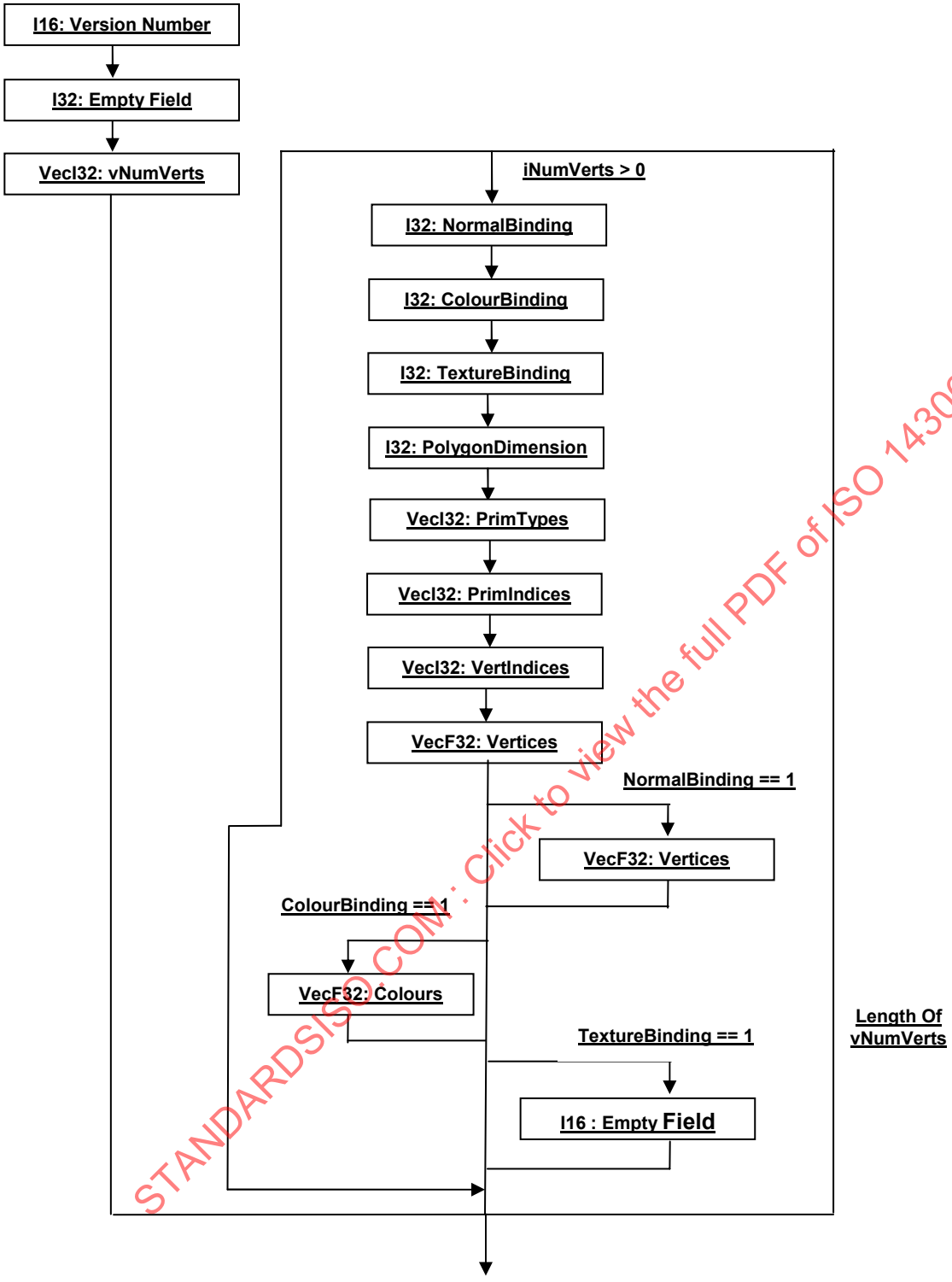
CAD Tag Index specifies an index into a list of CAD Tags, identifying the CAD Tag belonging to a particular PMI entity. There will be a total of CAD Tag Index Count number of CAD Tag Indices and the order of the indices will be as defined by the above documented CAD Tag Index Count formula (i.e. Line Weld CAD Tag Indices are first, followed by the Spot Weld CAD Tag Indices, followed by the Surface Finish CAD Tag Indices, etc.)

10.2.8 PMI Polygon Data

The PMI Polygon Data collection contains a list of vertices classified as polygonal primitives. Its composition is shown in the figure 177. Each block of PMI PolygonData contains a list of 0 or more PolygonData elements. Empty PolygonData elements are written with 0 vertices and no additional fields.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 176 — PMI Polygon Data

I16: Version Number

Version number is the version identifier for this PMI Polygon Data Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32: Empty Field

Refer to best practice 14.4 Empty Field

© ISO 2012 – All rights reserved

VecI32: vNumVerts

An integer vector used to record the number of vertices in each polygon data element. The length of this vector is equal to the number of PolygonData elements written in this block of PMI PolygonData. The presence of additional data fields in each PolygonData element is hinged upon that element having more than 0 vertices recorded in this vector.

Retrieve next vertCount from vNumVerts

If the next element in the vNumVerts vector is non-zero, proceed to read other fields that make up a single PMI PolygonData element. Otherwise, skip reading more data for this element and loop back to seek the next element in the vector.

iNumVerts

Number of vertices for the i^{th} PolygonData element.

Length Of vNumVerts

Number of Polygon Data elements.

I32: NormalBinding

A Boolean value that indicates if there are normals present along with the list of coordinates at each vertex.

I32: ColourBinding

A Boolean value that indicates if there are colours present along with the list of coordinates at each vertex.

I32: TextureBinding

A Boolean value that indicates if there are Texture Coordinates present along with the list of coordinates at each vertex.

I32: PolygonDimension

Indicates the dimension of vertex coordinates.

VecI32: PrimTypes

An array indicating the type of each of the primitive stored in the PrimIndices array. Adjacent numbers in the array form tuples of the form [PrimIndex, PrimType]. All primitives to the left of the PrimIndex are of type PrimType unless they are already to the left of an earlier PrimIndex in this array.

VecI32: PrimIndices

Indices of vertices that form a single primitive. The difference between two adjacent values in this array determines the length of the primitive. An extra element is stored at the end of this array to identify the length of the last primitive. Values in this array are indices into the VertIndices array.

VecI32: VertIndices

An array of indices into the Vertices array. This index array eliminates the need to duplicate floating point vertices that are shared by multiple primitives.

VecF32: Vertices

The list of vertex coordinates. Each vertex is made of PolygonDimension coordinates. The length of this list is equal to number of vertices multiplied by PolygonDimension.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecF32: Normals

An optional list of Normals for each vertex. Presence of this list is indicated by the NormalBinding flag. Each normal consists of PolygonDimension components. The size of this list is equal to number of vertices multiplied by PolygonDimension.

VecF32: Colours

An optional list of Colours for each vertex. Presence of this list is indicated by the ColourBinding flag. Each colour consists of PolygonDimension components. The size of this list is equal to number of vertices multiplied by PolygonDimension.

VecF32: Texture Coords

An optional list of Texture coordinates for each vertex. Presence of this list is indicated by the TexCoordBinding flag. Each TexCoord consists of 2 components. The size of this list is equal to number of vertices multiplied by 2.

11 PMI Data Segment

The PMI Manager Meta Data Element (as documented in 10.2 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 10.2 PMI Manager Meta Data Element.

The recommended best practice for working with PMI in a JT file is to use the descriptions found in 10.2 PMI Manager Meta Data Element as opposed to working with PMI Data Segment.

12 JT ULP Segment

JT ULP Segment contains an Element that defines the semi-precise geometric Boundary Representation data for a particular Part in JT ULP format. Note that there is also two other Boundary Representation formats (i.e. JT B-Rep and XT B-Rep) supported by the JT file format within a different file Segment Type. Complete description for the JT B-Rep and the XT B-Rep can be found in 7 JT B-Rep Segment and

8 XT B-Rep Segment respectively.

JT ULP Segments are typically referenced by Part Node Elements (see 5.1.1.5 Part Node Element) using Late Loaded Property Atom Elements (see 5.2.7 Late Loaded Property Atom Element). The JT ULP Segment type supports ZLIB compression on all element data, so all elements in JT ULP Segment use the Logical Element Header ZLIB form of element header data.

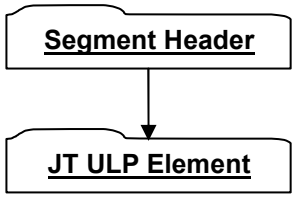


Figure 177 — JT ULP Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

12.1 JT ULP Element

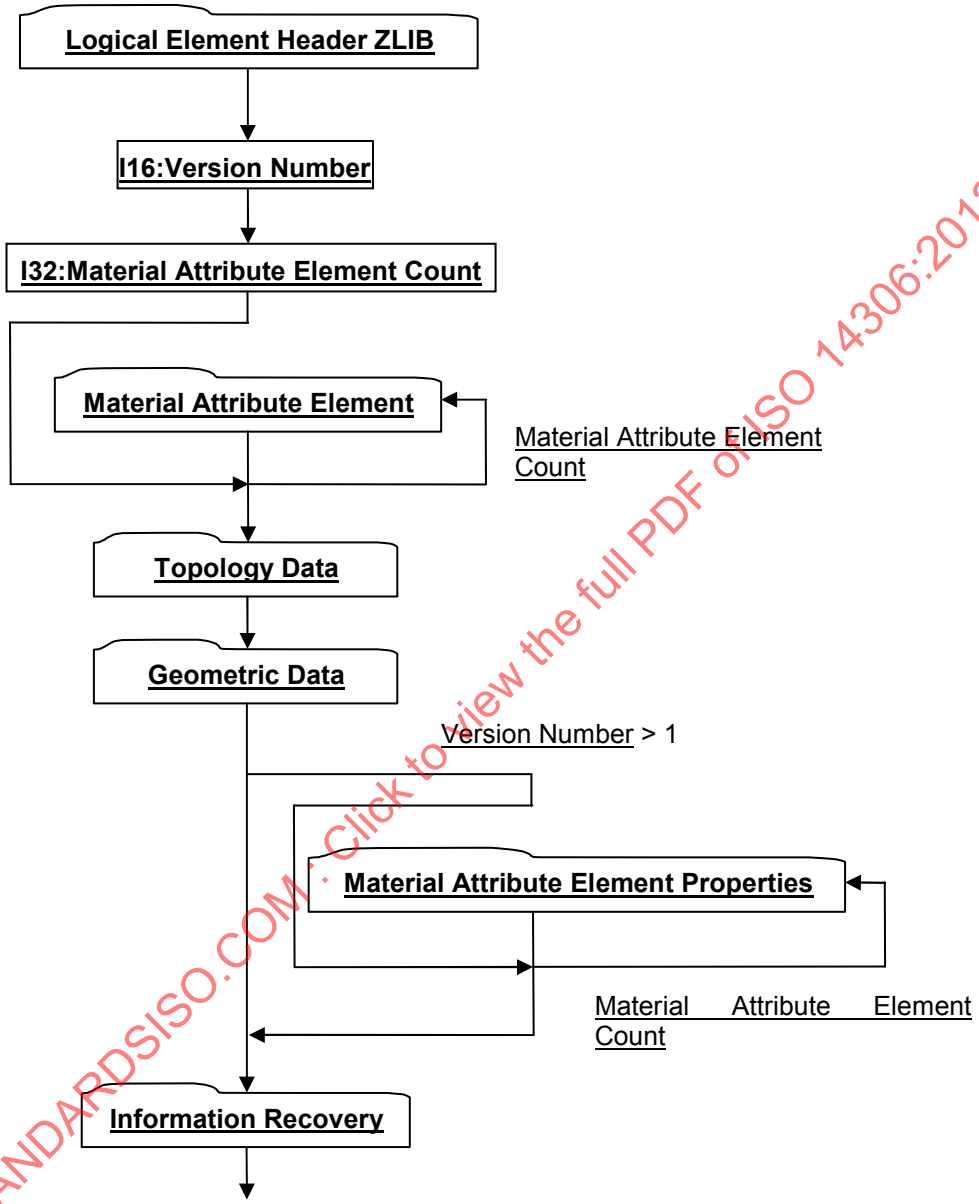
Object Type ID: 0xf338a4af, 0xd7d2, 0x41c5, 0xbc, 0xf2, 0xc5, 0x5a, 0x88, 0xb2, 0x1e, 0x73

JT ULP Element represents a particular Part's ultra-lightweight semi-precise B-Rep data. Like JT B-Rep Element or XT B-Rep Element, JT ULP Element contains all the topological and geometric information that

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

describes the shape of a part. The difference is that the size of JT ULP Element is typically around 10% of a typical JT file with B-Rep and LODs, and this is achieved by sophisticated compression techniques. In addition, JT ULP Element is semi-precise meaning that its geometric description is not as precise as either JT B-Rep Element or XT B-Rep Element. The precision loss of JT ULP Element, however, is carefully controlled to be equal to or better than 0.01% of the part size or 0.1mm, whichever is smaller.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 178 — JT ULP Element data collection

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

I16:Version Number

Version Number is the version identifier for this JT ULP Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32:Material Attribute Element Count

Material Attribute Element Count is the number of material attribute elements.

© ISO 2012 – All rights reserved

Complete description for Material Attribute Element can be found in 5.1.2.2 Material Attribute Element.

12.1.1 Topology Data

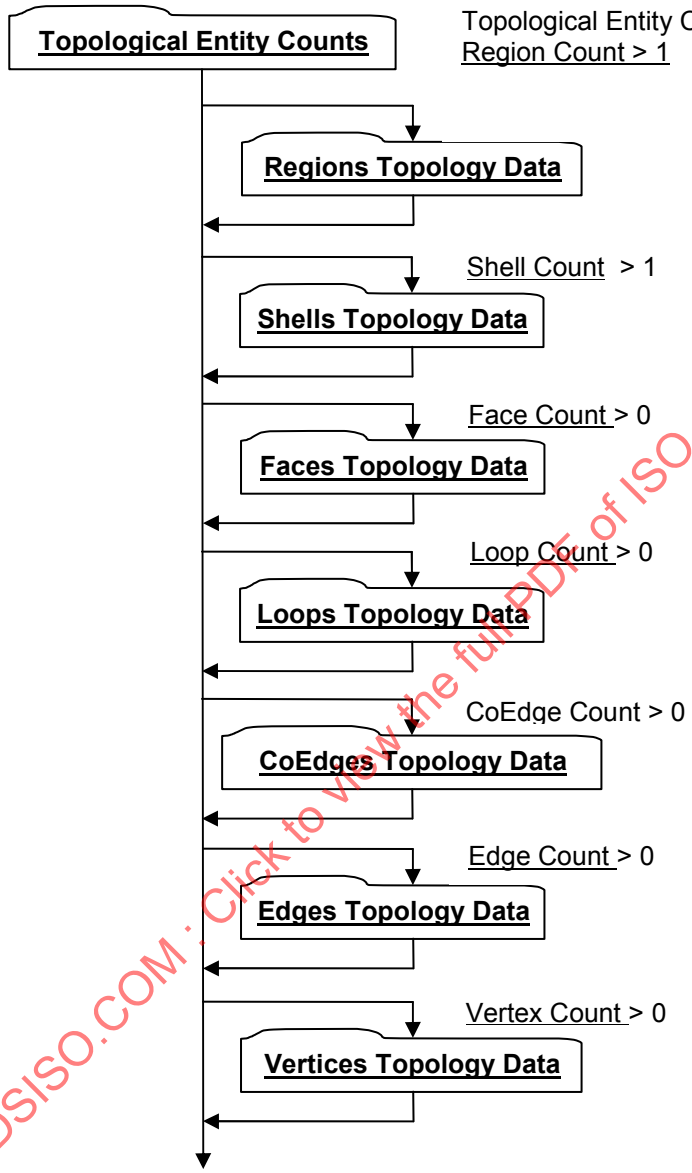


Figure 179 — Topology Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Topological Entity Counts

Topological Entity Counts data collection defines the counts for each of the various topological entities within a ULP.

© ISO 2012 – All rights reserved

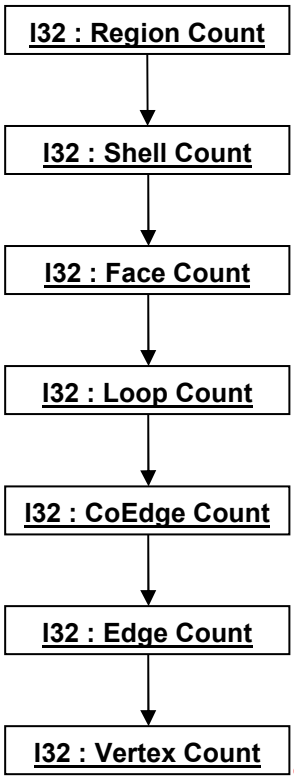


Figure 180 — Topological Entity Counts data collection

I32 : Region Count

Region Count indicates the number of topological region entities in the ULP.

I32 : Shell Count

Shell Count indicates the number of topological shell entities in the ULP.

I32 : Face Count

Face Count indicates the number of topological face entities in the ULP.

I32 : Loop Count

Loop Count indicates the number of topological loop entities in the ULP.

I32 : CoEdge Count

CoEdge Count indicates the number of topological coedge entities in the ULP.

I32 : Edge Count

Edge Count indicates the number of topological edge entities in the ULP.

I32 : Vertex Count

Vertex Count indicates the number of topological vertex entities in the ULP.

Combined Predictor Type

A predictor type may be combined with additional processing. When Combined Predictor Type is used, the additional processing step is encoded. For example, combined predictor type *Combined:NULL* means that the data collection follows the logical diagram in Figure 181 with ePredictorType set to be NULL.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

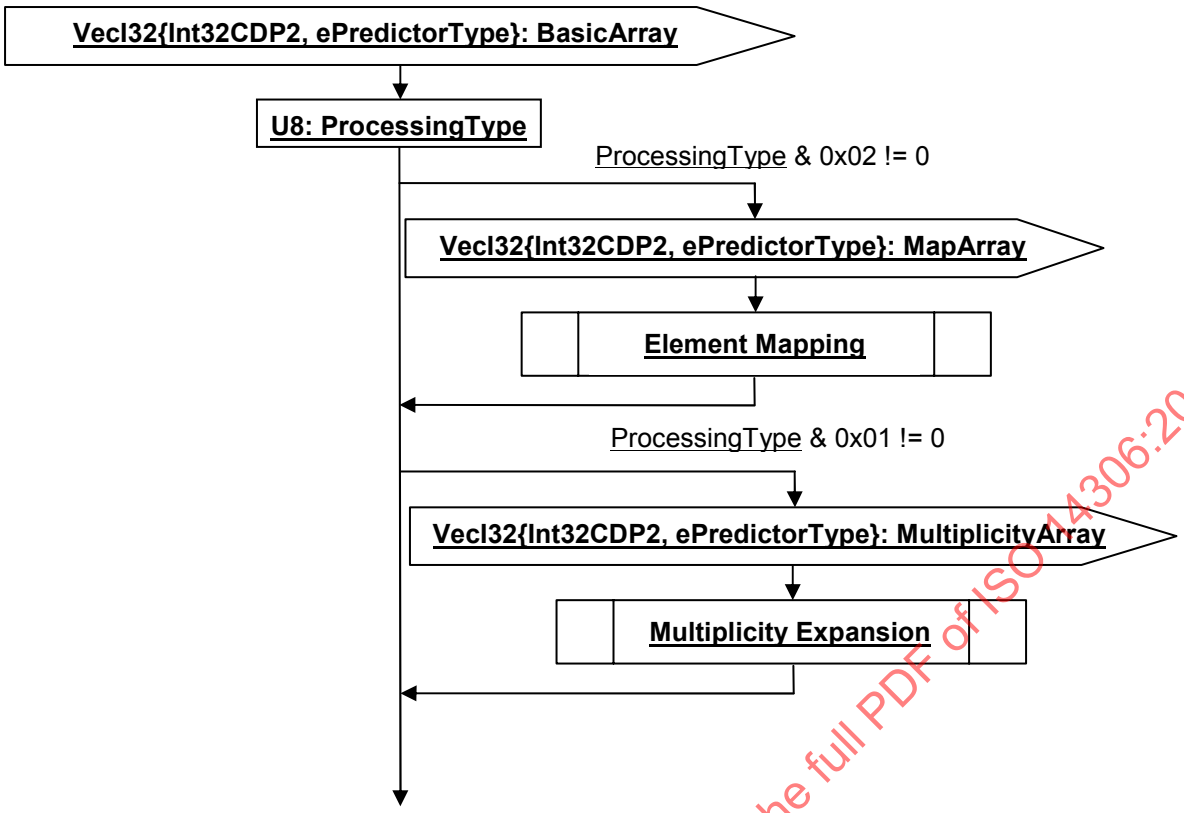


Figure 181 — Combined PredictorType data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Vecl32{Int32CDP2, ePredictorType}: BasicArray

BasicArray is an integer array, compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

U8: ProcessingType

Two bits of this value are currently used. If bit 0x02 is set, then the integer array is a list of elements with unique values and Element Mapping step is needed to recover the original values. If bit 0x01 is set, then the some elements in the integer array may be repeated, and Multiplicity Expansion is used to recover the original values.

Vecl32{Int32CDP2, ePredictorType}: MapArray

MapArray is an integer array, where each element represents the index mapping information. MapArray is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Element Mapping

Element Mapping recovers the original array from BasicArray and MapArray, using relationship $\mathrm{OriginalArray}[i] = \mathrm{BasicArray}[\mathrm{MapArray}[i]]$. After Element Mapping, the value of $\mathrm{BasicArray}$ is updated with $\mathrm{OriginalArray}$.

Vecl32{Int32CDP2, ePredictorType}: MultiplicityArray

MultiplicityArray is an integer array, where each element represents the multiplicity of each element in BasicArray. MultiplicityArray is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

© ISO 2012 – All rights reserved

Multiplicity Expansion

Multiplicity Expansion recovers the original array from BasicArray and MultiplicityArray. The original array is an expansion of the BasicArray. If the corresponding multiplicity value is greater than 1, the element in BasicArray is contiguously repeated in the original array according to multiplicity value.

Regions Topology Data

Regions Topology Data defines the disjoint set of non-overlapping Shells making up each Region. Each Region is defined by one or more non-overlapping Shells. The volume of a Region is that volume lying inside each "anti-hole Shell" and outside each simply-contained "hole Shell" belonging to the particular Region. A Region is analogous to a dimensionally elevated face where Region corresponds to Face and Shell corresponds to Trim Loop.

Each Region's defining Shells are identified in a list of Shells by an index for both the first Shell and the last Shell in each Region (i.e. all Shells inclusive between the specified first and last Shell list index define the particular Region). In addition, the indices of all the shells in a single Region are contiguous. The first shell index of the first region is 0, and the first shell index of other regions is one greater than the last shell index of the previous region. Therefore only the number of shells of each region is stored. In the special case when the number of regions is 1, no information needs be stored since its last Shell index is known to be Shell Count-1.

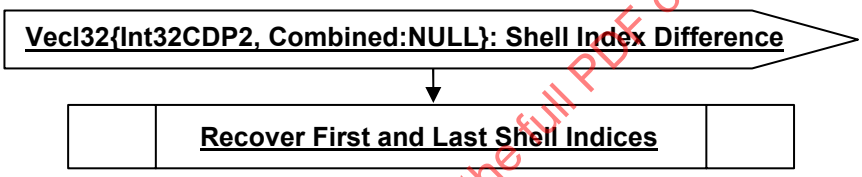


Figure 182 — Regions Topology Data collection

Vecl32{Int32CDP2, Combined:NULL}: Shell Index Difference

Shell Index Difference is a vector of indices representing the integer value by subtracting first shell index from last shell index in each region, encoded using Combined Predictor Type. Shell Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover First and Last Shell Indices

The first shell index of the first region is 0, and the last shell index of the first region is element 0 of Shell Index Difference. The first shell index of region $k, k \geq 1$ equals to the last shell index of region $k - 1$ plus 1. The last shell index of region $k, k \geq 1$ equals to the first shell index of region k plus element k of Shell Index Difference array.

Shells Topology Data

Shells Topology Data defines the set of topological adjacent Faces making up each Shell. A Shell's set of topological adjacent Faces define a single (usually closed) two manifold solid that in turn defines the boundary between the finite volume of space enclosed within the Shell and the infinite volume of space outside the Shell. In addition, each Shell has a flag that denotes whether the Shell refers to the finite interior volume (i.e. a "hole Shell") or the infinite exterior volume (i.e. an "anti-hole Shell").

Each Shell's defining Faces are identified in a list of Faces by an index for both the first Face and the last Face in each Shell (i.e. all Faces inclusive between the specified first and last Face list index define the particular Shell). In addition, the indices of all the faces in a single Shell are contiguous. The first face index of the first shell is 0, and the first face index of other shells is one greater than the last face index of the previous shell. Therefore only the number of faces of each shell is stored. In the special case when the number of shells is 1, no information needs be stored since its last face index is known to be Face Count-1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

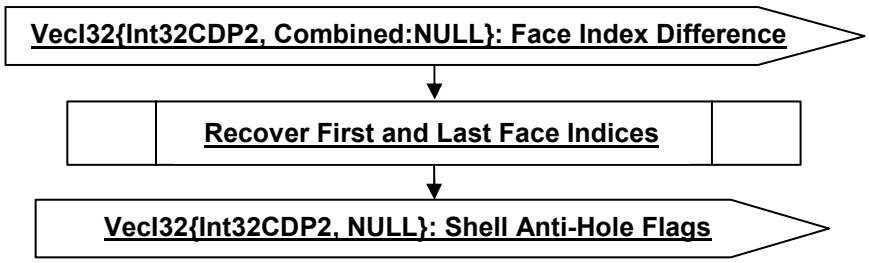


Figure 183 — Shells Topology Data collection

Vecl32{Int32CDP2, Combined:NULL}: Face Index Difference

Face Index Difference is a vector of indices representing the integer value by subtracting first face index from last face index in each shell, encoded using Combined Predictor Type. Face Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover First and Last Face Indices

The first face index of the first shell is 0, and the last face index of the first shell is element 0 of Face Index Difference. The first face index of shell $k, k \geq 1$ equals to the last face index of shell $k-1$ plus 1. The last face index of shell $k, k \geq 1$ equals to the first face index of shell k plus element k of Face Index Difference array.

Vecl32{Int32CDP2, NULL}: Shell Anti-Hole Flags

Each Shell has a flag identifying whether the Shell is an anti-hole Shell. Shell Anti-Hole Flags is a vector of anti-hole flags for a set of Shells.

In an uncompressed/decoded form the flag values have the following meaning:

Table 86 — JT ULP Shell Anti-Hole Flag values

= 0	Shell is not an anti-hole Shell
= 1	Shell is an anti-hole Shell

Shell Anti-Hole Flags uses the Int32 version of the CODEC to compress and encode data.

Faces Topology Data

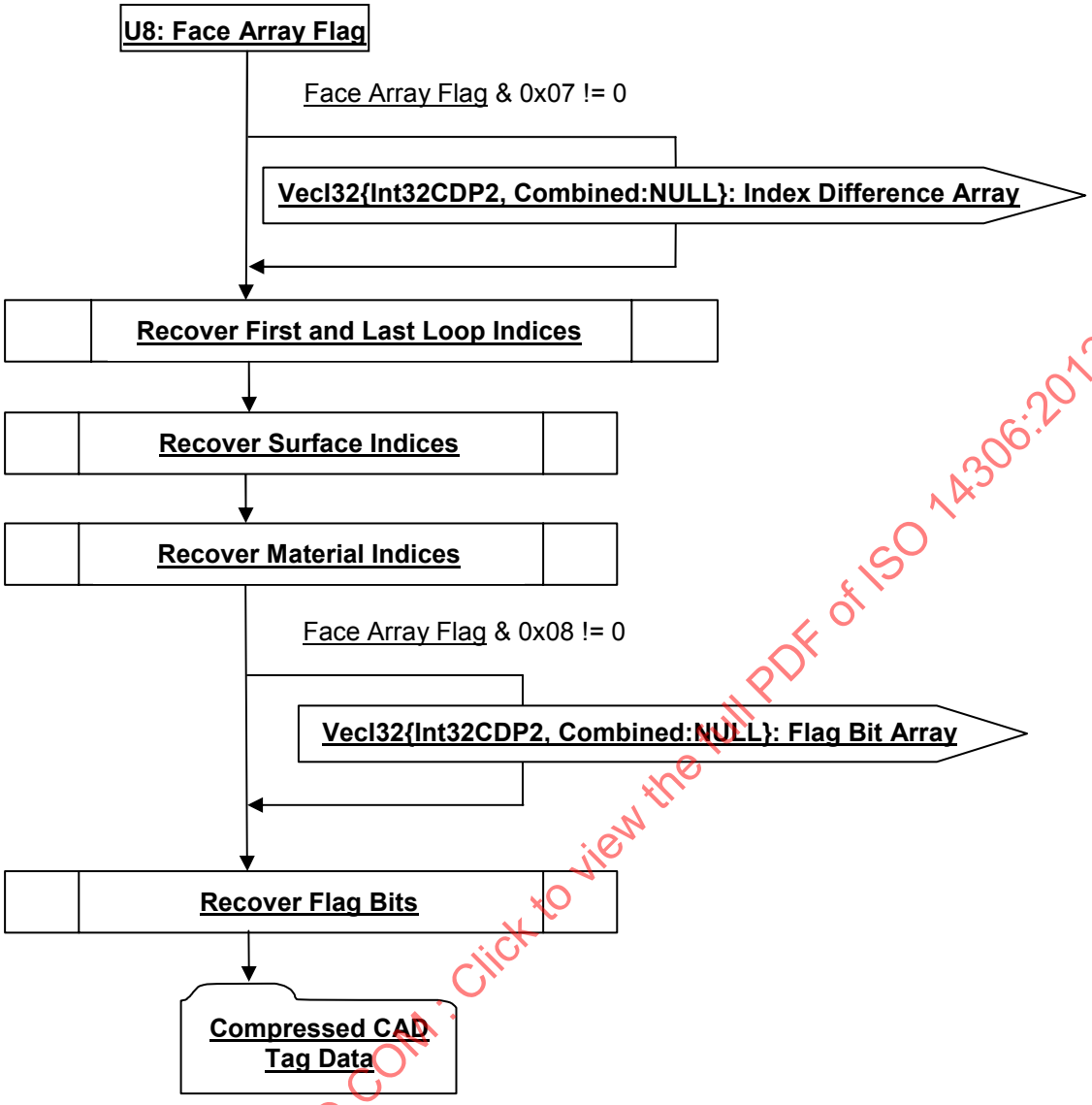
A Face shall be trimmed with at least one “anti-hole” Trim Loop and may be trimmed with one or more “hole” Trim Loops. The complete description of face and its relation to the trim loops can be found in Faces Topology Data.

Each Face’s defining Trim Loops are identified in a list of trim Loops by an index for both the first Trim Loop and the last Trim Loop in each Face (i.e. all Trim Loops inclusive between the specified first and last Trim Loop list index define the particular Face). In addition, the indices of all the loops in a single Face are contiguous. The first loop index of the first face is 0, and the first loop index of other faces is one greater than the last loop index of the previous face. Therefore only the number of loops of each face is stored. In the special case when the number of faces is 1, no information needs be stored since its last loop index is known to be Loop Count-1.

Each Face’s underlying Geometric Surface is identified by an index into a list of Geometric Surfaces. Each face’s material is identified by an index into the list of Material Attribute Elements.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 184 — Faces Topology Data collection

U8: Face Array Flag

Face Array Flag indicates which arrays of face topology data are not trivial and therefore encoded.

Vecl32{Int32CDP2, Combined:NULL}: Index Difference Array

Index Difference Array is a combined vector of indices encoded using Int32 version of CODEC and Combined Predictor Type, with its content decided by the value of Face Array Flag. If Face Array Flag has bit 0x01 set, then the vector of integer values obtained by subtracting first loop index from last loop index in each face is appended to the end of Index Difference Array. If Face Array Flag has bit 0x02 set, then the vector of integer values obtained by subtracting surface index from face index in each face is appended to the end of Index Difference Array. If Face Array Flag has bit 0x04 set, then the vector of integer values representing the material index of each face is appended to the end of Index Difference Array.

Recover First and Last Loop Indices

The first loop index of the first face is 0, and the last loop index of the first face is element 0 of Index Difference Array if the array is encoded, or 0 if bit 0x01 of Face Array Flag is not set. The first loop index of face $k, k \geq 1$ equals to the last loop index of face $k-1$ plus 1. The last loop index of face $k, k \geq 1$ equals to

© ISO 2012 – All rights reserved

the first loop index of face k plus element k of Index Difference Array, or 0 if bit 0x01 of Face Array Flag is not set.

Recover Surface Indices

The surface index of each face equals to the face index if bit 0x02 of Face Array Flag is not set. Otherwise the surface index of face k is obtained by substracting element $k + offset$ of Index Difference Array from face index k, where $offset$ is equal to Face Count if bit 0x01 of Face Array Flag is set and 0 if the bit is not set.

Recover Material Indices

The material index of each face equals to 0 if bit 0x04 of Face Array Flag is not set. Otherwise the material index of face k equals to the element $k + offset$ of Index Difference Array, where $offset$ is equal to twice of Face Count if both bit 0x01 and bit 0x02 of Face Array Flag are set, is equal to Face Count if either bit 0x01 or bit 0x02 of Face Array Flag is set, and is equal to 0 if neither bit is set.

Vecl32{Int32CDP2, Combined:NULL}: Flag Bit Array

Only the lower 24 bits of the four integer indices, namely first loop index, last loop index, surface index, and material index, are used as integer identifiers. The other bits of these integers are documented as reserved for future use.

Table 87 — JT ULP Flag Bit Array Look Index values

	24	25	26	27	28	29	30	31
First Loop Index	Surface Type			U Knot Type		V Knot Type		isNormalReversed
Last Loop Index	isIsolated	Reserved						
Surface Index	Reserved							
Material Index	Reserved							

Each element of Flag Bit Array is a 32 bit integer obtained by combining all 32 flag bits from four different integers. More specifically:

— Bits 0~7 of Flag Bit Array are equal to bits 24~31 of First Loop Index.

— Bits 8~15 of Flag Bit Array are equal to bits 24~31 of Last Loop Index.

— Bits 16~23 of Flag Bit Array are equal to bits 24~31 of Surface Index.

— Bits 24~31 of Flag Bit Array are equal to bits 24~31 of Material Index.

Supported Surface Type

In an uncompressed/decoded form, the supported surface types are listed below.

Table 88 — JT ULP Supported Surface Type values

0	Nurbs
1	Plane
2	Cylinder
3	Cone
4	Sphere
5	Torus
6	Reserved
7	Reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Supported Knot Type

In an uncompressed/decoded form, the supported knot types are listed below. The knot type of the underlying surface along both U and V parameter directions are encoded.

Table 89 — JT ULP Supported Knot Type Values

0	No Pattern
1	No knot value in between the clamped end knots
2	All knot values in between the end knots increase with an even interval
3	All knot values in between the end knots repeat exactly once, and the distinct values increase with an even interval

In an uncompressed/decoded form, the Face Reverse Normal Flag has the following meaning:

Table 90 — JT ULP Face Reverse Normal Flag values

= 0	Face normal is not reversed
= 1	Face normal is reversed.

Recover Flag Bits

If Face Array Flag & 0x08 is equal to 0, then each element in Flag Bit Array is set to have value 0. The flag bits are recovered by assigning bits 0~7 of Flag Bit Array to bits 24~31 of First Loop Index, bits 8~15 of Flag Bit Array to bits 24~31 of Last Loop Index, bits 16~23 of Flag Bit Array to bits 24~31 of Surface Index, and bits 24~31 of Flag Bit Array to bits 24~31 of Material Index.

Loops Topology Data

A Loop (often called Trimming Loop) defines in parameter space a 1D boundary around which geometric surfaces are trimmed to form a Face. Loops Topology Data specifies the CoEdges making up each Loop along with an anti-hole flag and identifier tag for each Loop. The complete description of loop and its relation to the CoEdges can be found in Loops Topology Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

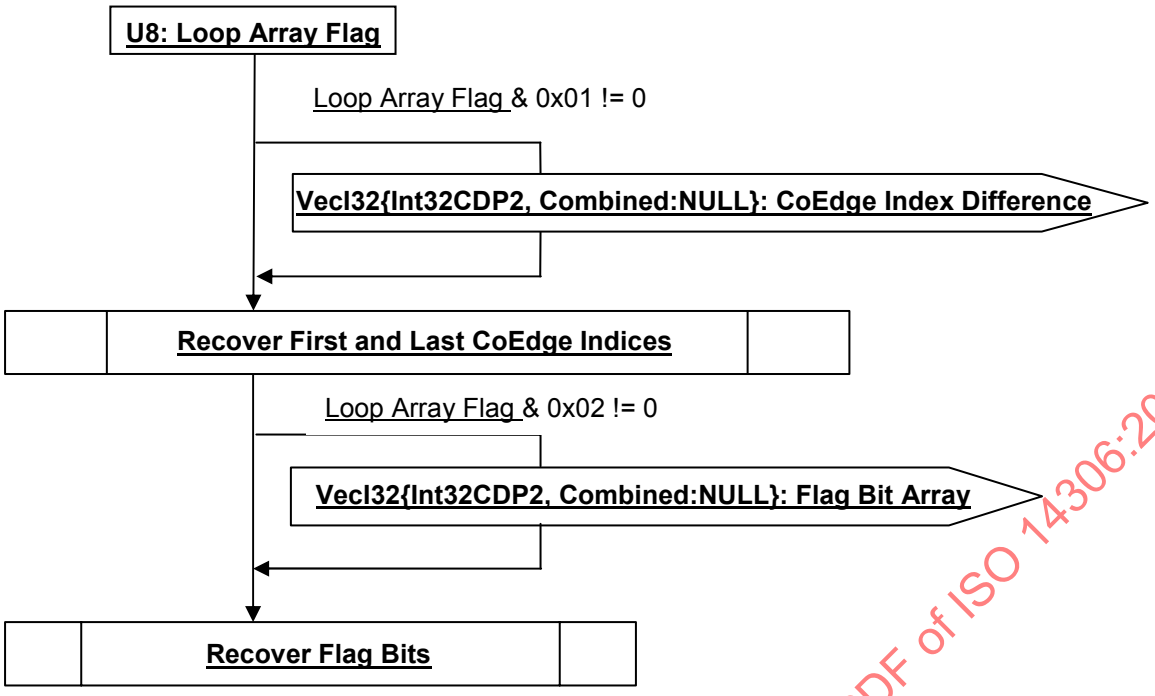


Figure 185 — Loops Topology Data collection

U8: Loop Array Flag

Loop Array Flag indicates which arrays of loop topology data are not trivial and therefore encoded.

Vecl32{Int32CDP2, Combined:NULL}: CoEdge Index Difference

CoEdge Index Difference is a vector of indices representing the integer value by subtracting first CoEdge index from last CoEdge index in each loop, encoded using Combined Predictor Type. CoEdge Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover First and Last CoEdge Indices

The first CoEdge index of the first loop is 0, and the last CoEdge index of the first loop is element 0 of CoEdge Index Difference. The first CoEdge index of loop $k, k \geq 1$ equals to the last CoEdge index of loop $k-1$ plus 1. The last CoEdge index of loop $k, k \geq 1$ equals to the first CoEdge index of loop k plus element k of CoEdge Index Difference array.

Vecl32{Int32CDP2, Combined:NULL}: Flag Bit Array

Only the lower 24 bits of the two integer indices, namely first CoEdge index and last CoEdge index are used as integer identifiers. The other bits of these integers documented as reserved for future use.

Table 91 — JT ULP Loops Topology Flag Bit Array values

	24	25	26	27	28	29	30	31
First CoEdge Index	Reserved							isAntiHoleLoop
Last CoEdge Index	Reserved							

Bits 0~7 of Flag Bit Array are equal to bits 24~31 of First CoEdge Index

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Bits 8~15 of Flag Bit Array are equal to bits 24~31 of Last CoEdge Index

Bits 16~31 of Flag Bit Array are set to be 0

In an uncompressed/decoded form, the AntiHole Loop Flag has the following meaning:

Table 92 — JT ULP Loops Topology Reverse Normal Flag values

= 0	Loop is not an anti-hole Loop
= 1	Loop is an anti-hole Loop

Recover Flag Bits

The flag bits are recovered by assigning bits 0~7 of Flag Bit Array to bits 24~31 of First CoEdge Index, and bits 8~15 of Flag Bit Array to bits 24~31 of Last CoEdge Index.

CoEdges Topology Data

A CoEdge defines a parameter space edge trim Loop segment (i.e. the projection of an Edge into the parameter space of the Face). CoEdges Topology Data specifies the underlying Edge and PCS Curve making up each CoEdge along with a MCS curve reversed flag and tag for each CoEdge. The complete description of CoEdge and its relation to the Edge can be found in CoEdges Topology Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

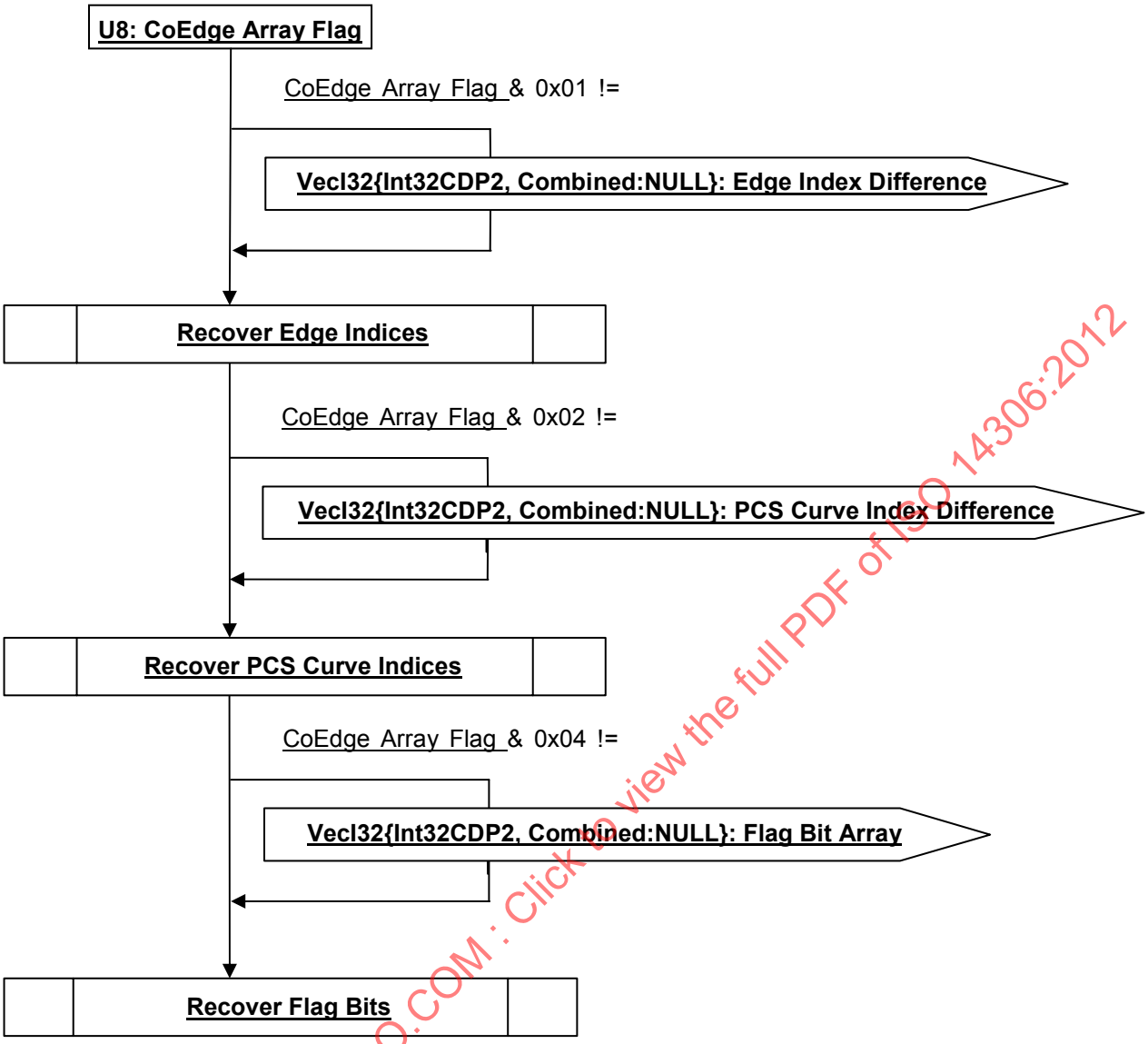


Figure 186 — CoEdges Topology Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

U8: CoEdge Array Flag

CoEdge Array Flag indicates which arrays of coedge topology data are not trivial and therefore encoded.

Vecl32{Int32CDP2, Combined:NULL}: Edge Index Difference

Edge Index Difference is a vector of indices representing the integer value by subtracting the Edge index from the CoEdge index for each CoEdge, encoded using Combined Predictor Type. Edge Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover Edge Indices

If CoEdge Array Flag & 0x01 is equal to 0, then the Edge index of each CoEdge is equal to the CoEdge index. Otherwise, the Edge index of CoEdge with index k can be computed by substracting element k of Edge Index Difference array from k, the CoEdge index.

© ISO 2012 – All rights reserved

Vecl32{Int32CDP2, Combined:NULL}: PCS Curve Index Difference

PCS Curve Index Difference is a vector of indices representing the integer value by subtracting the PCS Curve index from the CoEdge index for each CoEdge, encoded using Combined Predictor Type. PCS Curve Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover PCS Curve Indices

If CoEdge Array Flag & 0x02 is equal to 0, then the PCS Curve index of each CoEdge is equal to the CoEdge index. Otherwise, the PCS Curve index of CoEdge with index k can be computed by substracting element k of PCS Curve Index Difference array from k, the CoEdge index.

Vecl32{Int32CDP2, Combined:NULL}: Flag Bit Array

Only the lower 24 bits of the two integer indices, namely Edge index and PCS Curve index, are used as integer identifiers. The other bits of these integers are documented as reserved for future use

Table 93 — JT ULP Recover Edge Indices Flag Bit Array values

	24	25	26	27	28	29	30	31
Edge Index	Knot Type		Domain Type			PCS Curve Type		isXYZReversed
PCS Curve Index	isUvInc	Reserved						

Bits 0~7 of Flag Bit Array are equal to bits 24~31 of Edge Index

Bits 8~15 of Flag Bit Array are equal to bits 24~31 of PCS Curve Index

Bits 16~31 of Flag Bit Array are set to be 0

The Knot Type, defined in Supported Knot Type, is an integer with its value between 0 and 3.

Domain Type

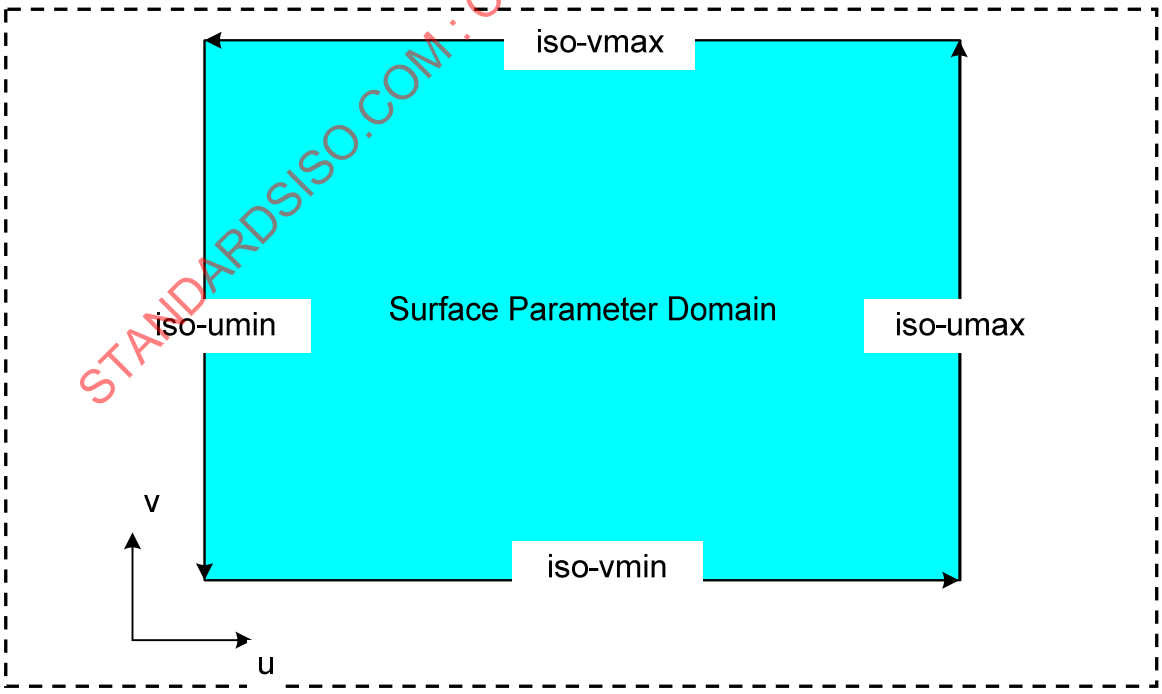


Figure 187 — Surface Domain Classification

In an uncompressed/decoded form, the supported PCS Curve types are listed below.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 94 — JT ULP Recover Edge Indices PCS curve type values

0	General
1	PCS curve is coincident with iso-umin curve in the surface parameter domain
2	PCS curve is coincident with iso-umax curve in the surface parameter domain
3	PCS curve is coincident with iso-vmin curve in the surface parameter domain
4	PCS curve is coincident with iso-vmax curve in the surface parameter domain
5	Reserved
6	Reserved
7	PCS curve is to be derived from MCS curve and surface geometry

PCS Curve Type

In an uncompressed/decoded form, the supported PCS Curve types are listed below.

Table 95 — JT ULP PCS Curve Type values

0	Nurbs
1	Line
2	Circle
3	Reserved

In an uncompressed/decoded form, the XYZReversed Flag has the following meaning:

Table 96 — JT ULP PCS Curve Type XYZ Reversed Flag values

= 0	Directional sense of associated edges MCS curve should not be interpreted as opposite the direction its parameterization implies.
= 1	Directional sense of associated edges MCS curve should be interpreted as opposite the direction its parameterization implies.

In an uncompressed/decoded form, the isUVInc Flag has the following meaning:

Table 97 — JT ULP PCS Curve Type isUVInc Flag values

= 0	PCS Curve is iso-parameteric in surface parameter domain in one direction and the parameter increases in the other direction
= 1	PCS Curve is iso-parameteric in surface parameter domain in one direction and the parameter decreases in the other direction

The isUVInc flag is set only if the Domain Type of this CoEdge has value between 1 and 4 inclusive.

Recover Flag Bits

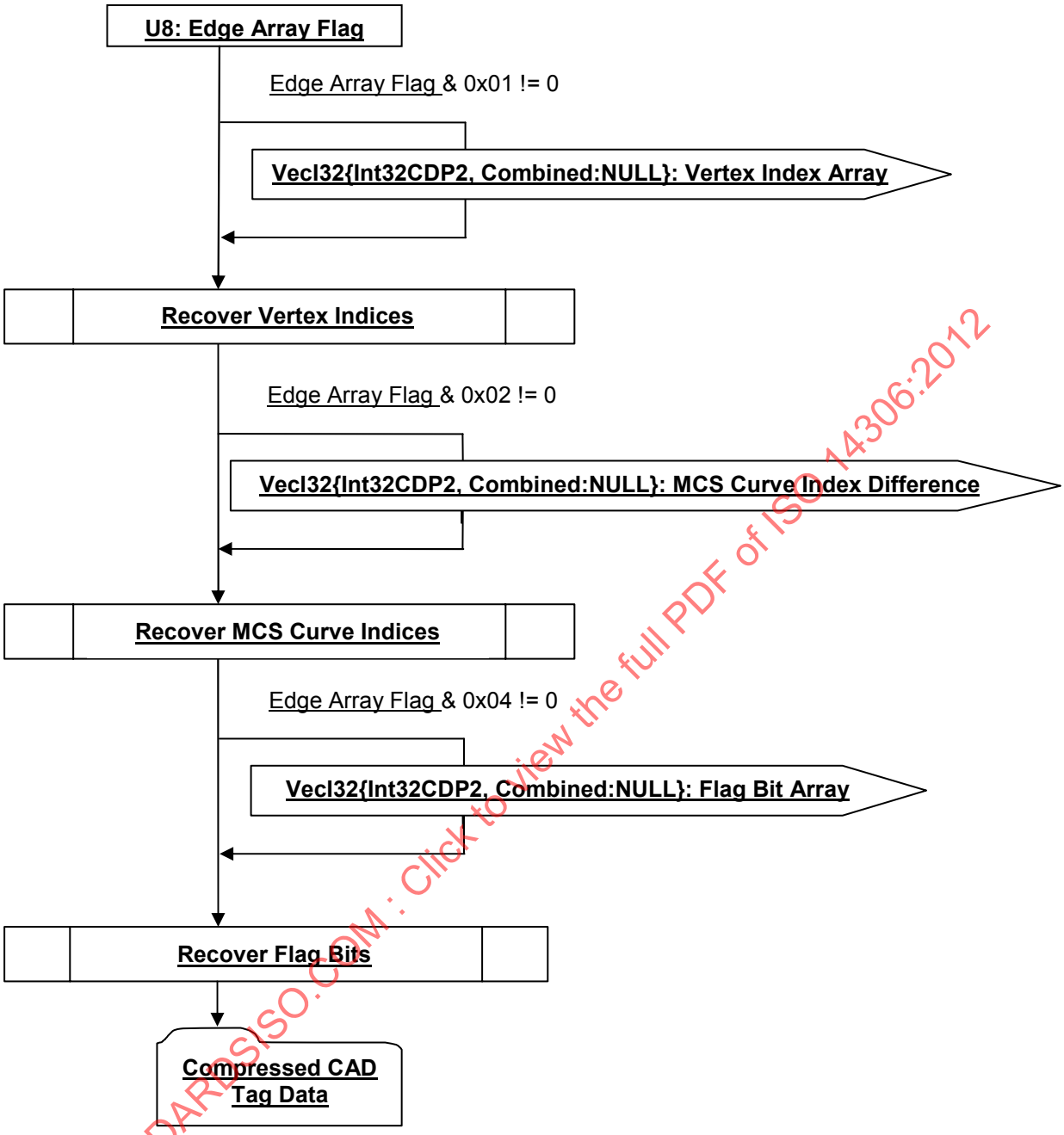
If CoEdge Array Flag & 0x04 is equal to 0, then each element in Flag Bit Array is set to have value 0. The flag bits are recovered by assigning bits 0~7 of Flag Bit Array to bits 24~31 of Edge Index, and bits 8~15 of Flag Bit Array to bits 24~31 of PCS Curve Index.

Edges Topology Data

An Edge defines a model space trim Loop segment. Edges Topology Data specifies the underlying MCS Curve and start and end Vertex making up each Edge along with an identification tag for each Edge. The complete description of Edge can be found in Edges Topology Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 188 — Edges Topology Data collection

U8: Edge Array Flag

Edge Array Flag indicates which arrays of edge topology data are not trivial and therefore encoded.

Vecl32{Int32CDP2, Combined:NULL}: Vertex Index Array

Vertex Index Array is a vector of indices representing the start and end vertex indices of each Edge, encoded using Combined Predictor Type. Vertex Index Array is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

© ISO 2012 – All rights reserved

Recover Vertex Indices

If Edge Array Flag & 0x01 is equal to 0, then all the vertex indices of each edge are set to be 0. Otherwise, the start vertex index of Edge with index k is set to be equal to element $2*k$ of Vertex Index Array, while the end vertex index of this Edge is set to be equal to element $2*k+1$ of Vertex Index Array.

Vecl32{Int32CDP2, Combined:NULL}: MCS Curve Index Difference

MCS Curve Index Difference is a vector of indices representing the integer value by subtracting the MCS Curve index from the Edge index for each Edge, encoded using Combined Predictor Type. MCS Curve Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover MCS Curve Indices

If Edge Array Flag & 0x02 is equal to 0, then the MCS Curve index of each Edge is equal to the Edge index. Otherwise, the MCS Curve index of Edge with index k can be computed by substracting element k of MCS Curve Index Difference array from k, the Edge index.

Vecl32{Int32CDP2, Combined:NULL}: Flag Bit Array

Only the lower 24 bits of the three integer indices, namely MCS Curve index, Start Vertex index, and End Vertex index, are used as integer identifiers. The other bits of these integers documented as reserved for future use..

Table 98 — JT ULP Edges Topology Recover MCS Curve Indices Flag Bit Array values

	24	25	26	27	28	29	30	31
MCS Curve Index	Knot Type		MCS Curve Type		Reserved			
Start Vertex Index	Reserved							
End Vertex Index	Reserved							

The Knot Type, defined in Supported Knot Type, is an integer with its value between 0 and 3.

MCS Curve Type

In an uncompressed/decoded form, the supported MCS Curve types are listed below.

Table 99 — JT ULP Edges Topology Recover MCS Curve Type values

0	Nurbs
1	Line
2	Circle
3	Projection: MCS curve geometry is to be computed from surface geometry and/or PCS curve geometry

Recover Flag Bits

If Edge Array Flag & 0x04 is equal to 0, then each element in Flag Bit Array is set to have value 0. The flag bits are recovered by assigning bits 0~7 of Flag Bit Array to bits 24~31 of MCS Curve Index.

Vertices Topology Data

A Vertex is the simplest topological entity and is basically made up of a geometric Point. Vertices Topology Data specifies the underlying geometric Point making up each Vertex. A Vertex is usually shared/referenced by two or more Edges (e.g. if the corners of four rectangular Faces touches at a common point, this point is represented by a Vertex and is shared by four Edges).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

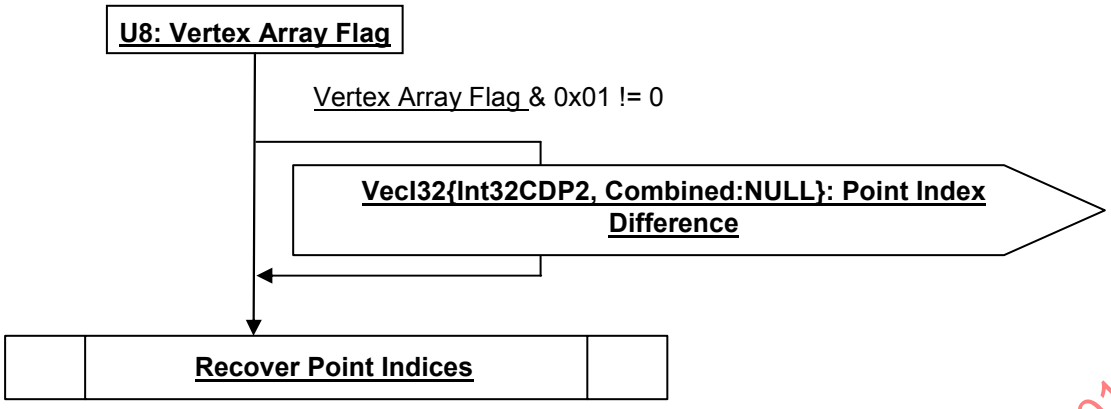


Figure 189 — Vertices Topology Data collection

U8: Vertex Array Flag

Vertex Array Flag indicates which arrays of vertex topology data are not trivial and therefore encoded.

Vecl32{Int32CDP2, Combined:NULL}: Point Index Difference

Point Index Difference is a vector of indices representing the integer value obtained by subtracting point index from vertex index, encoded using Combined Predictor Type. Point Index Difference is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

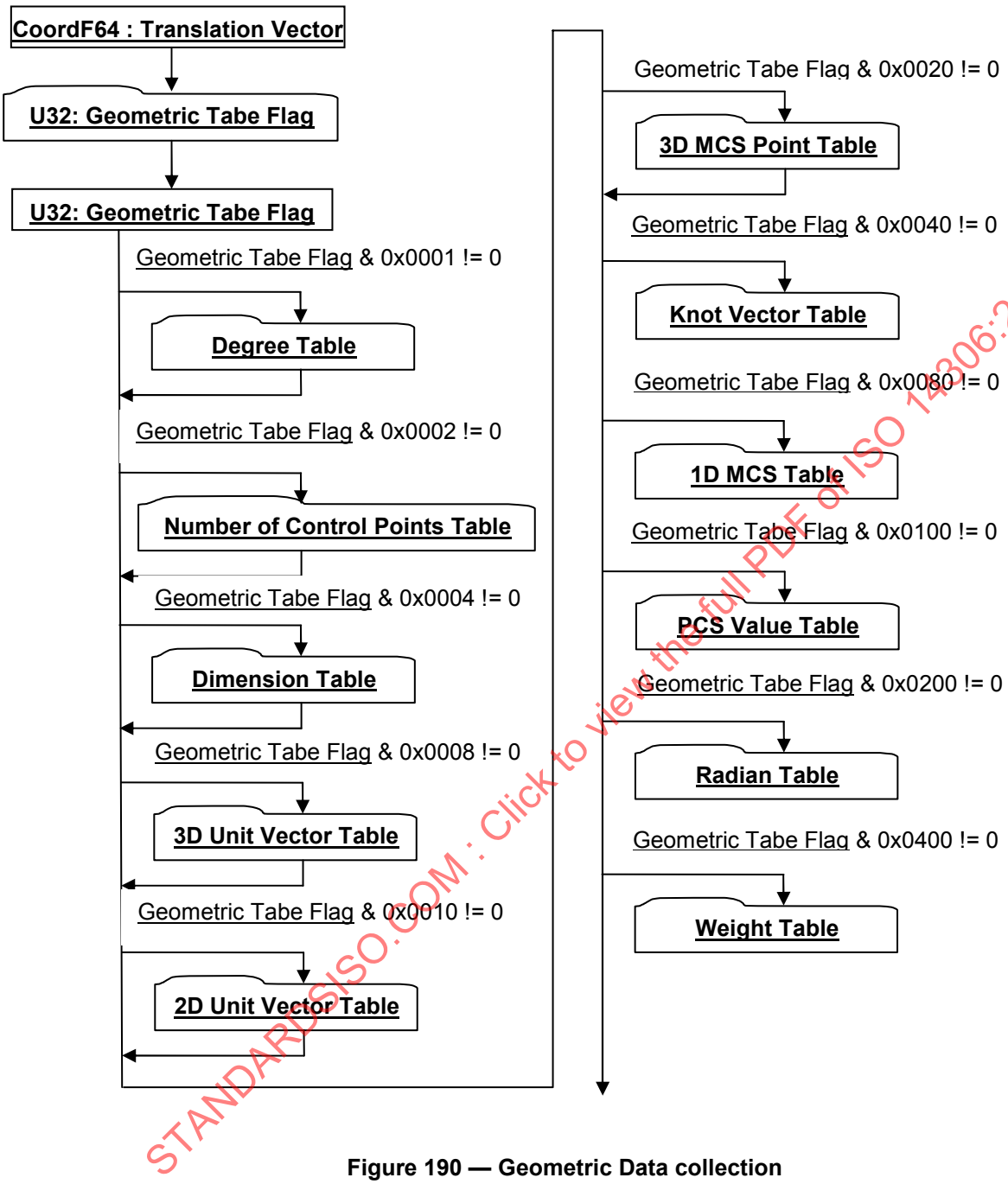
Recover Point Indices

If Vertex Array Flag & 0x01 is equal to 0, then the point index of each vertex is equal to the vertex index. Otherwise, the point index of vertex k is recovered by substrating element k of Point Index Difference array from k, the vertex index.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

12.1.2 Geometric Data



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 190 — Geometric Data collection

CoordF64 : Translation Vector

Translation Vector is a 3-dimensional vector that represents how the ULP geometry is defined w.r.t. the original B-Rep definition from which ULP geometry is derived. If the Translation Vector is not zero vector, then the ULP geometry read from disk is translated from original B-Rep definition by the amount of Translation Vector. This is usually done by the JT writer implementation to improve numerical accuracy of floating point numbers in the ULP geometry. It is important for all the JT readers to take this Translation Vector into consideration when consuming ULP geometry. For example if a LOD is generated from ULP geometry, e.g. by tessellation, then the LOD geometry shall be translated to undo the effect of Translation Vector for it be consistent with the original B-Rep definition. In other words, if we denote the Translation Vector as $\mathbf{v}$, then the LOD geometry from ULP shall be translated by $-\mathbf{v}$.

© ISO 2012 – All rights reserved

U32: Geometric Tabe Flag

Geometric Tabe Flag indicates which geometric tables are not trivial and therefore encoded.

Geometric Entity Counts

Geometric Entity Counts data collection defines the counts for each of the various geometric entities within a ULP.

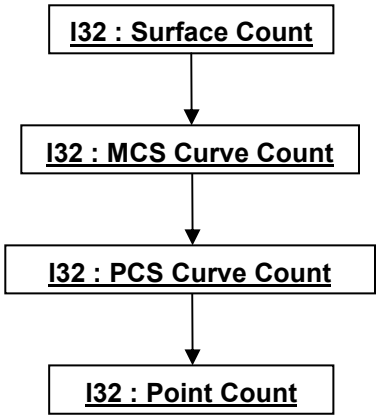


Figure 191 — Geometric Entity Counts

I32 : Surface Count

Surface Count indicates the number of distinct geometric surface entities in the ULP

I32 : MCS Curve Count

MCS Curve Count indicates the number of distinct geometric (Model Coordinate Space) curves (i.e. XYZ curve) entities in the ULP.

I32 : PCS Curve Count

PCS Curve Count indicates the number of distinct geometric Parameter Coordinate Space curves (i.e. UV curve) entities in the ULP

I32 : Point Count

Point Count indicates the number of distinct geometric point entities in the ULP.

Degree Table

Degree Table stores a vector of integers that represent the degree information of Nurbs surfaces and/or curves. If the ULP does not contain any Nurbs entity, then the table is empty and bit 0x0001 in Geometric Tabe Flag is set to be 0.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

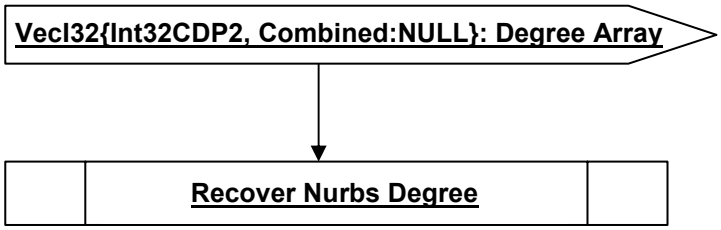


Figure 192 — Degree Table data collection

Vecl32{Int32CDP2, Combined:NULL}: Degree Array

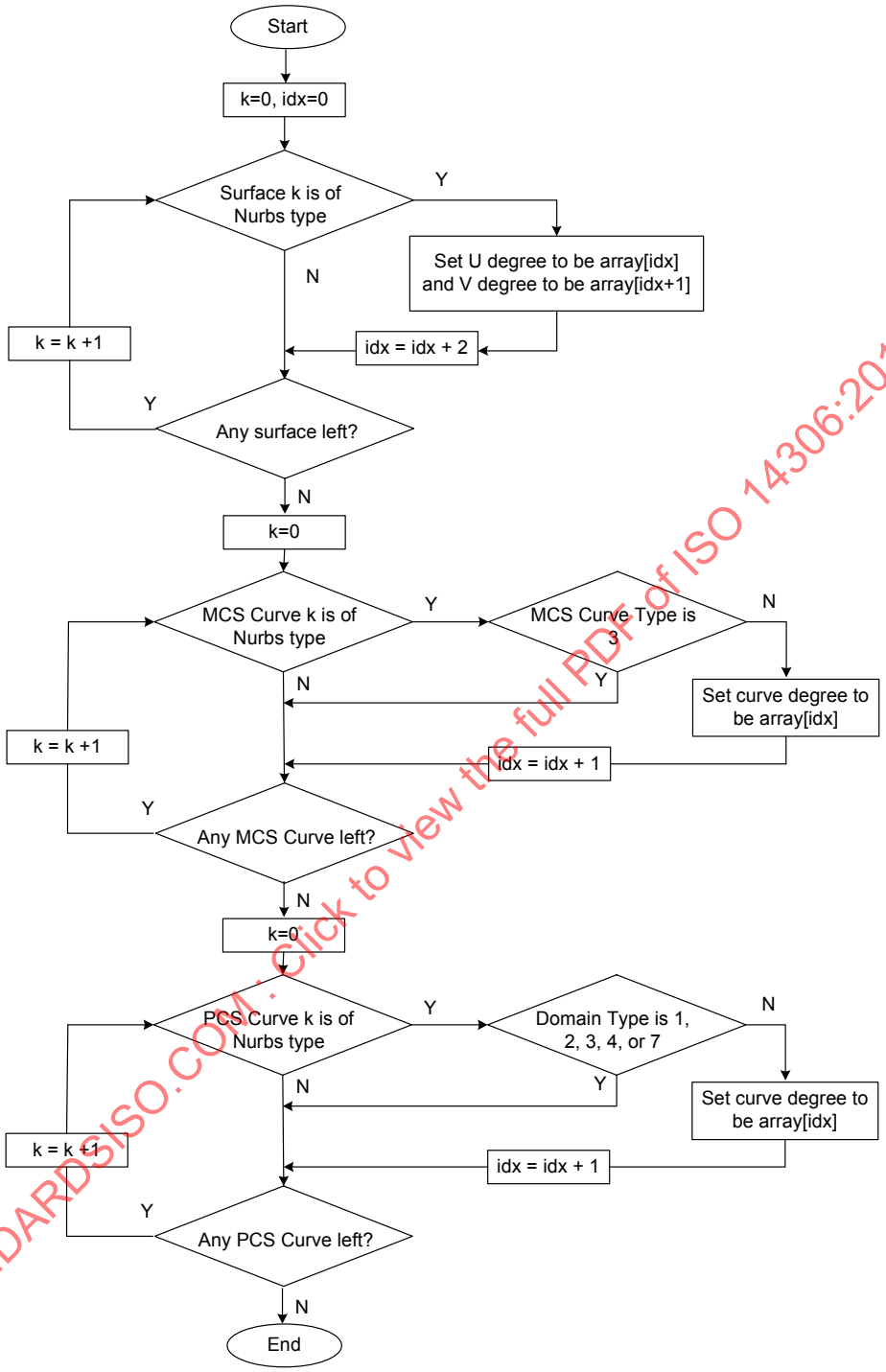
Degree Array is a vector of integers that stores the degree information for all the Nurbs entities in the ULP, encoded using Combined Predictor Type. Degree Array is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover Nurbs Degree

The logic diagram to recover degree information for all the Nurbs entities in the ULP from the Degree Array is shown below.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 193 — Recover Nurbs Degree

Number of Control Points Table

Number of Control Points Table stores a vector of integers that represent the number of control points information of Nurbs surfaces and/or curves. If the ULP does not contain any Nurbs entity, then the table is empty and bit 0x0002 in Geometric Tabe Flag is set to be 0.

© ISO 2012 – All rights reserved

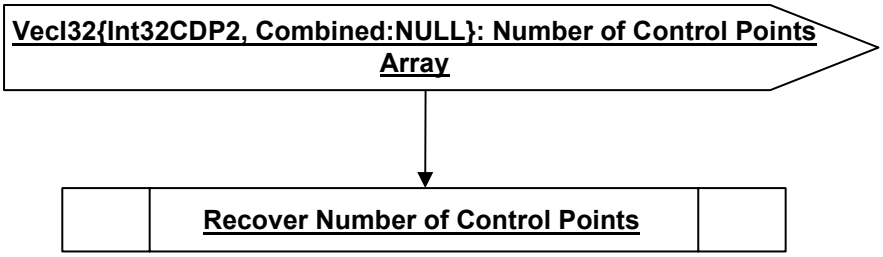


Figure 194 — Number of Control Points Table data collection

Vecl32{Int32CDP2, Combined:NULL}: Number of Control Points Array

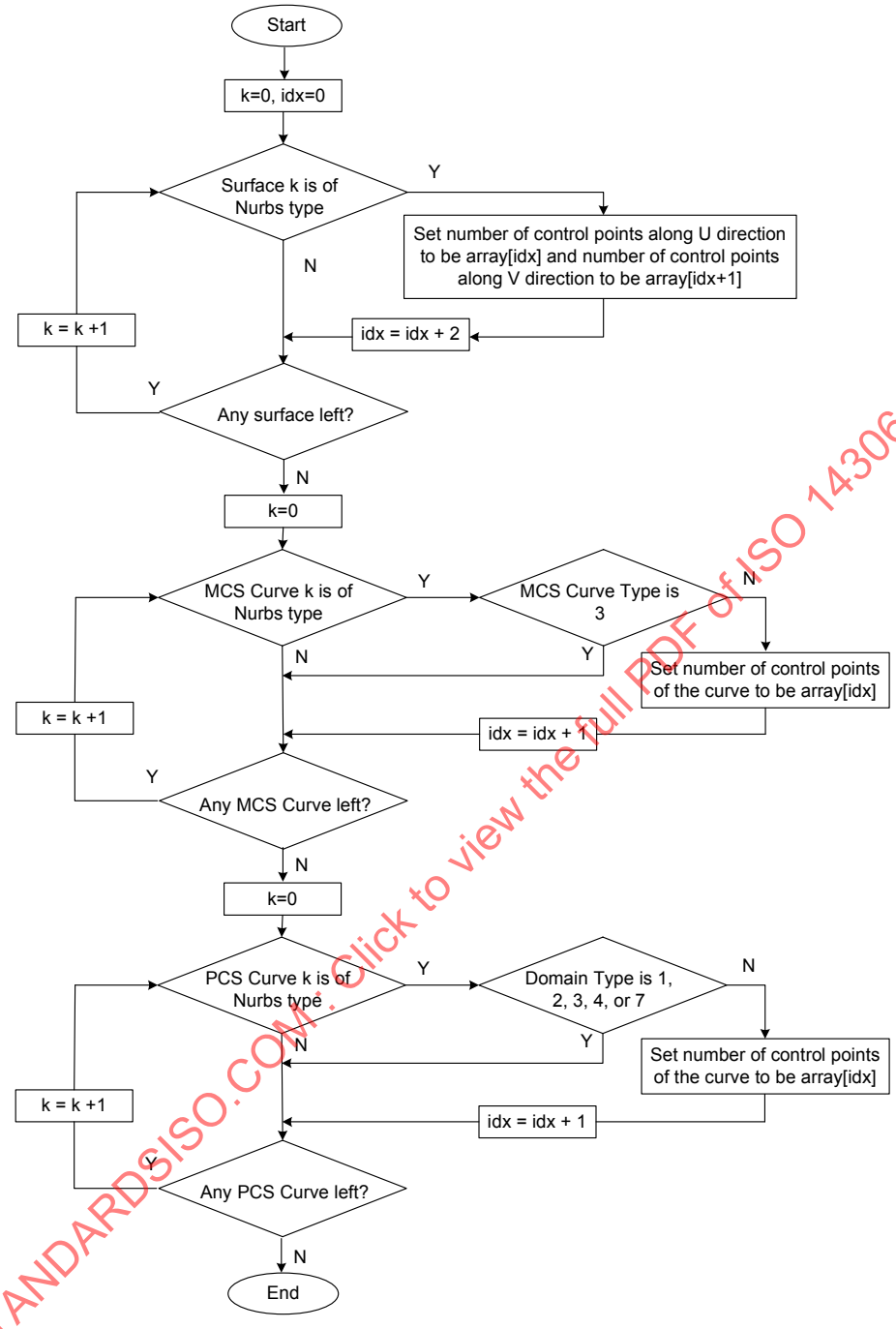
Number of Control Points Array is a vector of integers that stores the number of control points information for all the Nurbs entities in the ULP, encoded using Combined Predictor Type. Number of Control Points Array is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover Number of Control Points

The logic diagram to recover number of control points information for all the Nurbs entities in the ULP from the Number of Control Points Array is shown below.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 195 — Recover Number of Control Points

Dimension Table

Dimension Table stores a vector of integers that represent the dimension information of Nurbs surfaces and/or curves. If the ULP does not contain any Nurbs entity, then the table is empty and bit 0x0004 in Geometric Tabe Flag is set to be 0.

© ISO 2012 – All rights reserved

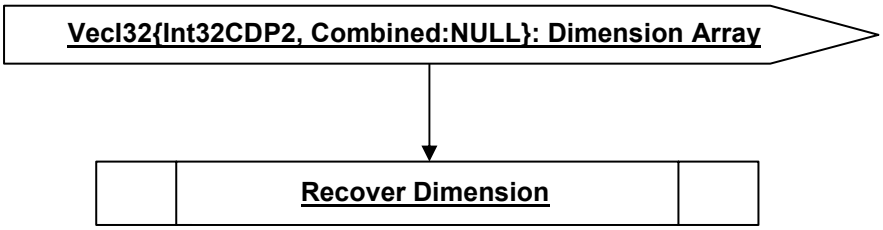


Figure 196 — Dimension Table data collection

Vecl32{Int32CDP2, Combined:NULL}: Dimension Array

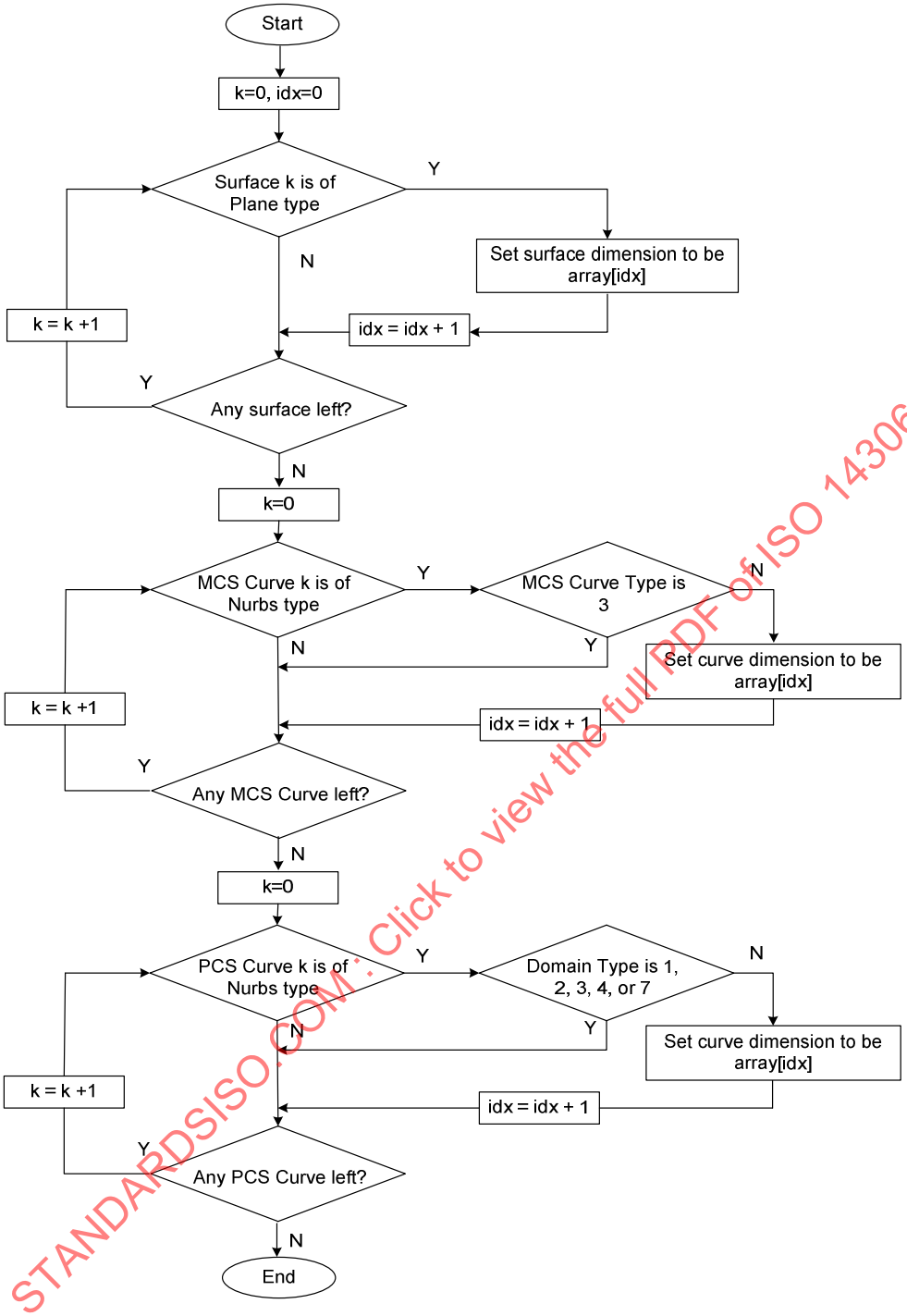
Dimension Array is a vector of integers that stores the dimension information for all the Nurbs entities in the ULP, encoded using Combined Predictor Type. Dimension Array is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC.

Recover Dimension

The logic diagram to recover dimension information for all the Nurbs entities in the ULP from the Dimension Array is shown below.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 197 — Recover Dimension

3D Unit Vector Table

3D Unit Vector Table stores an array of unit vectors in 3D that form part of the analytic surface or curve representation in ULP. If the ULP does not contain any analytic entity, then the table is empty and bit 0x0008 in Geometric Tabe Flag is set to be 0. The supported analytic surface types include plane, cylinder, cone, sphere, and torus, and the supported analytic curve types include line and circle for both parameter space and model space curves. The analytic representation of ULP follows XT convention as detailed in Annex E.

© ISO 2012 – All rights reserved

Similar to the coding of 13.1.5 Compressed Vertex Normal Array, each 3D unit vector is encoded as a single 32 bit integer using 13.2.4 Deering Normal CODEC.

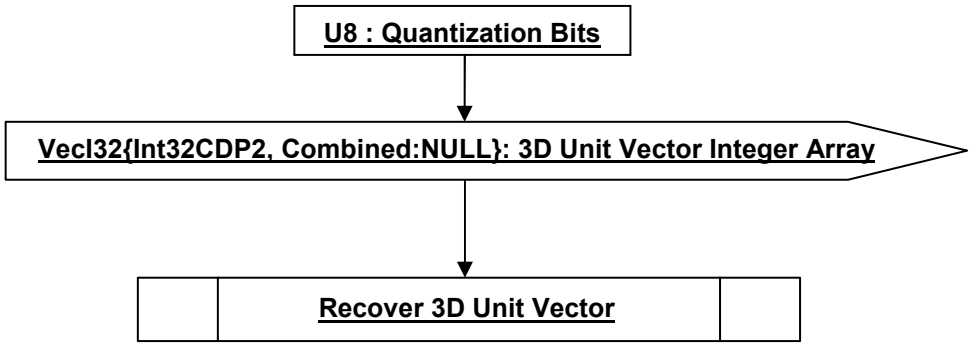


Figure 198 — 3D Unit Vector Table data collection

U8 : Quantization Bits

The number of bits used for the Deering Normal CODEC if quantization is enabled. A value of 0 denotes that quantization is disabled.

Vecl32{Int32CDP2, Combined:NULL}: 3D Unit Vector Integer Array

3D Unit Vector Integer Array is a vector of integers that stores the encoded 3D unit vector from all analytic entities in the ULP, encoded using Combined Predictor Type. 3D Unit Vector Integer Array is compressed and encoded using the 13.1.2 Int32 Compressed Data Packet Mk 2 version of second generation CODEC.

Recover 3D Unit Vector

The logic diagram to recover 3D unit vector information for all the analytic entities in the ULP from the 3D Unit Vector Integer Array is shown below.

The recovery of a unit vector from an element in the 3D Unit Vector Integer Array is done as part of Deering Normal CODEC.

As described in Annex E, the representation of an analytic surface of types plane, cylinder, cone, sphere, or torus, includes two 3D unit vectors. One is called “axis” and the other is called “x_axis”. These two unit vectors of each analytic surface are recovered for each analytic surface. In addition, the “normal” vector to the plane containing a 3D circle is also recovered.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

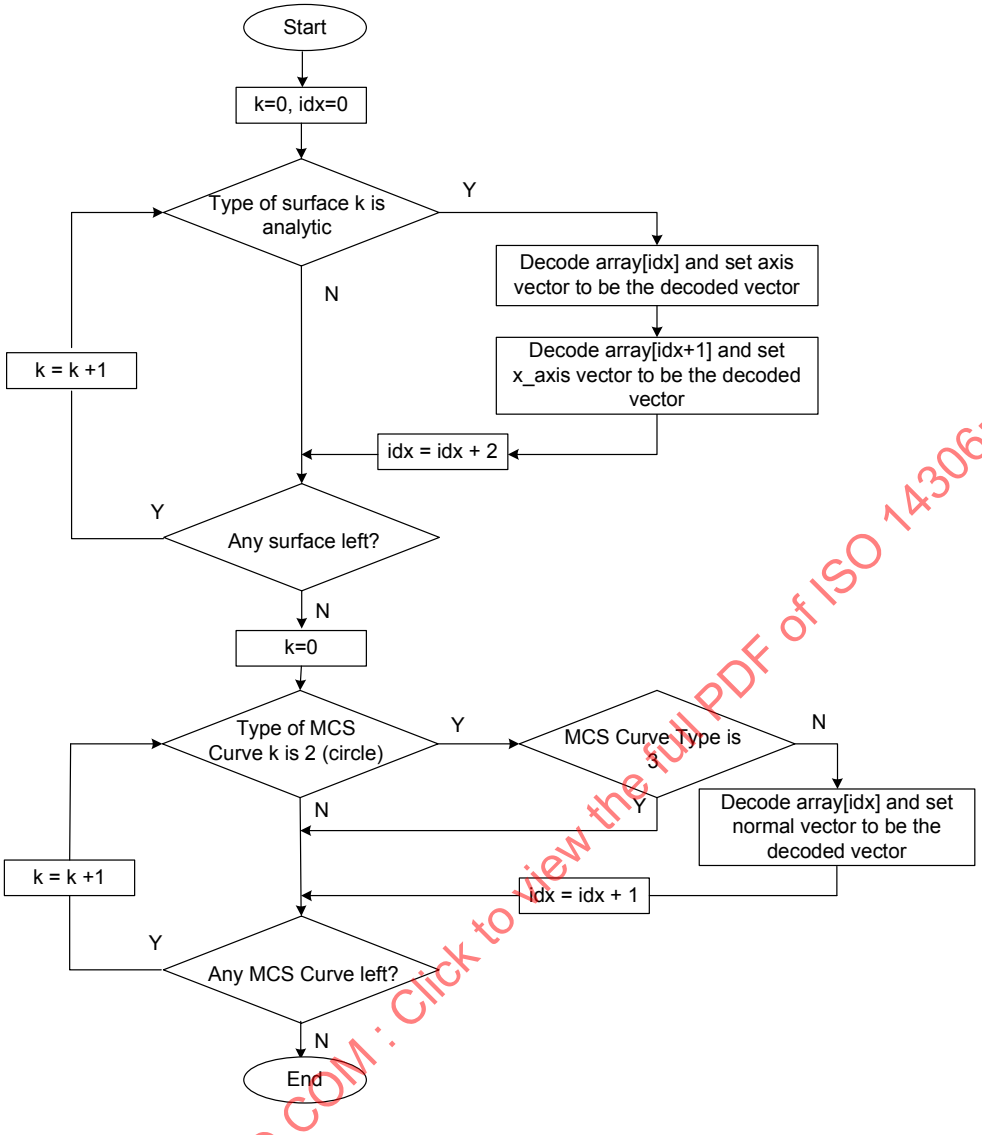


Figure 199 — Recover Dimension

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

2D Unit Vector Table

2D Unit Vector Table stores an array of unit vectors in 2D that form part of PCS analytic circle representation in ULP. If the ULP does not contain any analytic circle in the PCS, then the table is empty and bit 0x0010 in Geometric Tabe Flag is set to be 0. The analytic curve representation of ULP follows XTconvention as detailed in Annex E.

Similar to the coding of 13.1.5 Compressed Vertex Normal Array, each 2D unit vector is treated as a 3D unit vector with z component set to be 0.0, and encoded as a single 32 bit integer using 13.2.4 Deering Normal CODEC. In addition, the Quantization Bits information of Deering Normal CODEC used to encode 2D Unit Vector Table is always the same as the one used for 3D Unit Vector Table.

© ISO 2012 – All rights reserved

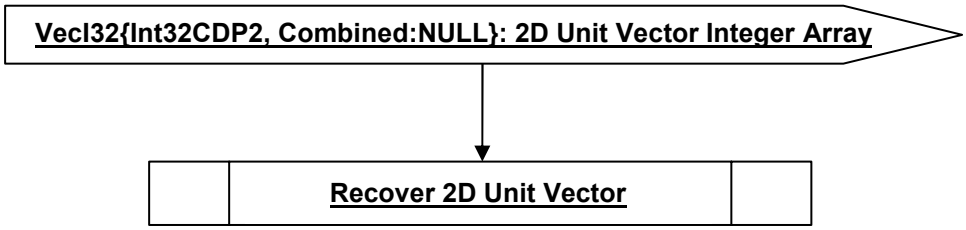


Figure 200 — 2D Unit Vector Table data collection

VecI32{Int32CDP2, Combined:NULL}: 2D Unit Vector Integer Array

2D Unit Vector Integer Array is a vector of integers that stores the encoded 2D unit vector from all analytic entities in the ULP.

Recover 2D Unit Vector

The logic diagram to recover 2D unit vector information for all the analytic entities in the ULP from the 2D Unit Vector Integer Array is shown below.

The recovery of a unit vector from an element in the 2D Unit Vector Integer Array is done as part of Deering Normal CODEC. The Quantization Bits read from 3D Unit Vector Table should be used for Deering Normal CODEC to decode the vector information from each element in 2D Unit Vector Integer Array.

The "x_axis" vector to the circle in the PCS, as described in Annex E, is recovered.

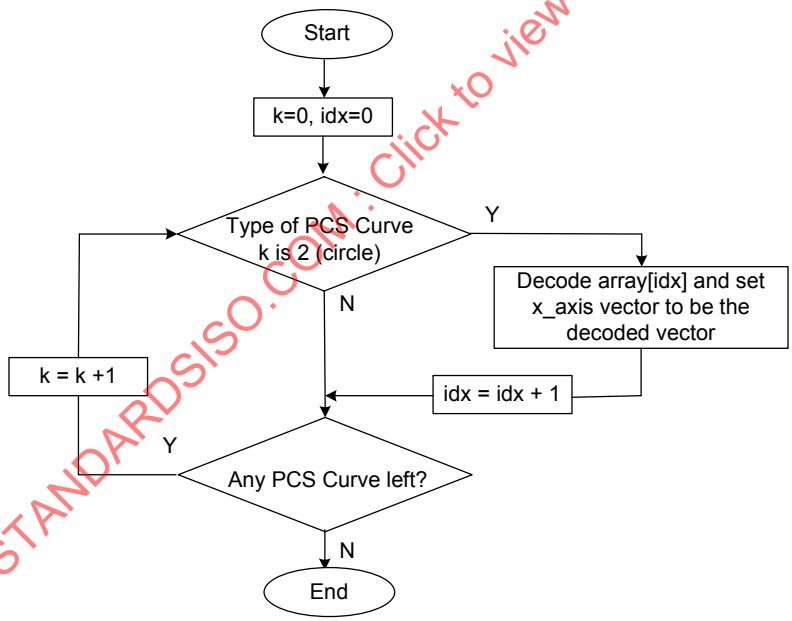


Figure 201 — Recover 2D Unit Vector

3D MCS Point Table

3D MCS Point Table stores the quantization representation of an array of 3D MCS points in ULP. If the ULP does not contain 3D MCS points, then the table is empty and bit 0x0020 in Geometric Tabe Flag is set to be 0.

Each point coordinate is first encoded into an integer with uniform quantizer (see 13.1.12 Uniform Quantizer Data) and then all the integers from each coordinate are grouped into an integer array, which is then encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC with Combined Predictor Type.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

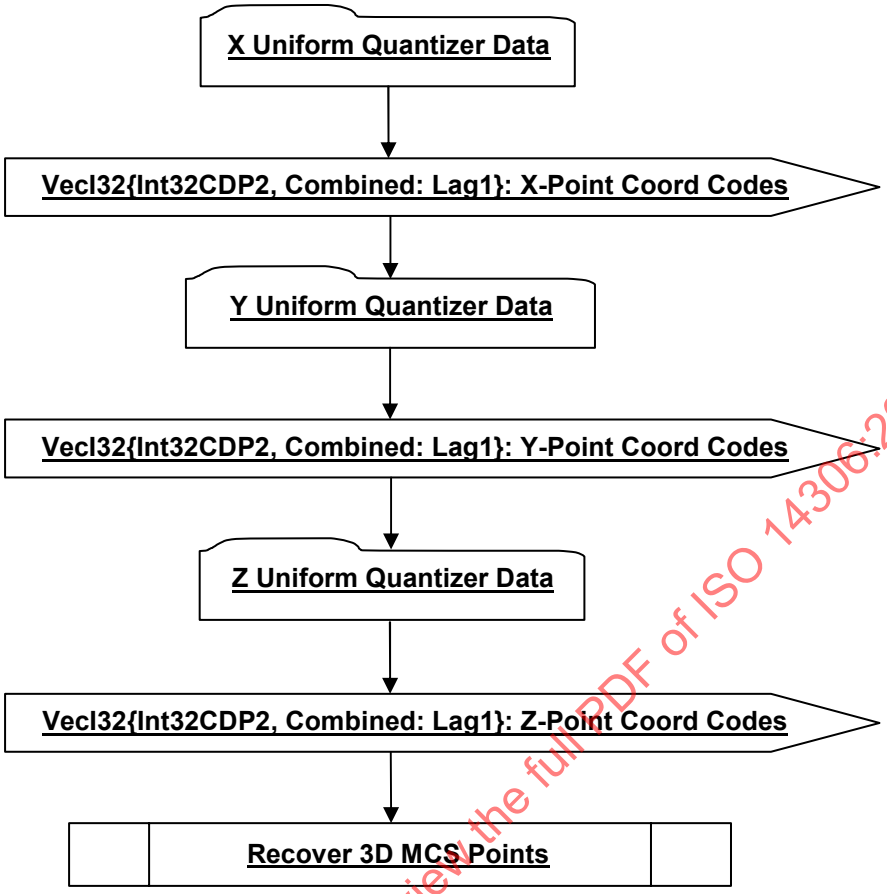


Figure 202 — 3D MCS Point Table data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Vecl32{Int32CDP2, Combined: Lag1}: X-Point Coord Codes

X-Point Coord Codes is a vector of quantizer “codes” for all the X-components of an array of point coordinates. X-Point Coord Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC to compress and encode data.

Vecl32{Int32CDP2, Combined: Lag1}: Y-Point Coord Codes

Y-Point Coord Codes is a vector of quantizer “codes” for all the Y-components of an array of point coordinates. Y-Point Coord Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC to compress and encode data.

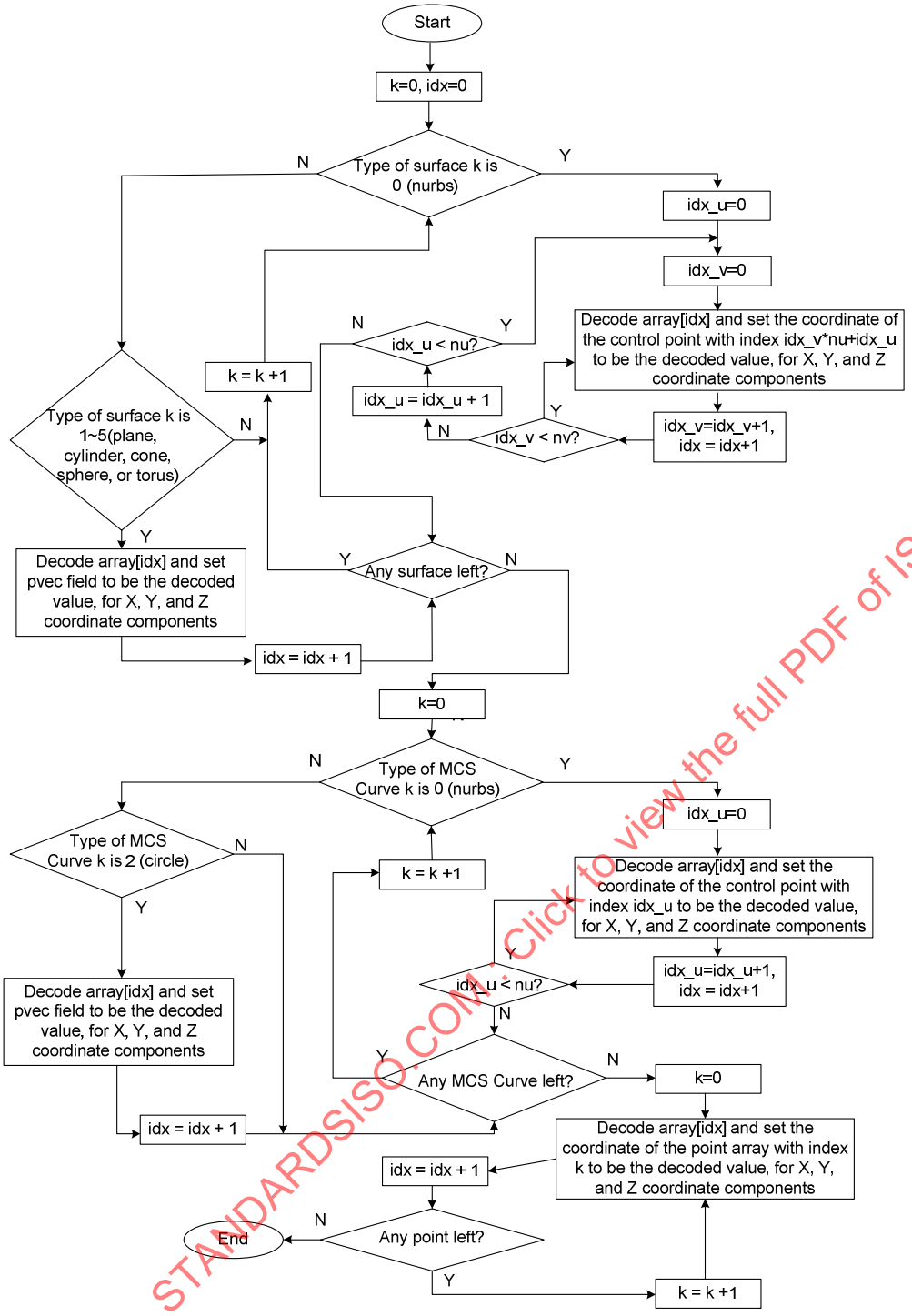
Vecl32{Int32CDP2, Combined: Lag1}: Z-Point Coord Codes

Z-Point Coord Codes is a vector of quantizer “codes” for all the Z-components of an array of point coordinates. Z-Point Coord Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC to compress and encode data.

Recover 3D MCS Points

The logic diagram to recover 3D MCS points information in the ULP from the three arrays, X-Point Coord Codes, Y-Point Coord Codes, and Z-Point Coord Codes, is shown below. Note that the point coordinates are decoded from the integer elements with Uniform Quantizer (see 13.1.12 Uniform Quantizer Data).

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 203 — Recover 3D MCS Points

Knot Vector Table

Knot Vector Table stores the quantization representation of knot vectors in ULP. If the ULP does not contain any knot vector that needs be stored, then the table is empty and bit 0x0040 in Geometric Tabe Flag is set to be 0.

In ULP every knot vector starts with 0.0 and ends with 1.0 and is always clamped at both ends. The encoding of knot vector depends on its classified knot type. The knot values in the middle of a knot vector need be written only if the knot type is 0 (see Supported Knot Type). For all the knot values that need be written, each of them is encoded into an integer with uniform quantizer (see 13.1.12 Uniform Quantizer Data) and then all

© ISO 2012 – All rights reserved

the integers are grouped into an integer array. The integer array is then encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC with Combined Predictor Type.

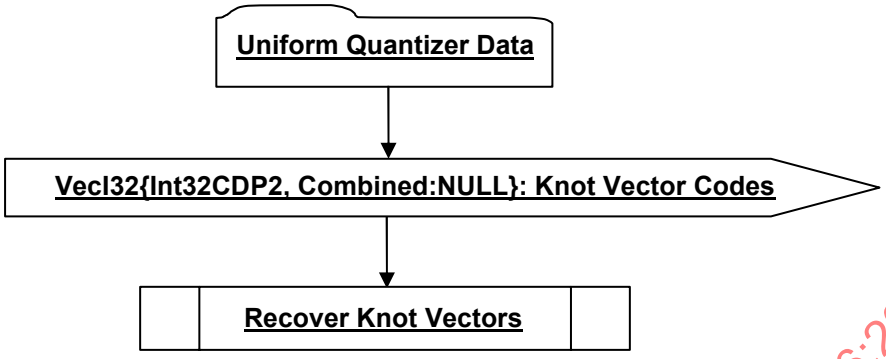


Figure 204 — Knot Vector Table data collection

Vecl32{Int32CDP2, Combined:NULL}: Knot Vector Codes

Knot Vector Codes is a vector of quantizer "codes" for all the knot vectors. Knot Vector Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC with Combined Predictor Type to compress and encode data.

Recover Knot Vectors

The logic diagram to recover knot vector information in the ULP from the Knot Vector Codes is shown below. Note that each integer element in the Knot Vector Codes array is decoded with Uniform Quantizer.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

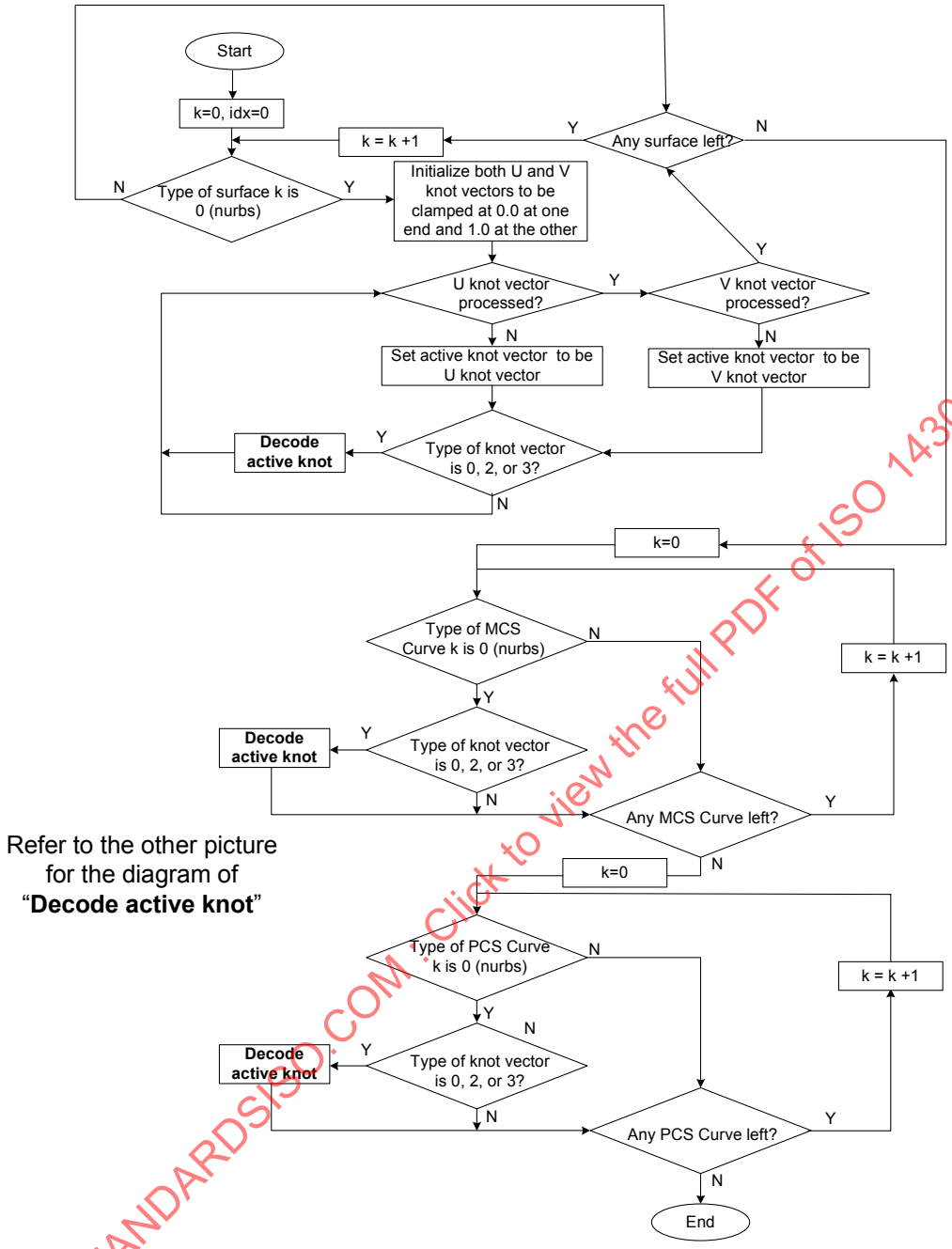


Figure 205 — Recover Knot Vectors

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

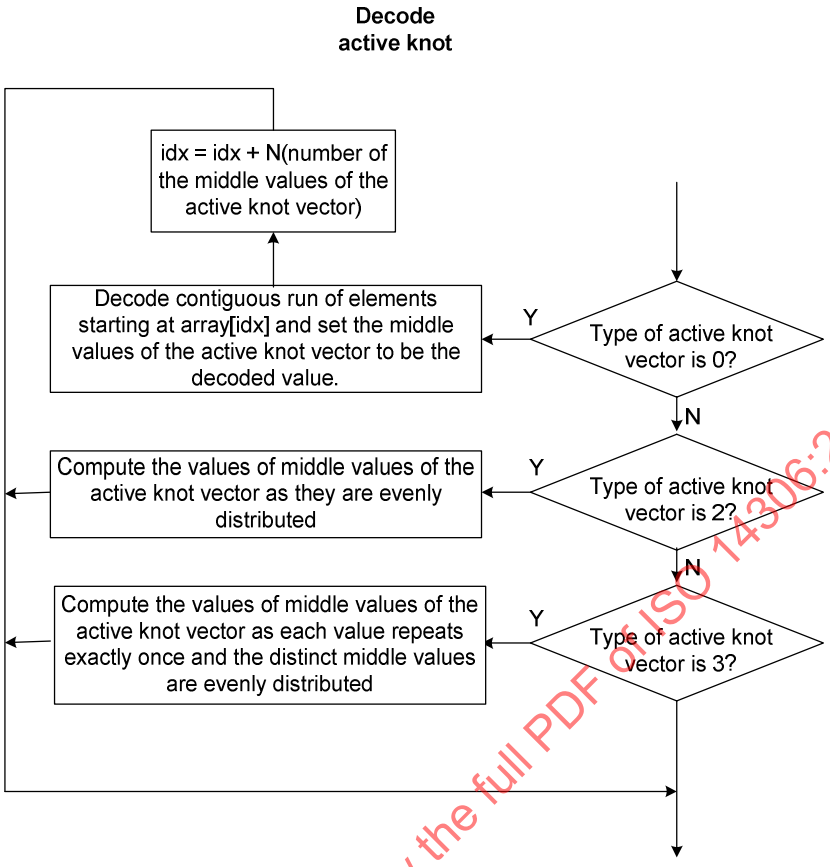


Figure 205 — Recover Knot Vectors

(continued)

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

1D MCS Table

1D MCS Table stores the quantization representation of floating point values in MCS. If the ULP does not contain any such value, then the table is empty and bit 0x0080 in Geometric Tabe Flag is set to be 0. Each floating point value is encoded into an integer with uniform quantizer (see 13.1.12 Uniform Quantizer Data) and then all the integers are grouped into an integer array. The integer array is then encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC with Combined Predictor Type.

© ISO 2012 – All rights reserved

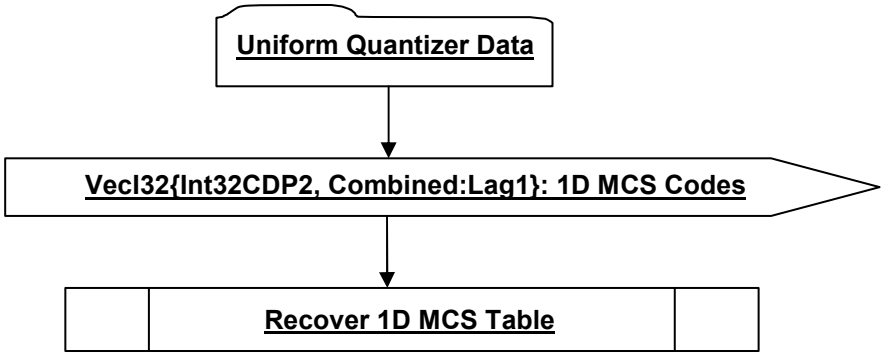


Figure 206 — 1D MCS Table data collection

Vecl32{Int32CDP2, Combined:Lag1}: 1D MCS Codes

1D MCS Codes is a vector of quantizer "codes" for all the 1D floating point values in MCS . 1D MCS Codes uses the Int32 version of the second generation CODEC with Combined Predictor Type to compress and encode data.

Recover 1D MCS Table

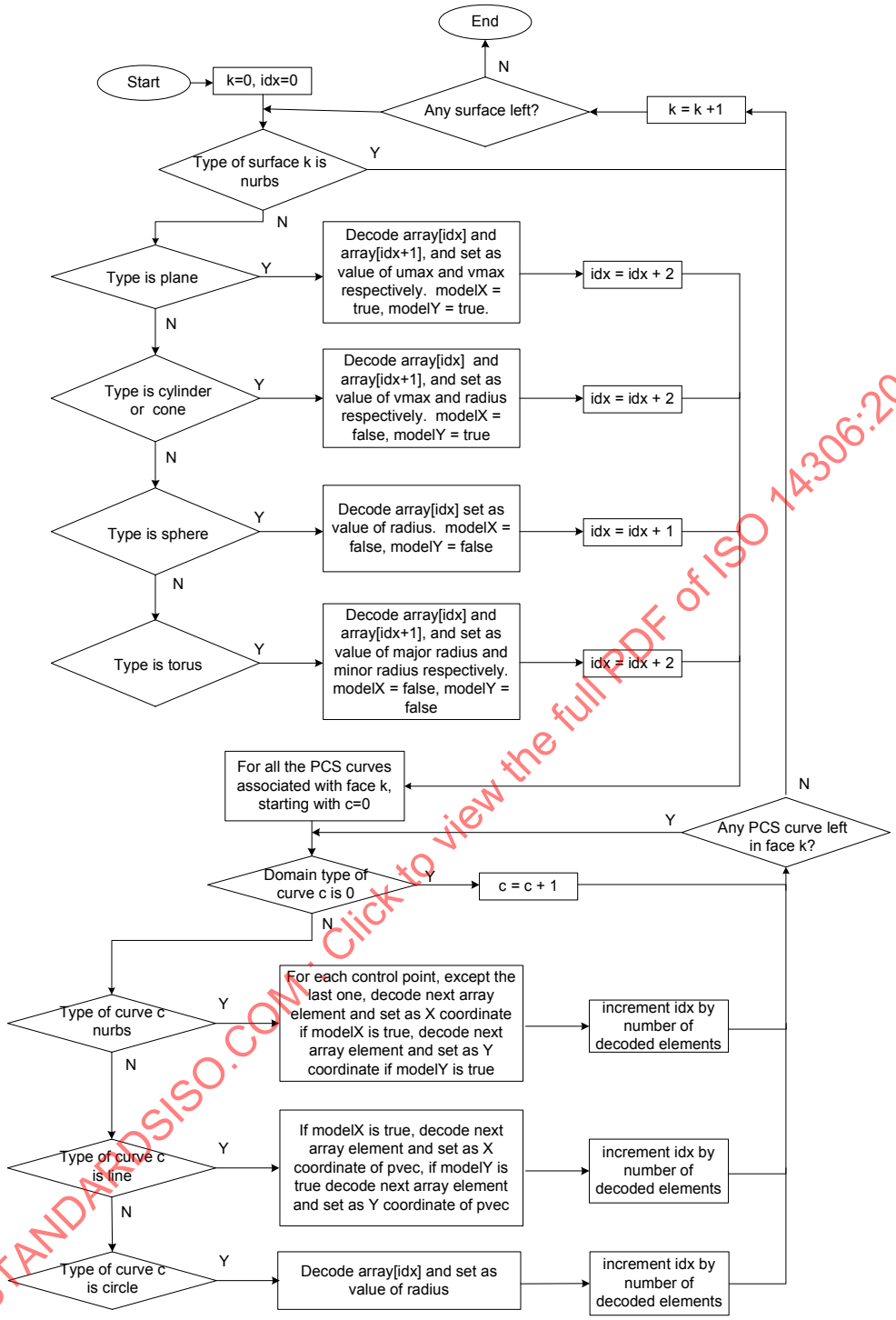
The representation of each surface or curve in ULP includes information that describes the extent of the surface or curve in the parameter domain. For curves the extent information is represented by two numbers, umin and umax, while for surfaces it is represented by two additional numbers for the other parametric direction, vmin and vmax. For surfaces or curves of NURBS type such extent information is implied by the knot vector information. For surfaces or curves of other types the extent information needs be read from 1D MCS Table if the parameter value represents value in MCS, or Radian Table if the parameter value represents angle information. The detailed information about how the parameter domain information of different entities should be read is listed in Table 100.

Table 100 — Parameter Domain

Entity Type	umin	umax	vmin	vmax
NURBS Surface	n/a (from knot)	n/a (from knot))	n/a (from knot)	n/a (from knot)
Plane	n/a (always 0)	1D MCS Table	n/a (always 0)	1D MCS Table
Cylinder	n/a (always 0)	Radian Table	n/a (always 0)	1D MCS Table
Cone	n/a (always 0)	Radian Table	n/a (always 0)	1D MCS Table
Sphere	n/a (always 0)	Radian Table	Radian Table	Radian Table
Torus	n/a (always 0)	Radian Table	Radian Table	Radian Table
XYZ NURBS Curve	n/a (from knot)	n/a (from knot)	n/a	n/a
XYZ Line	n/a (always 0)	n/a (from vertex geometry)	n/a	n/a
XYZ Circle	n/a (always 0)	Radian Table	n/a	n/a
UV NURBS Curve	n/a (from knot)	n/a (from knot)	n/a	n/a
UV Line	n/a (always 0)	n/a (from next uv curve)	n/a	n/a
UV Circle	Radian Table	Radian Table	n/a	n/a

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 207 — Recover 1D MCS Table

The logic diagram to recover 1D MCS table information in the ULP from the 1D MCS Codes is shown in igure 206 — 1D MCS Table data collectionFigure 207. Note that each integer element in the 1D MCS Codes array is decoded with Uniform Quantizer.

PCS Value Table

PCS Value Table stores the quantization representation of floating point values in PCS. If the ULP does not contain any such value, then the table is empty and bit 0x0100 in Geometric Tabe Flag is set to be 0. Each floating point value is encoded into an integer with uniform quantizer (see 13.1.12 Uniform Quantizer Data)

© ISO 2012 – All rights reserved

and then all the integers are grouped into an integer array. The integer array is then encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC with Combined Predictor Type.

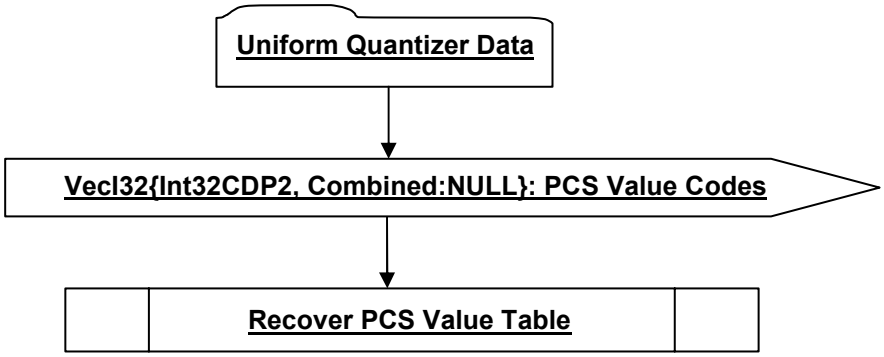


Figure 208 — PCS Value Table data collection

Vecl32{Int32CDP2, Combined:NULL}: PCS Value Codes

PCS Value Codes is a vector of quantizer "codes" for all the floating point values in PCS . PCS Value Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC with Combined Predictor Type to compress and encode data.

Recover PCS Value Table

The logic diagram to recover PCS Value Table information in the ULP from the PCS Value Codes is shown in Figure 209. Note that each integer element in the PCS Value Codes array is decoded with Uniform Quantizer.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

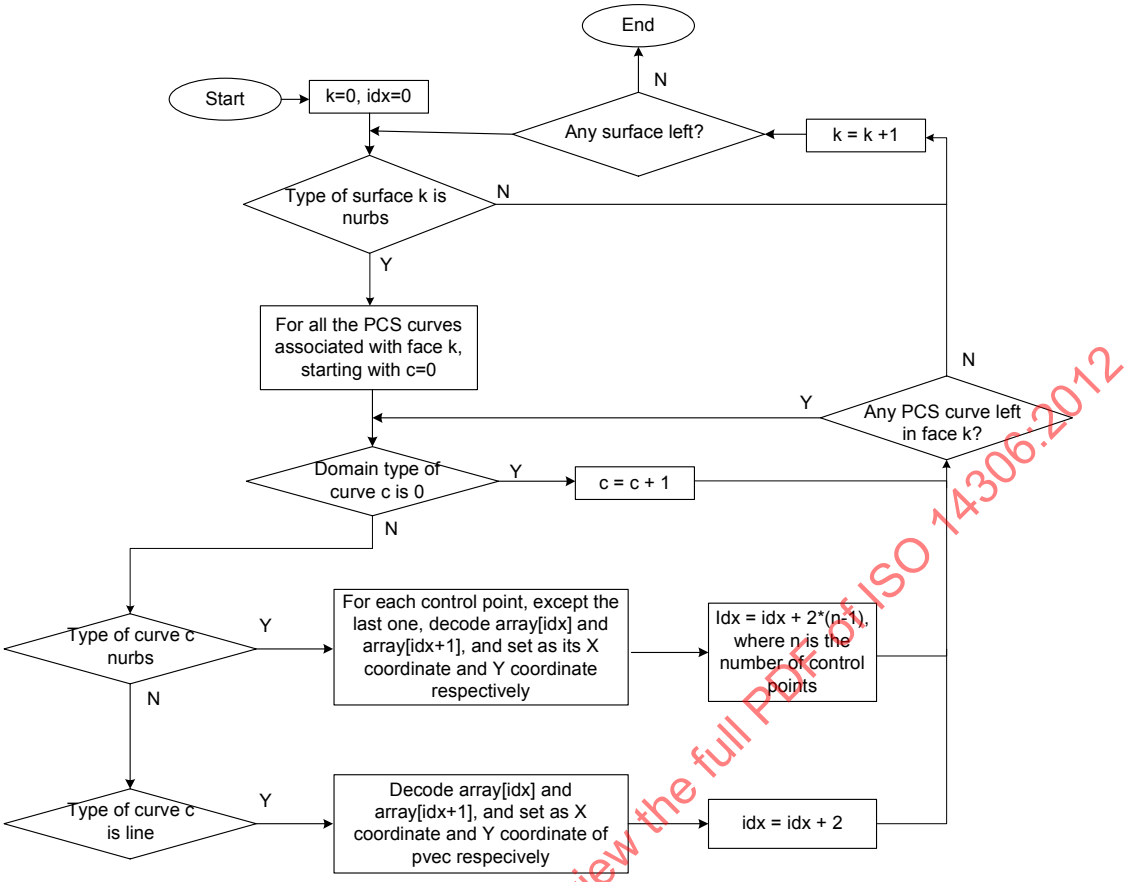


Figure 209 — Recover PCS Value Table

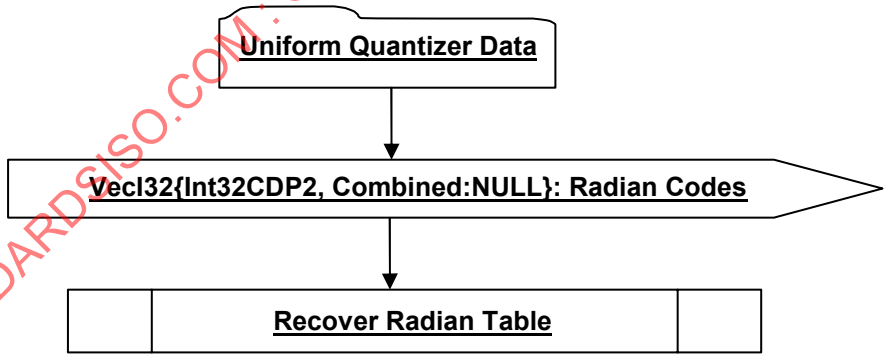


Figure 210 — Radian Table data collection

Radian Table

Radian Table stores the quantization representation of angular values. If the ULP does not contain any such angular value, then the table is empty and bit 0x0200 in Geometric Tabe Flag is set to be 0. Each angular value is encoded into an integer with uniform quantizer (see 13.1.12 Uniform Quantizer Data) and then all the integers are grouped into an integer array. The integer array is then encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC with Combined Predictor Type.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Vecl32{Int32CDP2, Combined:NULL}: Radian Codes

Radian Codes is a vector of quantizer “codes” for all the angular values. Radian Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC with Combined Predictor Type to compress and encode data.

Recover Radian Table

The logic diagram to recover Radian Table information in the ULP from the Radian Codes is shown in

Figure 211. Note that each integer element in the Radian Codes array is decoded with Uniform Quantizer.

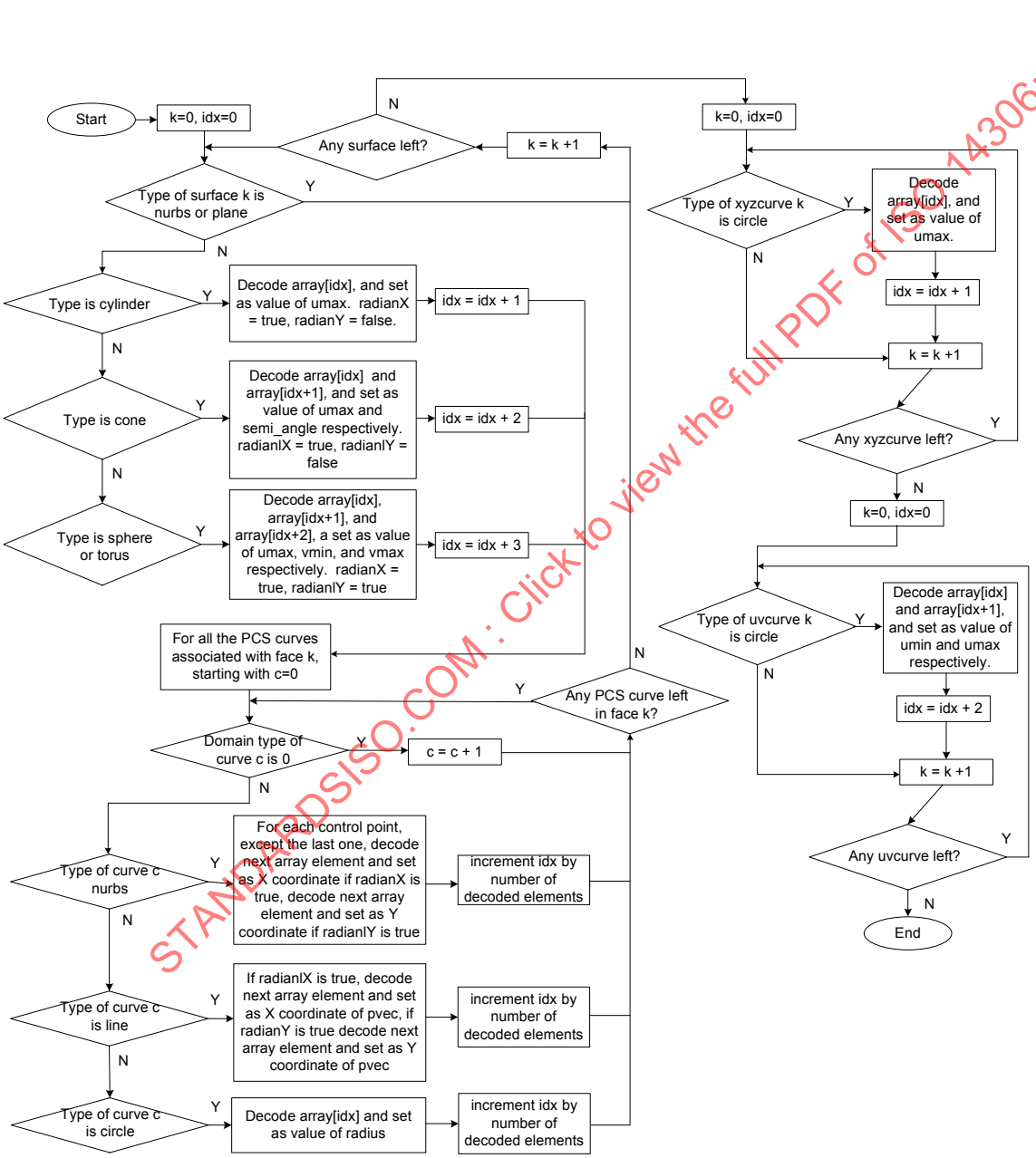


Figure 211 — Recover Radian Table

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

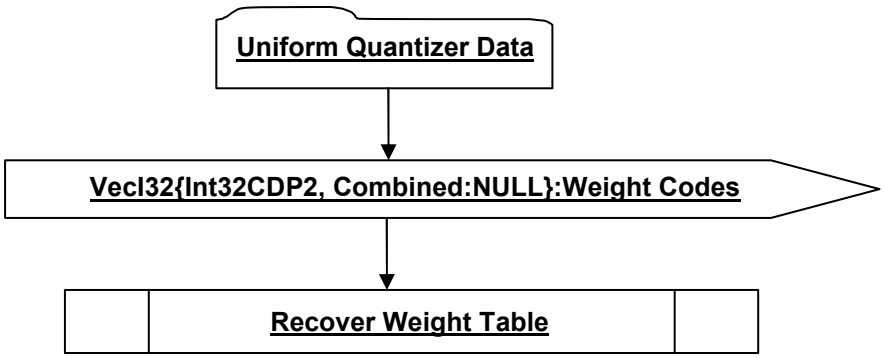


Figure 212 — Weight Table data collection

Weight Table

Weight Table stores the quantization representation of weight values. If the ULP does not contain any such weight value, then the table is empty and bit 0x0400 in Geometric Tabe Flag is set to be 0. Each weight value is encoded into an integer with uniform quantizer (see 13.1.12 Uniform Quantizer Data) and then all the integers are grouped into an integer array. The integer array is then encoded using the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of second generation CODEC with Combined Predictor Type.

VecI32{Int32CDP2, Combined:NULL}:Weight Codes

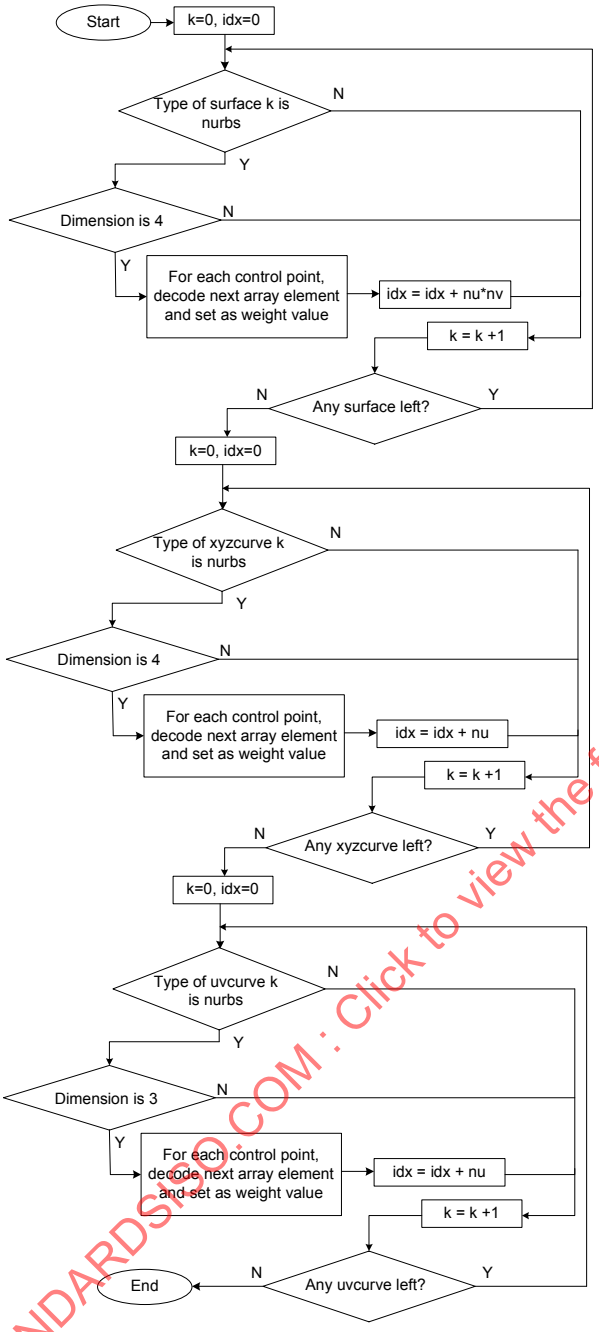
Weight Codes is a vector of quantizer "codes" for all the weight values. Weight Codes uses the 13.1.2 Int32 Compressed Data Packet Mk. 2 version of the second generation CODEC with Combined Predictor Type to compress and encode data.

Recover Weight Table

The logic diagram to recover Weight Table information in the ULP from the Weight Codes is shown in Figure 213. Note that each integer element in the Weight Codes array is decoded with Uniform Quantizer.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 213 — Recover Weight Table

12.1.3 Material Attribute Element Properties

The properties attached to material attribute are standard JT properties, and the logic diagram to read the properties attached a material attribute is shown in Figure 214.

© ISO 2012 – All rights reserved

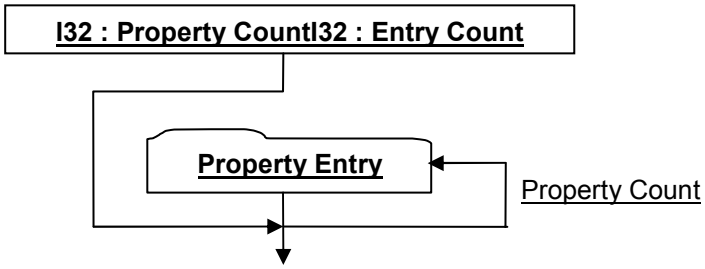


Figure 214 — Material Attribute Element Properties

I32 : Property Count

Property count is the number of properties attached.

Property Entry

Standard JT property entry, consisting of key and value pair.

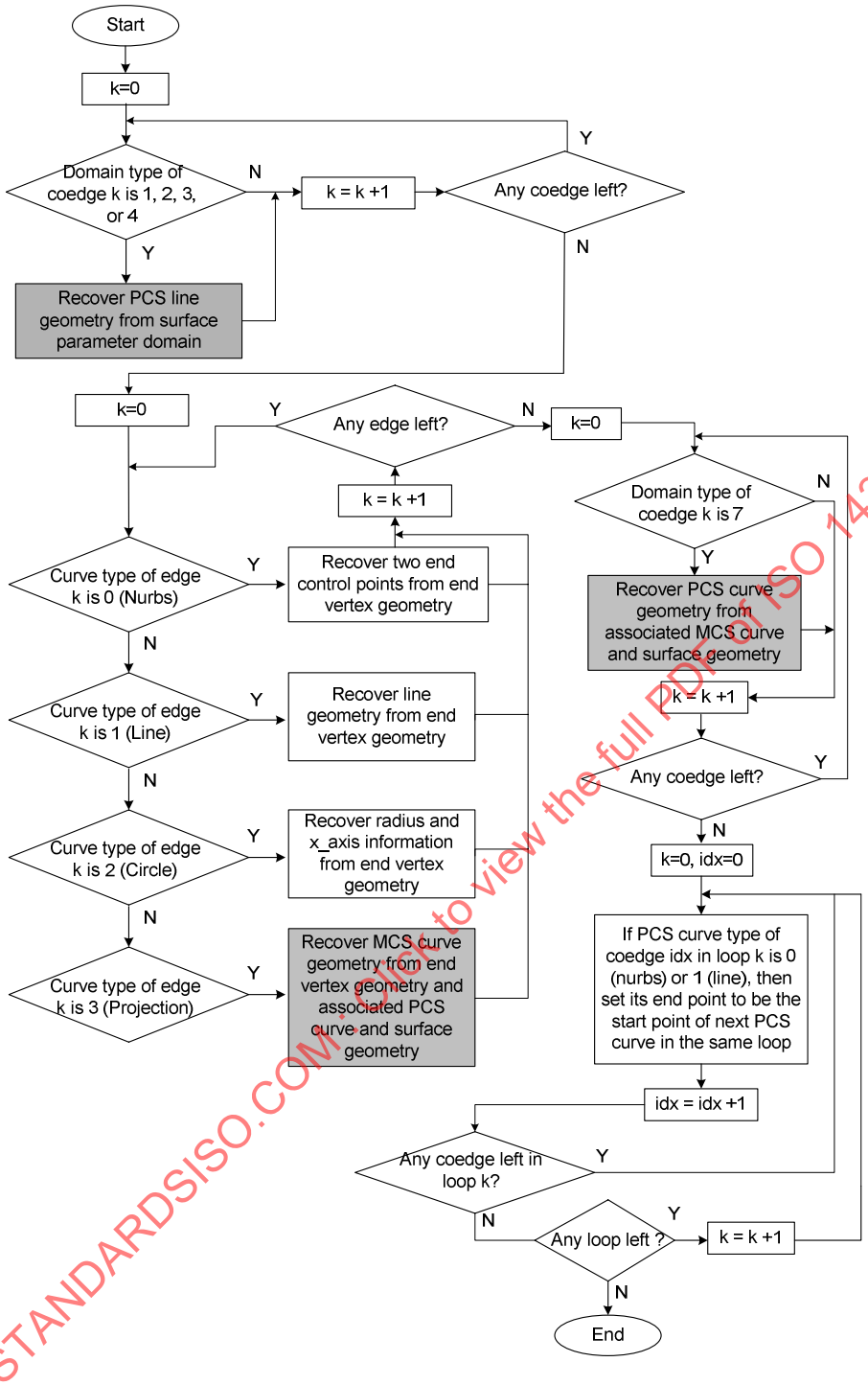
12.1.4 Information Recovery

The information in ULP is classified as “essential information” that is explicitly written on disk, and “derivative information” that can be computed from the “essential information”. How “essential information” of ULP can be read from disk was covered in previous sections, and this section focuses on the logic to recover “derivative information” from “essential information”.

The derivative information consists of curve information either in the parameter or model space. For example, the PCS curves associated with an untrimmed face can be inferred from the parameter domain of the surface, or an MCS curve may be computed from vertex information and/or the combination of corresponding PCS curve geometry and surface geometry, etc.. Shown in Figure 217 is the high level diagram to recover “derivative information”. First, all the PCS line geometry are recovered from the associated surface domain information if the domain type of those PCS curves, stored in its associated coedge, are of value 1, 2, 3, 4 meaning that the PCS curve is identical to one of the parameter boundaries of the surface. Second, the MCS curve geometry is recovered depending on its type. If the MCS curve type is 0, 1, or 2, then the geometry of its two end vertices is used to compute the curve geometry. If the MCS curve type is 3, then its geometry is computed by projecting PCS curve onto the surface geometry. After all the MCS curve geometry is computed, all the PCS curves of type 7 is computed by projecting MCS curve onto the parameter domain. Some part of PCS curve definition may still be missing after all these steps. At the end, all the information that is still missing in some of the PCS curves is recovered by leveraging the knowledge that all PCS curves within the same loop are connected in a head to tail fashion. The logical steps that are displayed with dark colour indicate steps that will be elaborated in more detail later.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 215 — Material Attribute Element Properties

PCS Curve Recovery from Surface Domain

Shown in Figure 216 is the diagram illustrating how the PCS curve geometry is recovered from surface parameter domain information.

© ISO 2012 – All rights reserved

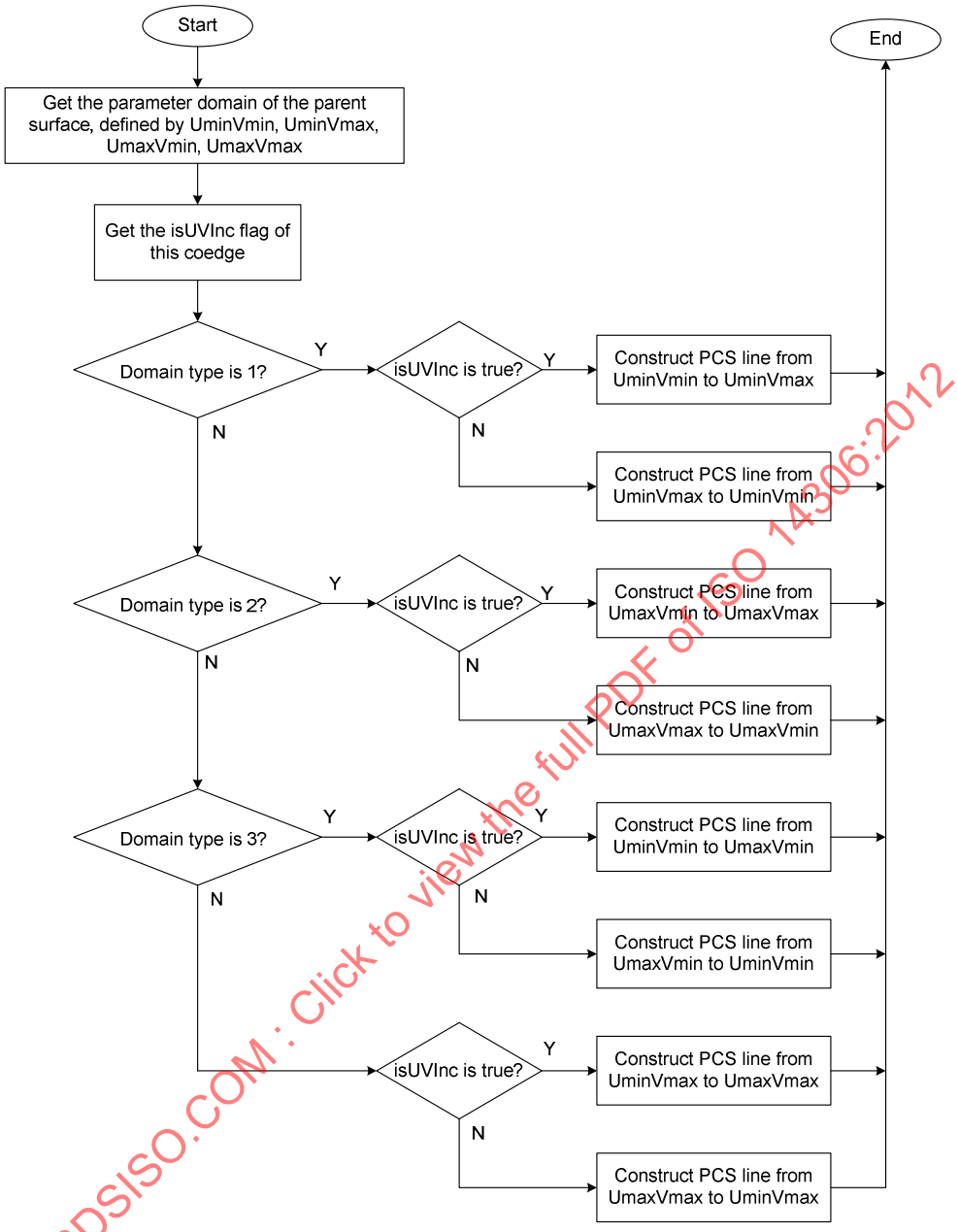


Figure 216 — PCS Curve Recovery from Surface Domain

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

MCS Curve Recovery

Shown in Figure 217 is the diagram illustrating how MCS curve geometry is recovered from its end vertex geometry, and/or its associated PCS curve geometry and surface geometry. If the associated PCS curve is coincident with one of the parameter boundaries of the parent surface, then the MCS curve can be recovered from parent surface geometry. Otherwise, if the surface type is planar and PCS curve is of type NURBS, then the MCS curve geometry can be recovered by projecting the PCS curve from parameter domain to model space onto the planar surface.

© ISO 2012 – All rights reserved

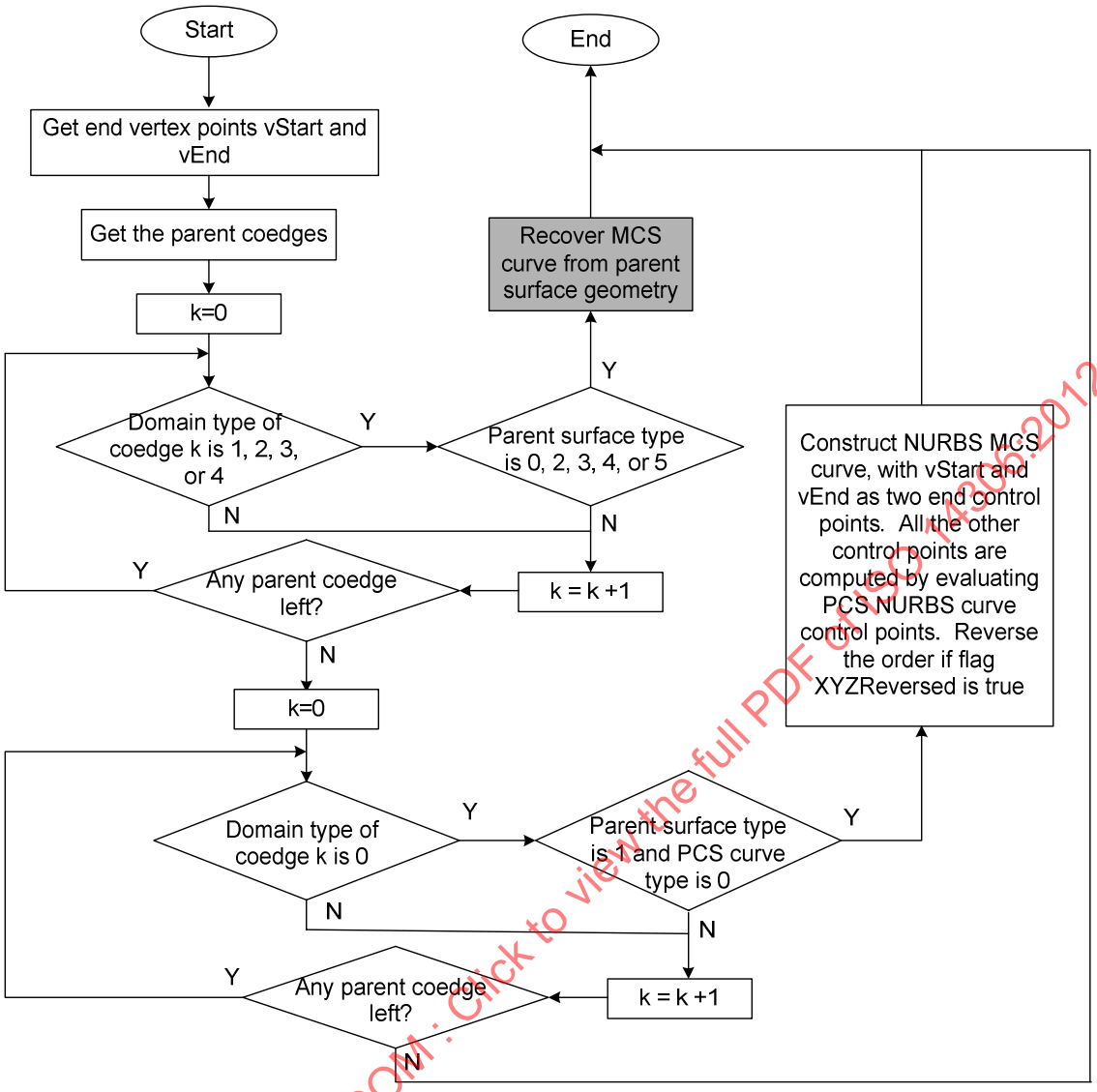


Figure 217 — MCS Curve Recovery

Shown in Figure 218 is the detailed description of how MCS curve can be recovered from surface geometry.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

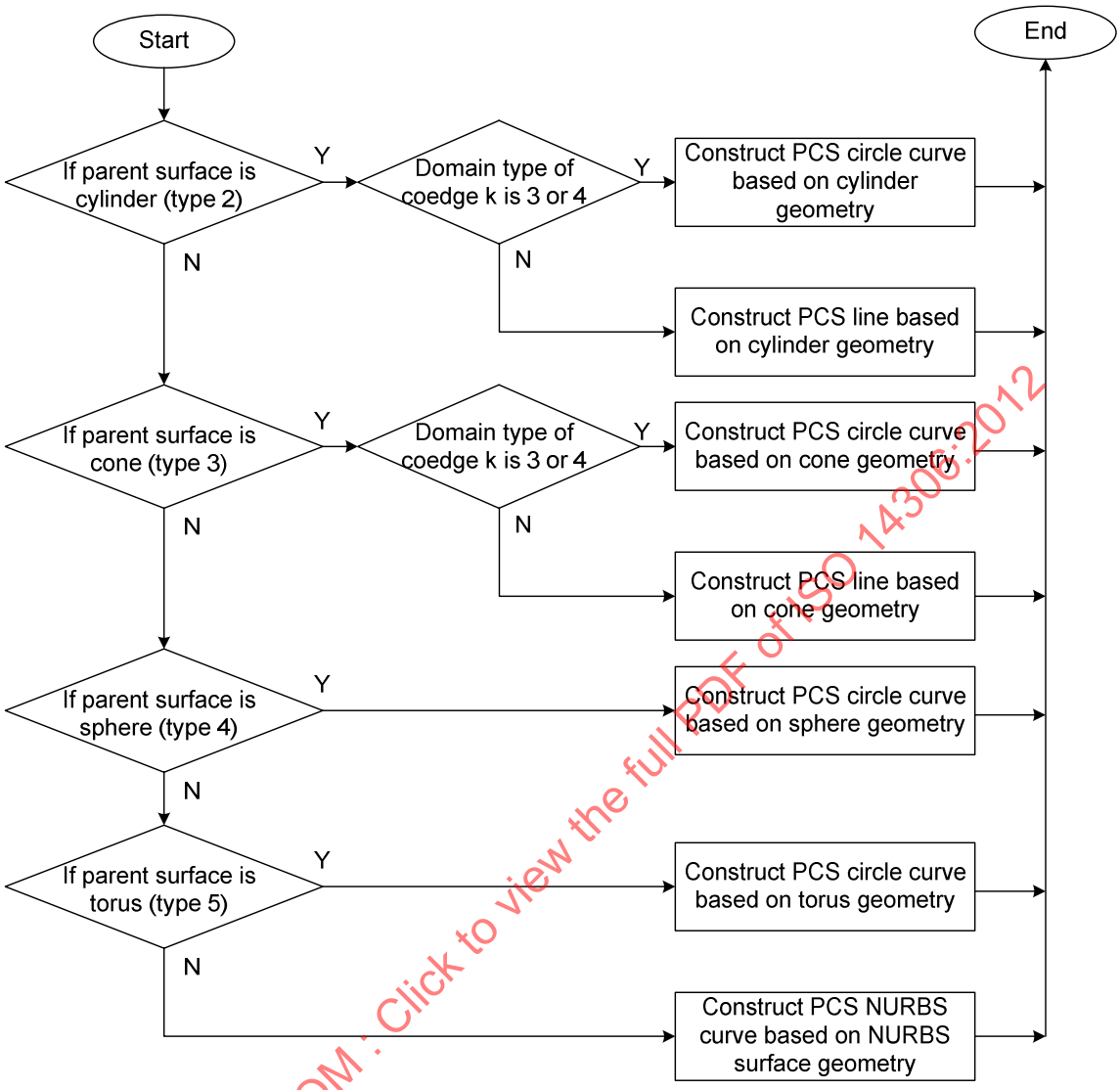


Figure 218 — MCS Curve Recovery from Surface Geometry

PCS Curve Recovery from MCS Curve and Surface Geometry

Shown in Figure 219 is the diagram illustrating how PCS curve geometry can be recovered from the combination of MCS curve and surface geometry.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

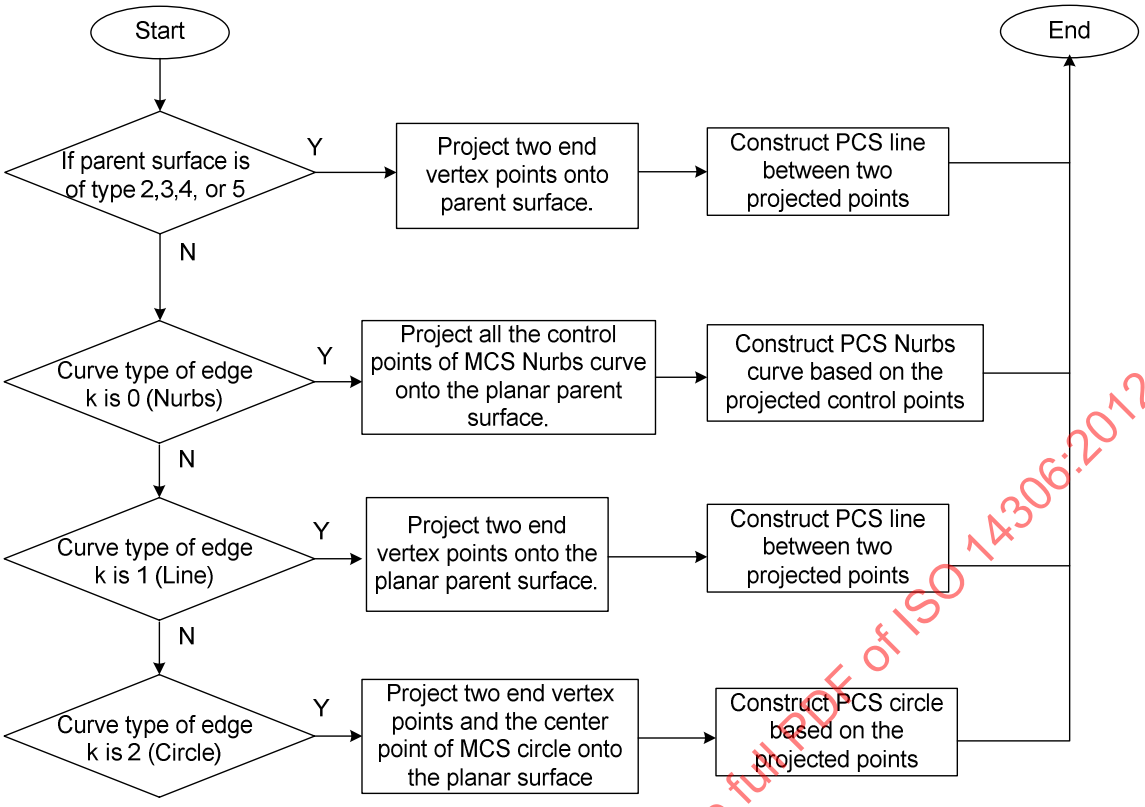


Figure 219 — PCS Curve Recovery from MCS Curve and Surface Geometry

12.2 JT LWPA Segment

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.

JT LWPA Segments are typically referenced by Part Node Elements (see 5.1.1.5 Part Node Element) using Late Loaded Property Atom Elements (see 5.2.7 Late Loaded Property Atom Element). The JT LWPA Segment type supports ZLIB compression on all element data, so all elements in JT LWPA Segment use the Logical Element Header ZLIB form of element header data.

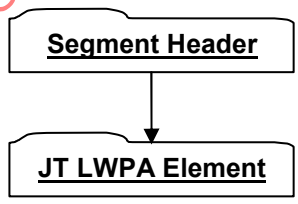


Figure 220 — JT LWPA Segment data collection

Complete description for Segment Header can be found in 4.1.3.1 Segment Header.

12.3 JT LWPA Element

Object Type ID: 0xd67f8ea8, 0xf524, 0x4879, 0x92, 0x8c, 0x4c, 0x3a, 0x56, 0x1f, 0xb9, 0x3a

JT LWPA Segment represents a particular Part's precise analytic surfaces. It can be viewed as a subset of B-Rep representation where the subset refers to the complete collection of all the surfaces that are of one of the analytic types shown in the Supported Surface Type table, i.e., plane, cylinder, cone, sphere, or torus. Unlike JT B-Rep Element or XT B-Rep Element, JT LWPA Element does not contain any B-Rep topology information,

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

nor does it contain geometric curve or point information. LWPA is designed to represent most essential part geometry information with much lighter weight on disk and much faster to load than B-Rep. Typically LWPA is less than 2 percent of B-Rep size on disk, and takes less than 5 percent time to load into memory. The analytic representation of LWPA follows XT convention as detailed in Annex E.

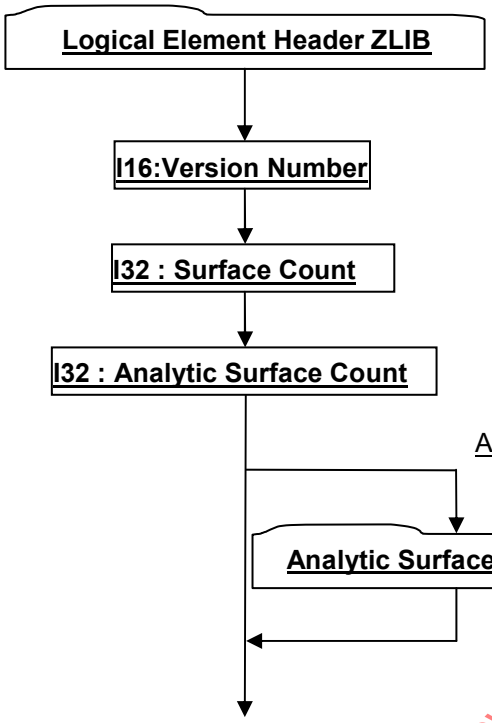


Figure 221 — JT LWPA Element data collection

I16:Version Number

Version Number is the version identifier for this JT LWPA Element. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : Surface Count

Surface Count indicates the number of surface entries in LWPA. The number of surface entries is equal to the number of surfaces in the B-Rep representation. The surface entry does not contain any information if the corresponding B-Rep surface is not of analytic type.

I32 : Analytic Surface Count

Analytic Surface Count indicates the number of analytic surface entries in LWPA.

12.3.1 Analytic Surface Geometry

Analytic Surface Geometry defines a collection of analytic surfaces and their mapping to the original B-Rep surfaces.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

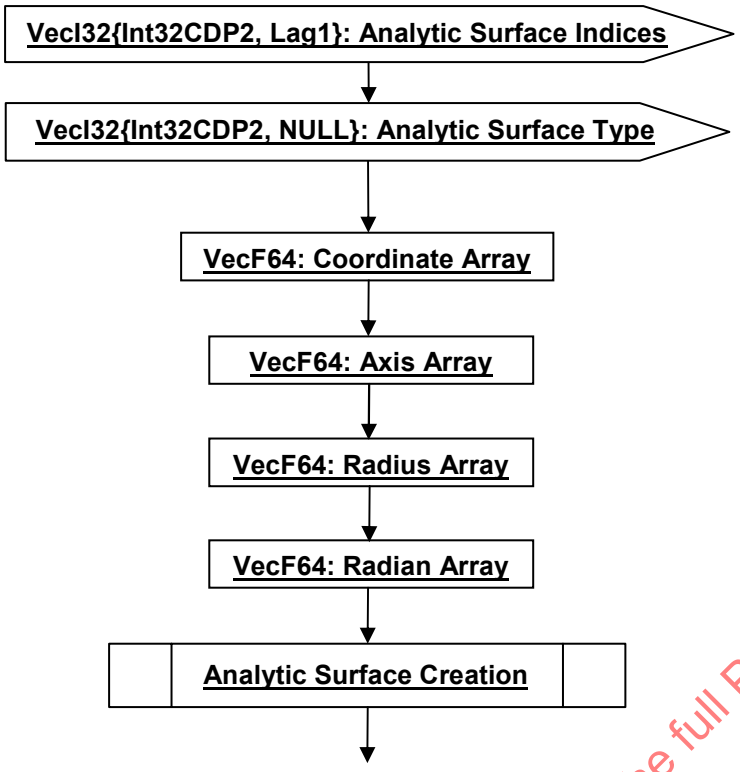


Figure 222 — Analytic Surface Geometry data collection

Vecl32{Int32CDP2, Lag1}: Analytic Surface Indices

Analytic Surface Indices is an integer array with its length equal to the number of analytic surfaces in the LWPA. The value of each element in this array represents the index of this analytic surface in the original B-Rep representation.

Vecl32{Int32CDP2, NULL}: Analytic Surface Type

Analytic Surface Type is an integer array with its length equal to the number of analytic surfaces in the LWPA. The value of each element in this array represents the type of each analytic surface, as defined in table Supported Surface Type

VecF64: Coordinate Array

Coordinate Array contains an array of double precision floating point numbers that represent the collection of point coordinate information in the definition of the analytic surface entities. The composite type VecF64 is defined in Table 6. Each floating point number in the array is written in binary form.

VecF64: Axis Array

Axis Array contains an array of double precision floating point numbers that represent the collection of unit vector information in the definition of the analytic surface entities. The composite type VecF64 is defined in Table 6. Each floating point number in the array is written in binary form.

VecF64: Radius Array

Radius Array contains an array of double precision floating point numbers that represent the collection of radius information in the definition of the analytic surface entities. The composite type VecF64 is defined in Table 6. Each floating point number in the array is written in binary form.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecF64: Radian Array

Radian Array contains an array of double precision floating point numbers that represent the collection of radian information in the definition of the analytic surface entities. The composite type VecF64 is defined in Table 6. Each floating point number in the array is written in binary form.

Analytic Surface Creation

Analytic surfaces in LWPA is constructed based on the information of the above arrays, as illustrated by logical diagram in Figure 223.

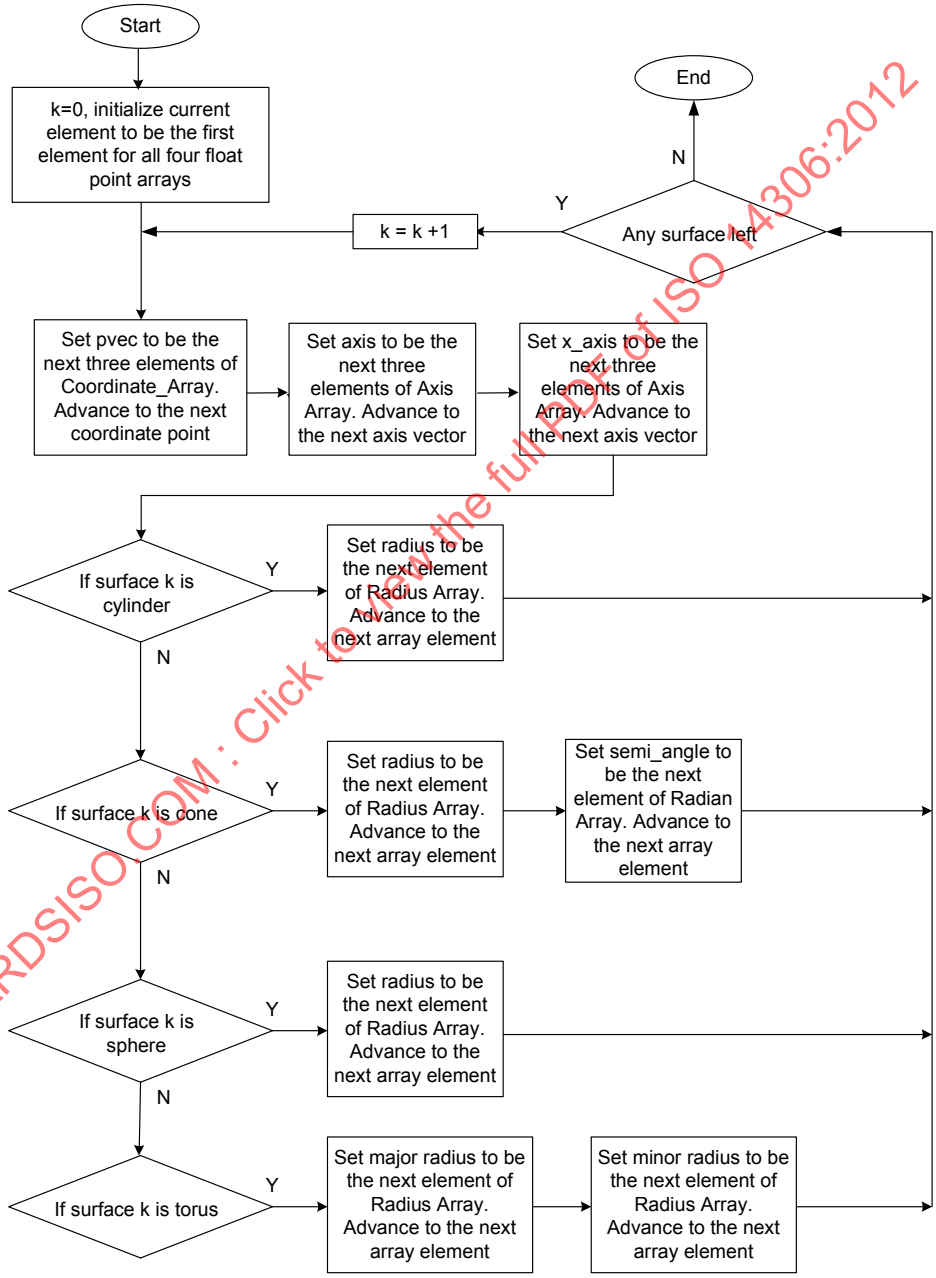


Figure 223 — Analytic Surface Creation

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

13 Data Compression and Encoding

The JT File format utilizes best-in-class compression and encoding algorithms to produce compact and efficient representations of data. The types of compression algorithms supported by the JT format vary from standard data type agnostic ZLIB deflation to advanced arithmetic algorithms that exploit knowledge of the characteristics of the data types they are compressing. Some of the JT format data collections are always stored in a compressed format, whereas other data collections support multiple compression storage formats that qualitatively vary from "Lossless" compression to more aggressive strategies that employ "lossy" compression. This support by the JT format of varying qualitative levels of compression allows producers of JT data to fine tune the tradeoff between compression ratio and fidelity of the data.

In some instances, data may be encoded/compressed using multiple techniques applied on top of one another in a serial fashion (i.e. encoding applied to the output of another encoder). One common example of this multiple encoding is when an array/vector of floating point data is first quantized into some integer codes and then these resulting integer codes are further compressed/encoded using an Arithmetic or BitLength CODEC (see 13.2 Encoding Algorithms).

Beyond the data collection specific compression/encoding, some JT format Data Segment types (see Data Segment) also support having a ZLIB compression conditionally applied to all the bytes of information persisted within the segment. So individual fields or collections of data may first have data type specific encoding/compression algorithms applied to them, and then if their Data Segment type supports it, the resulting data may be additionally compressed using a ZLIB deflation algorithm.

Whether, and at what qualitative level, a particular Data Segment's data is compressed/encoded is indicated through compression related data values stored as part of the particular Data Segment storage format. In general, aggressive application of advanced compression/encoding techniques is reserved for the heavy-weight renderable geometric data (e.g. triangles and wireframe lines) which can exist in a JT File.

The following sections document the format of the data compression/encoding within the JT file. Along with documenting the format, a technical description of the various compression/encoding algorithms is included and an example implementation of the decoding portion of the algorithms can be found within Annex C.

13.1 Common Compression Data Collection Formats

For convenience and brevity in documenting the JT format, this section of the reference documents the format for several common "data compression/encoding" related data collections that can exist in the JT format. You will find references to these common compression data collections in the 4.2 Data Segments section of the document.

13.1.1 Int32 Compressed Data Packet

The Int32 Compressed Data Packet collection represents the format used to encode/compress a collection of data into a series of Int32 based symbols. Note that the Int32 Compressed Data Packet collection can in itself contain another Int32 Compressed Data Packet collection if there are any "Out-Of-Band data." In the context of the JT format data compression algorithms and Int32 Compressed Data Packet, "out-of-band data" has the following meaning.

CODECs (e.g. Arithmetic, see 13.2 Encoding Algorithms for technical description) exploit the statistics present in the relative frequencies of the values being encoded. Values that occur frequently enough allow these methods to encode each of the values as a "symbol" in fewer bits that it would take to encode the value itself. Values that occur too infrequently to take advantage of this property are written *aside* into the "out-of-band data" array to be encoded separately. An "escape" symbol is encoded in their place as a placeholder in the primal CODEC (note, see "Symbol" data field definition in Int32 Probability Context Table Entry for further details on the representation of "escape" symbol).

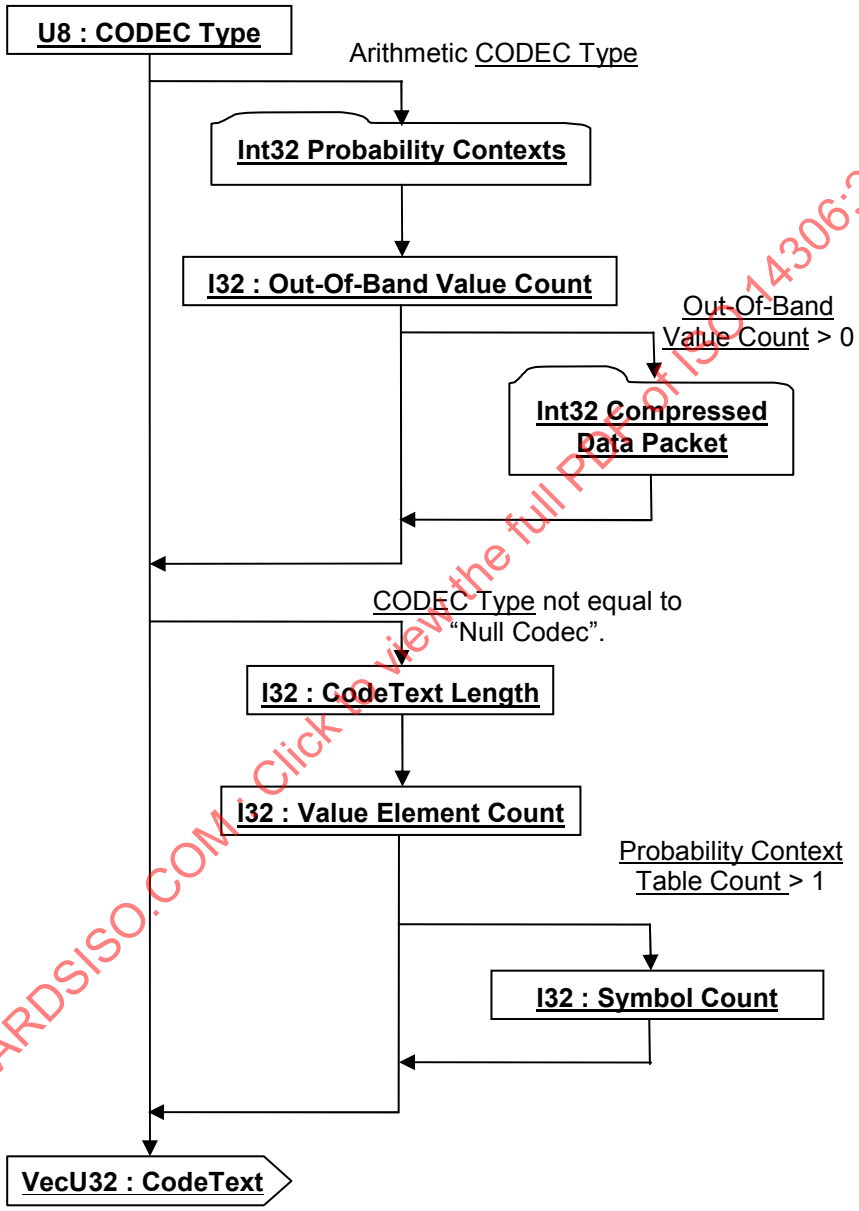
Essentially the "out-of-band data" is the high-entropy residue left over after the CODECs have squeezed all the advantage out of the original data stream that they can. However, this "out-of-band data" is sent back around for another pass because sometimes there are *different* statistics to be taken advantage of. When all other coding options have been exhausted, the Bitlength CODEC is invoked. The Bitlength CODEC directly

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

encodes all values given to it, does not require a probability context, and hence never produces additional “out-of-band data”. The byte stops there, in other words.

In some cases, all values may be written as "out of band" when the Codec cannot perform *any* useful compression. In this case, the encoded I32 : CodeText Length field will be 0, and the I32 : Out-Of-Band Value Count will be equal to I32 : Value Element Count. The implied action in this case is to merely copy the Out-Of-Band value data into the output Value Element array instead of invoking the Codec.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 224 — Int32 Compressed Data Packet data collection

U8 : CODEC Type

CODEC Type specifies the algorithm used to encode/decode the data. See 13.2 Encoding Algorithms for complete explanation of each of the encoding algorithms.

© ISO 2012 – All rights reserved

Table 101 — Common Compression CODEC Type values

= 0	Null CODEC
= 1	Bitlength CODEC
= 3	Arithmetic CODEC
= 4	Chopper CODEC

I32 : Out-Of-Band Value Count

Out-Of-Band Value Count specifies the number of values that are “Out-Of-Band.” This data field is only present for the Arithmetic CODEC Type.

I32 : CodeText Length

CodeText Length specifies the total number of bits of CodeText data (CodeText data field is described below). This data field is only present if CODEC Type is not equal to “Null CODEC.”

I32 : Value Element Count

Value Element Count specifies the number of values that the CODEC is expected to decode (i.e. it’s like the “length” field written if you’re just writing out a vector of integers). This data field is only present if CODEC Type is not equal to “Null CODEC.” Upon completion of decoding the CodeText data field below, the number of decoded Values should be equal to Value Element Count. When only a single Probability Context Table is used, Value Element Count will also be equal to the number of Symbols decoded upon completion of decoding.

I32 : Symbol Count

When two Probability Context Tables are being used, Symbol Count specifies the number of Symbols to be decoded by the Arithmetic CODEC. There is a subtlety present in the method CodecDriver::addOutputSymbol() when it is passed an Escape symbol. Only if the Codec is using Probability Context Table 0 when it receives an Escape symbol does it emit a Value from the "Out-Of-Band" data array. Because of this subtlety, the number of Symbols decoded can be larger than the number of Values produced, thus the reason for writing this field distinct from Value Element Count.

VecU32 : CodeText

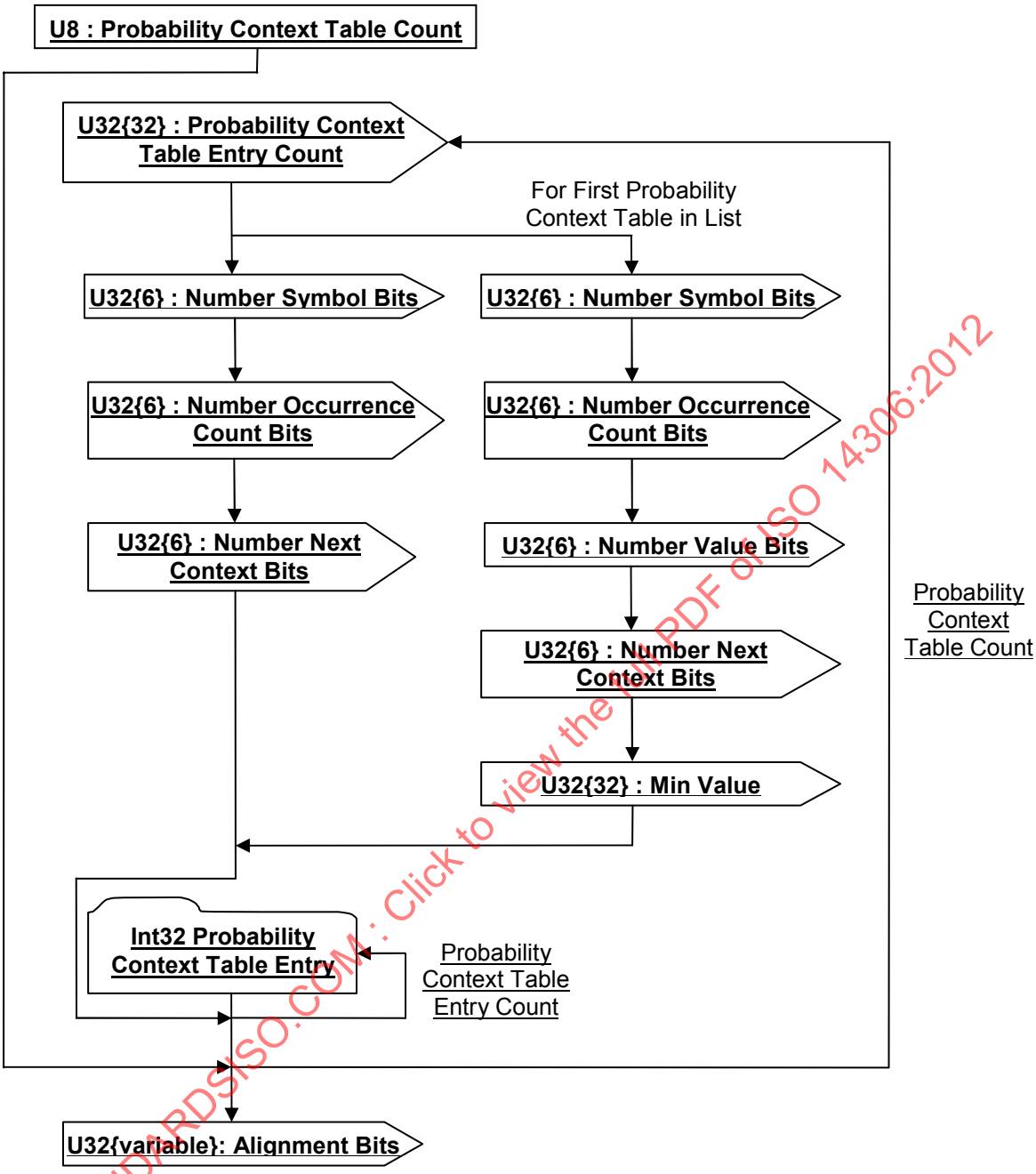
CodeText is the array/vector of encoded symbols. For CODEC Type not equal to “Null CODEC”, the total number of bits of encoded data in this array is indicated by the previously described CodeText Length data field.

Int32 Probability Contexts

Int32 Probability Contexts data collection is a list of Probability Context Tables. The Int32 Probability Contexts data collection is only present for the Arithmetic CODEC Type. A Probability Context Table is a trimmed and scaled histogram of the input values. It tallies the frequencies of the several most frequently occurring values. It is central to the operation of the arithmetic CODEC.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 225 — Int32 Probability Contexts data collection

U8 : Probability Context Table Count

Probability Context Table Count specifies the number of Probability Context Tables to follow and will always have a value of either “1” or “2”.

U32{32} : Probability Context Table Entry Count

Probability Context Table Entry Count specifies the number of entries in this Probability Context Table.

U32{6} : Number Symbol Bits

Number Symbol Bits specifies the number of bits used to encode the Symbol range.

© ISO 2012 – All rights reserved

U32{6} : Number Occurrence Count Bits

Number Occurrence Count Bits specifies the number of bits used to encode the Occurrence Count range.

U32{6} : Number Value Bits

Number Value Bits specifies the number of bits used to encode the Associated Value range. Note that Number Value Bits is only specified in the JT file for the *first* Probability Context Table. If a second Probability Context Table is present, the Number Value Bits from the first should be used for the second as well.

U32{6} : Number Next Context Bits

Number Next Context Field Bits specifies the number of bits used for the Next Context Field in Int32 Probability Context Table Entry.

U32{32} : Min Value

Min Value specifies the minimum of all Associated Values (i.e. one per table entry) stored in this Probability Context Table. This value is used to compute the real Associated Value for a Probability Context Table Entry. See Associated Value description in Int32 Probability Context Table Entry.

U32{variable}: Alignment Bits

Alignment Bits represents the number of additional padding bits stored to arrive at the next even multiple of 8 bits. Values of "0" are stored in the alignment bits.

Note: Data written into the JT file is always aligned on bytes. Therefore after reading in a block of bit data such as the probability context tables it is necessary to discard any remaining bits on the last byte that is read in. This is represented by the "Alignment Bits" entry.

Int32 Probability Context Table Entry

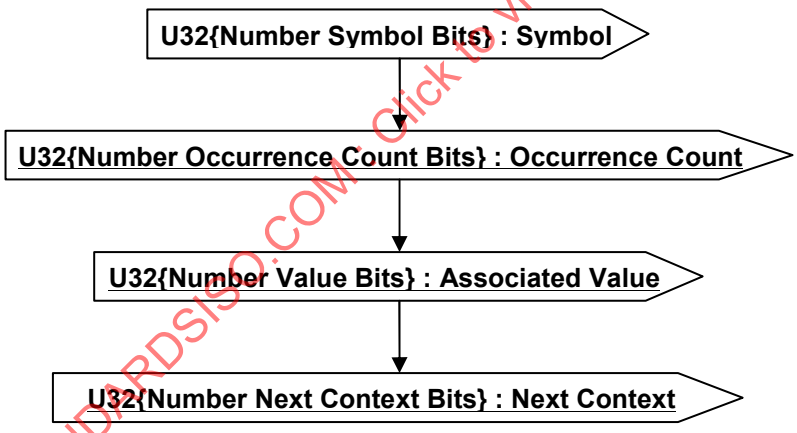


Figure 226 — Int32 Probability Context Table Entry data collection

U32{Number Symbol Bits} : Symbol

Symbol is a small integer number associated with a specific value in the context table. It serves only to impose an order on the entries in the Probability Context Table. The symbol is stored with a "+2" added to the value and thus a reader shall subtract "2" from the read value to get the true symbol value. Complete description for Number Symbol Bits can be found in Int32 Probability Contexts.

Note: Even though the symbol is written as a U32{Number Symbol Bits} it is possible to end up with a negative number after subtracting "2" from the read in value. One example that will occur frequently is the escape symbol used for out-of-band data which will have the value "0" in the file, however it will become "-2", its true symbol value, after subtracting "2" from the read in "0" value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

U32{Number Occurrence Count Bits} : Occurrence Count

Occurrence Count specifies the relative frequency of the value. Complete description for Number Occurrence Count Bits can be found in Int32 Probability Contexts.

Note: Occurrence Counts for all symbols are normalized (converted to a relative frequency) during the write process in order to ensure the minimum amount of bits possible is used to write them. This has several implications the reader should be aware of:

The sum of all Occurrence Counts is not guaranteed to equal the number of symbols to be decoded (see Value Element Count in section 13.1.1 for number of symbols to be decoded). During Arithmetic decoding as described in Annex B.

pDriver->numSymbolsToRead() – Refers to the total number of symbols to be decoded (i.e. Value Element Count in section 13.1.1 when the number of Probability Context Tables is equal to 1, or Symbol Count when the number of Probability Context Tables is 2).

pCurrContext->totalCount() – Refers to the sum of the "Occurrence Count" values for all the symbols associated with a Probability Context.

U32{Number Value Bits} : Associated Value

Associated Value is the value (from the input data) that the symbol represents. The CODECs don't directly encode values, they encode symbols. Symbols, then, are associated with specific values, so when the CODEC decodes an array of symbols, you can reconstruct the array of values that was intended by looking up the symbols in the Probability Context Table. This value is stored with "Min Value" subtracted from the value. Complete descriptions for "Min Value" and Number Value Bits can be found in Int32 Probability Contexts.

Note: The associated value for an escape symbol is undefined and therefore can be any valid U32 number.

U32{Number Next Context Bits} : Next Context

Next Context field specifies which Probability Context Table to use when decoding the next symbol. The value of this field will be greater than or equal to 0, and less than Probability Context Table Count.

13.1.2 Int32 Compressed Data Packet Mk. 2

The Int32 Compressed Data Packet Mk. 2 collection represents an enhanced form of the original Int32 Compressed Data Packet. Note that the Int32 Compressed Data Packet Mk. 2 collection can in itself contain another Int32 Compressed Data Packet Mk. 2 collection if there are any "Out-Of-Band data." In the context of the JT format data compression algorithms and Int32 Compressed Data Packet Mk. 2, "out-of-band data" has the meaning described below.

Entropy CODECs (e.g. Arithmetic) exploit the statistics present in the relative frequencies of the values being encoded. Values that occur frequently enough allow these methods to encode each of the values as a "symbol" in fewer bits that it would take to encode the value itself. Values that occur too infrequently to take advantage of this property are written *aside* into the "out-of-band data" array to be encoded separately. An "escape" symbol is encoded in their place as a placeholder in the primal CODEC (note, see "Symbol" data field definition in Int32 Probability Context Table Entry Mk. 2 for further details on the representation of "escape" symbol).

Essentially the "out-of-band data" is the high-entropy residue left over after the CODEC has squeezed all the advantage out of the original data stream that it can. However, this "out-of-band data" is sent back around for another pass because sometimes there are *new* or *different* statistics to be exploited.

The Int32 Compressed Data Packet Mk. 2 brings the new *Chopper* pseudo-CODEC to the table. Its job is to identify fields of bits in a sequence of otherwise incompressible data that may be hiding low-entropy statistics that can be profitably exploited. In other words, it "chops" the input data up into bit fields, and then encodes them separately using the Arithmetic or BitLength CODECs, or in some cases, another round of chopping. The Chopper also removes *value bias* from the original input data array. Some input data arrays may contain values that are clustered around a certain central value. In these cases, it is profitable to first subtract out a

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

bias value from the original input data. In some cases, this simple expedient may dramatically reduce the apparent field width necessary to code the variation in the original sequence.

In some cases, all values may be written as "out of band" when the Codec cannot perform *any* useful compression. In this case, the encoded I32 : CodeText Length field will be 0, and the I32 : Out-Of-Band Value Count will be equal to I32 : Value Element Count. The implied action in this case is to merely copy the Out-Of-Band value data into the output Value Element array instead of invoking the Codec.

When all other coding options have been exhausted, the Bitlength CODEC is invoked. The Bitlength CODEC directly encodes all values given to it, does not require a probability context, and hence never produces additional “out-of-band data”. The byte stops there, in other words.

Note that in the diagram below, encoding can loop back recursively for Out-Of-Band data and chopper fields. *For JT v9 files, the maximum recursion depth may not exceed three*.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

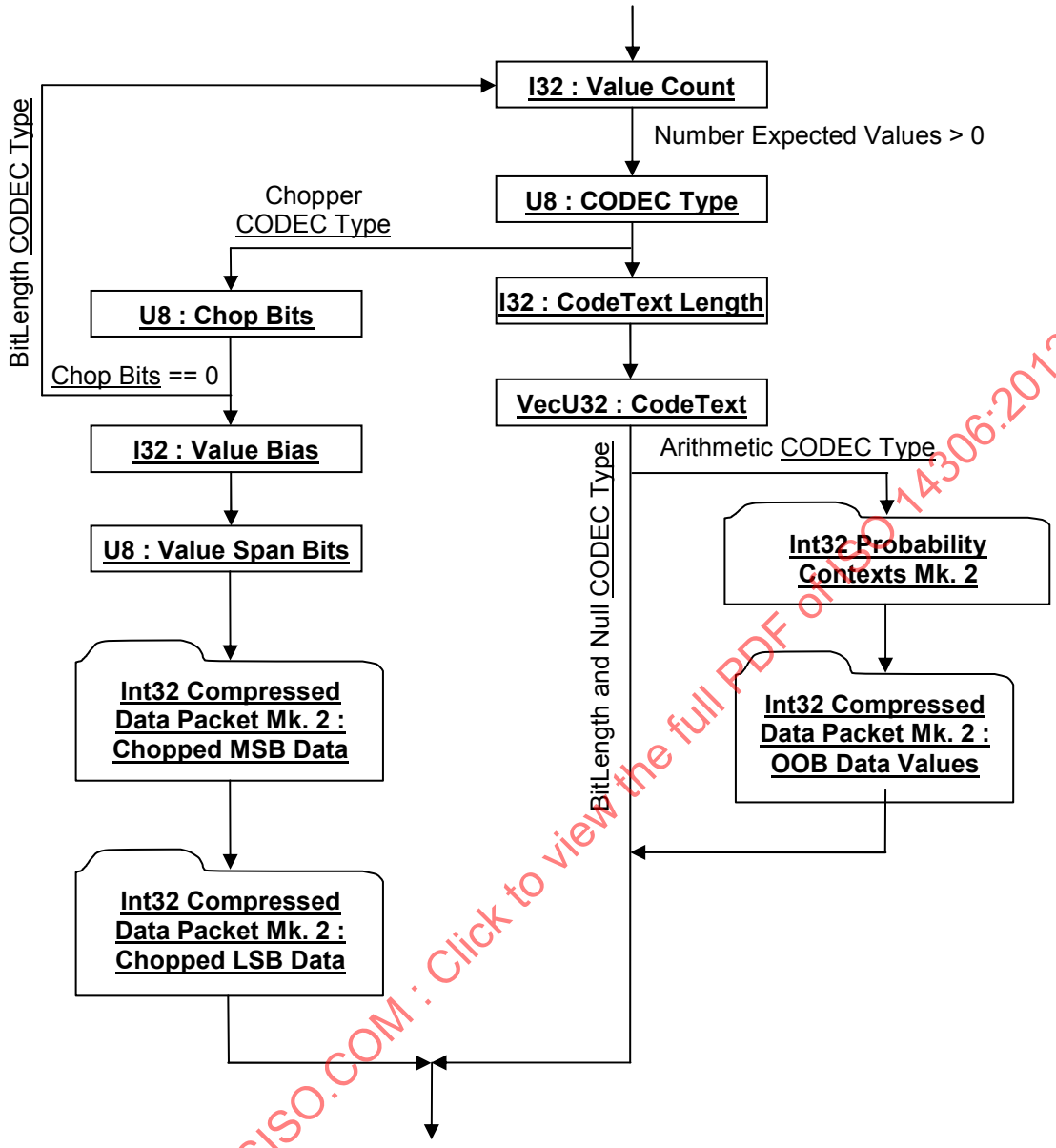


Figure 227 — Int32 Compressed Data Packet Mk. 2 data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

I32 : Value Count

Value Count specifies the number of values that the CODEC is expected to decode (i.e. it's like the "length" field written if you're just writing out a vector of integers). Upon completion of decoding the CodeText data field below, the number of decoded Values should be equal to Value Count. When only a single Probability Context Table is used, Value Element Count will also be equal to the number of Symbols decoded upon completion of decoding.

U8 : CODEC Type

CODEC Type specifies the algorithm used to encode/decode the data. See 13.2 Encoding Algorithms for complete explanation of each of the encoding algorithms.

© ISO 2012 – All rights reserved

Table 102 — Int32 Probability Contexts CODEC Type values

= 0	Null CODEC
= 1	Bitlength CODEC
= 3	Arithmetic CODEC
= 4	Chopper CODEC

I32 : CodeText Length

CodeText Length specifies the total number of bits of CodeText data (CodeText data field is described below).

VecU32 : CodeText

CodeText is the array/vector of encoded symbols. For CODEC Type not equal to "Null CODEC", the total number of bits of encoded data in this array is indicated by the previously described CodeText Length data field.

U8 : Chop Bits

Chop Bits specifies the number of high-order bits "chopped off" from the *biased* input data array and coded separately from the low-order bits. Repeated applications of the Chopper pseudo-CODEC can expose low-entropy bit fields that would be inaccessible by directly coding the data array. Chop Bits is the number of bits coded into the Chopped MSB Data field.

I32 : Value Bias

Value Bias is the (signed) number that is subtracted from the original input data array elements *before* computing Value Span Bits and Chop Bits. See Chopped LSB Data below for a full explanation of how to reconstitute the original data values using Value Bias and the two chopped fields.

U8 : Value Span Bits

Value Span Bits specifies the total bit width of the *biased* input data array. Note that Value Span Bits minus Chop Bits is the number of low-order bits present in the Chopped LSB Data field.

Int32 Compressed Data Packet Mk. 2 : Chopped MSB Data

This field contains the separately compressed most significant bits of the *biased* input data array, whose elements contain Value Span Bits bits of significance. In other words, this field contains the bit field from the *biased* data array beginning at bit number ValueSpan-ChopBits and ending at bit number ValueSpan-1 inclusive. This field may contain negative numbers.

Int32 Compressed Data Packet Mk. 2 : Chopped LSB Data

This field contains the separately compressed most significant bits of the original input data array, whose elements contain Value Span Bits bits of significance. In other words, this field contains the bit field from the original data array beginning at bit number 0 and ending at bit number ValueSpan-ChopBits-1 inclusive. This field may only contain positive numbers; all bits above this range shall encode to 0. A pseudo-code representation of the re-constituting the original data values is as follows:

```
OrigValue[i] = (LSBValue[i] | (MSBValue[i] << (ValSpanBits - ChopBits))) + ValueBias;
```

Int32 Compressed Data Packet Mk. 2 : OOB Data Values

This field encodes the out-of-band values associated with the Arithmetic CODEC.

Int32 Probability Contexts Mk. 2

Int32 Probability Contexts Mk. 2 data collection encodes a Probability Context Table, and is present only for the Arithmetic CODEC Type. A Probability Context Table is a trimmed and scaled histogram of the input values. It tallies the frequencies of the several most frequently occurring values. It is central to the operation of the Arithmetic CODEC.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

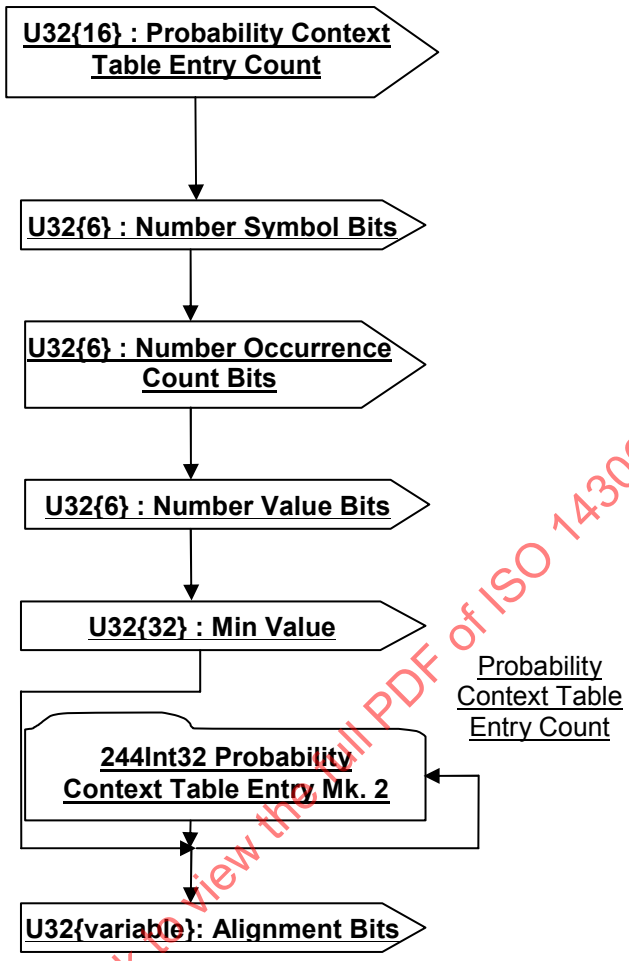


Figure 228 — Int32 Probability Contexts Mk. 2 data collection

U32{16} : Probability Context Table Entry Count

Probability Context Table Entry Count specifies the number of entries in this Probability Context Table.

U32{6} : Number Symbol Bits

Number Symbol Bits specifies the number of bits used to encode the Symbol range.

U32{6} : Number Occurrence Count Bits

Number Occurrence Count Bits specifies the number of bits used to encode the Occurrence Count range.

U32{6} : Number Value Bits

Number Value Bits specifies the number of bits used to encode the Associated Value range. Note that Number Value Bits is only specified in the JT file for the *first* Probability Context Table. If a second Probability Context Table is present, the Number Value Bits from the first should be used for the second as well.

U32{32} : Min Value

Min Value specifies the minimum of all Associated Values (i.e. one per table entry) stored in this Probability Context Table. This value is used to compute the real Associated Value for a Probability Context Table Entry. See Associated Value description in Int32 Probability Context Table Entry.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

U32{variable}: Alignment Bits

Alignment Bits represents the number of additional padding bits stored to arrive at the next even multiple of 8 bits. Values of “0” are stored in the alignment bits.

Note: Data written into a JT file is always aligned on bytes. Therefore after reading in a block of bit data such as the probability context tables it is necessary to discard any remaining bits on the last byte that is read in. This is represented by the “Alignment Bits” entry.

Int32 Probability Context Table Entry Mk. 2

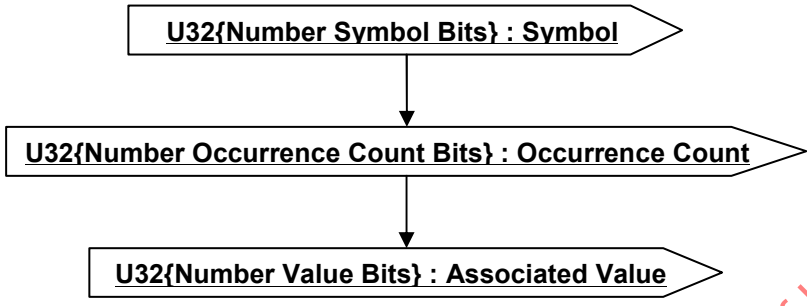


Figure 229 — Int32 Probability Context Table Entry Mk. 2 data collection

U32{Number Symbol Bits} : Symbol

Symbol is a small integer number associated with a specific value in the context table. It serves only to impose an order on the entries in the Probability Context Table. The symbol is stored with a “+2” added to the value and thus a reader shall subtract “2” from the read value to get the true symbol value. Complete description for Number Symbol Bits can be found in Int32 Probability Contexts Mk. 2.

Note: Even though the symbol is written as a U32{Number Symbol Bits} it is possible to end up with a negative number after subtracting “2” from the read in value. One example that will occur frequently is the escape symbol used for out-of-band data which will have the value “0” in the file, however it will become “-2”, its true symbol value, after subtracting “2” from the read in “0” value.

U32{Number Occurrence Count Bits} : Occurrence Count

Occurrence Count specifies the relative frequency of the value. Complete description for Number Occurrence Count Bits can be found in Int32 Probability Contexts Mk. 2.

Note: Occurrence Counts for all symbols are normalized (converted to a relative frequency) during the write process in order to ensure the minimum amount of bits possible is used to write them while closely approximating their actual frequency. This has several implications the reader should be aware of:

The sum of all Occurrence Counts is not guaranteed to equal the number of symbols to be decoded (see I32 : Value Count in section 13.1.2 for number of symbols to be decoded).

During Arithmetic decoding as described in Annex B .

pDriver->numSymbolsToRead() – Refers to the total number of symbols to be decoded (i.e. I32 : Value Count in section 13.1.2).

pCurrContext->totalCount() – Refers to the sum of the “Occurrence Count” values for all the symbols associated with a Probability Context.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

U32{Number Value Bits} : Associated Value

Associated Value is the value (from the input data) that the symbol represents. The CODECs don't directly encode values, they encode symbols. Symbols, then, are associated with specific values, so when the CODEC decodes an array of symbols, you can reconstruct the array of values that was intended by looking up the symbols in the Probability Context Table. This value is stored with "Min Value" subtracted from the value. Complete descriptions for "Min Value" and Number Value Bits can be found in Int32 Probability Contexts Mk. 2.

Note: The associated value for an escape symbol is undefined and therefore can be any valid U32 number.

13.1.3 Float64 Compressed Data Packet

The Float64 Compressed Data Packet collection represents the format used to encode/compress a collection of data into a series of Float64 based symbols. This compression format also uses the concept of "out-of-band data" in its data contents definition. In the context of the JT format data compression algorithms and Float64 Compressed Data Packet, "out-of-band data" has the following meaning.

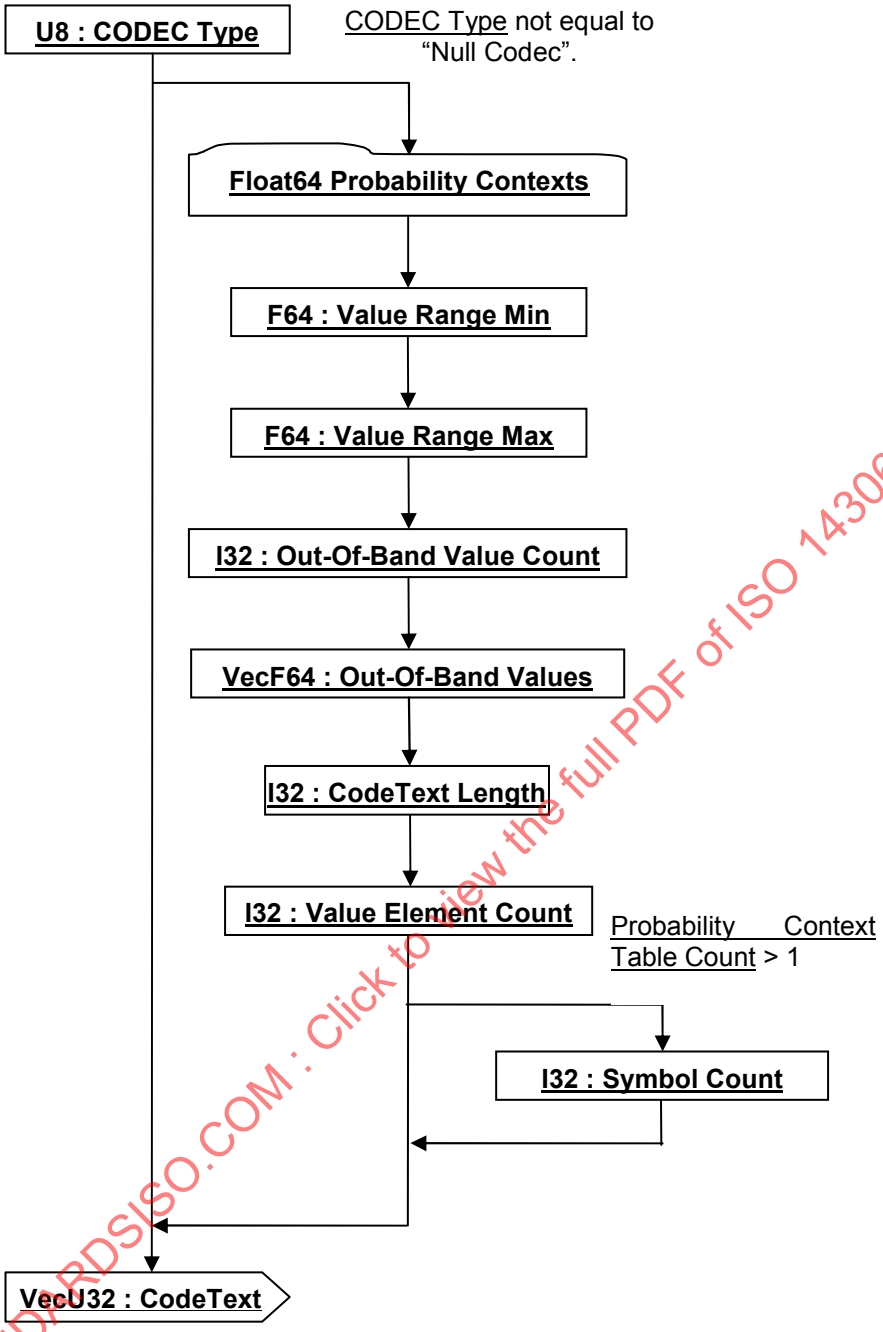
The Arithmetic CODEC (see 13.2 Encoding Algorithms for technical description) can exploit the statistics present in the relative frequencies of the values being encoded. Values that occur frequently enough allow the CODEC to encode each of the values as a "symbol" in fewer bits that it would take to encode the value itself. Values that occur too infrequently to take advantage of this property are written *aside* into the "out-of-band data" array. An "escape" symbol (i.e. value of "-2") is encoded in their place as a marker in the primal CODEC. Essentially the "out-of-band data" is the high-entropy junk/residue/slag left over after the CODECs have squeezed all the advantage out that it can.

Whereas the Int32 Compressed Data Packet (see 13.1.1 Int32 Compressed Data Packet) then sends this "out-of-band data" back around through a new CODEC looking for *different* statistics to be taken advantage of, the Float64 Compressed Data Packet simply writes out the "out-of-band data" array with no additional encoding attempted.

In some cases, all values may be written as "out of band" when the Codec cannot perform *any* useful compression. In this case, the encoded I32 : CodeText Length field will be 0, and the I32 : Out-Of-Band Value Count will be equal to I32 : Value Element Count. The implied action in this case is to merely copy the Out-Of-Band value data into the output Value Element array instead of invoking the Codec.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 230 — Float64 Compressed Data Packet data collection

U8 : CODEC Type

CODEC Type specifies the algorithm used to encode/decode the data. See 13.2 Encoding Algorithms for complete explanation of each of the encoding algorithms.

Table 103 — Float64 Compressed Data Packet CODEC Type values

= 0	Null CODEC
= 1	Bitlength CODEC
= 3	Arithmetic CODEC
= 4	Chopper CODEC

© ISO 2012 – All rights reserved

F64 : Value Range Min

Value Range Min specifies the minimum of the value range used to encode the values. This data field is only present if CODEC Type is not equal to “Null CODEC.”

F64 : Value Range Max

Value Range Max specifies the maximum of the value range used to encode the values. This data field is only present if CODEC Type is not equal to “Null CODEC.”

I32 : Out-Of-Band Value Count

Out-Of-Band Value Count specifies the number of values that are “Out-Of-Band.” This data field is only present if CODEC Type is not equal to “Null CODEC.”

VecF64 : Out-Of-Band Values

Out-Of-Band Values specifies the vector/list of “Out-Of-Band” values. This data field is only present if CODEC Type is not equal to “Null CODEC.”

I32 : CodeText Length

CodeText Length specifies the total number of bits of CodeText data (described below). This data field is only present if CODEC Type is not equal to “Null CODEC.”

I32 : Value Element Count

Value Element Count specifies the number of values that the CODEC is expected to decode (i.e. it’s like the “length” field written if you’re just writing out a vector of integers). This data field is only present if CODEC Type is not equal to “Null CODEC.” Upon completion of decoding the CodeText data field below, the number of decoded symbol values should be equal to Value Element Count.

I32 : Symbol Count

When two Probability Context Tables are being used, Symbol Count specifies the number of Symbols to be decoded by the Arithmetic CODEC. There is a subtlety present in the method CodecDriver::addOutputSymbol() when it is passed an Escape symbol. Only if the Codec is using Probability Context Table 0 when it receives an Escape symbol does it emit a Value from the "Out-Of-Band" data array. Because of this subtlety, the number of Symbols decoded can be larger than the number of Values produced, thus the reason for writing this field distinct from Value Element Count.

VecU32 : CodeText

CodeText is the array/vector of encoded symbols. For CODEC Type not equal to “Null CODEC”, the total number of bits of encoded data in this array is indicated by the previously described CodeText Length data field.

Float64 Probability Contexts

Float64 Probability Contexts data collection is a list of Probability Context Tables. A Probability Context Table is a trimmed and scaled histogram of the input values. It tallies the frequencies of the several most frequently occurring values. It is central to the operation of the arithmetic CODEC.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

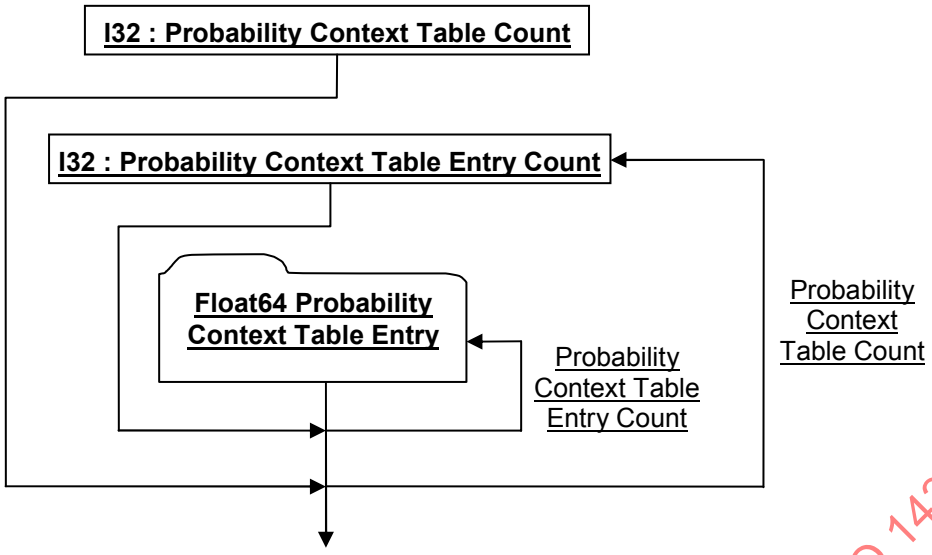


Figure 231 — Float64 Probability Contexts data collection

I32 : Probability Context Table Count

Probability Context Table Count specifies the number of Probability Context Tables to follow and will always have a value of either “1” or “2”.

I32 : Probability Context Table Entry Count

Probability Context Table Entry Count specifies the number of entries in this Probability Context Table.

Float64 Probability Context Table Entry

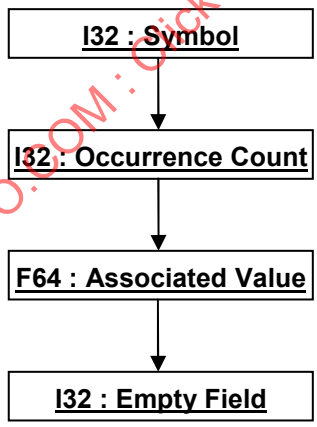


Figure 232 — Float64 Probability Context Table Entry data collection

I32 : Symbol

Symbol is a small integer number associated with a specific value in the context table. It serves only to impose an order on the entries in the Probability Context Table. Note that a value of “-2” represents the “escape” symbol placeholder encoded for “out-of-band data” (see 13.1.3 Float64 Compressed Data Packet for additional details).

I32 : Occurrence Count

Occurrence Count specifies the relative frequency of the value.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

F64 : Associated Value

Associated Value is the value (from the input data) that the symbol represents. The CODECs don't directly encode *values,* they encode *symbols*. Symbols, then, are associated with specific values, so when the CODEC decodes an array of symbols, you can reconstruct the array of values that was intended by looking up the symbols in the Probability Context Table.

I32 : Empty Field

Refer to best practice 14.4 Empty Field.

13.1.4 Compressed Vertex Coordinate Array

The Compressed Vertex Coordinate Array data collection contains the quantization data/representation for a set of vertex coordinates.

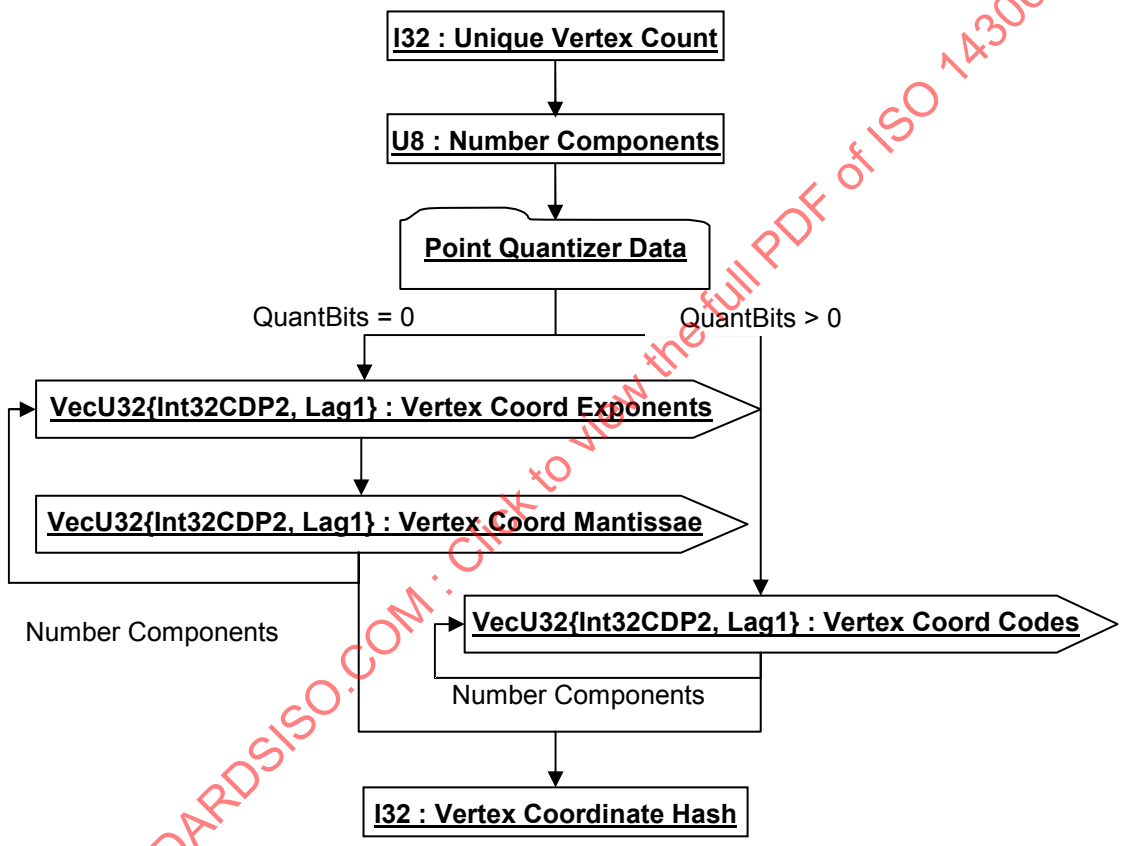


Figure 233 — Compressed Vertex Coordinate Array data collection

Complete description for Point Quantizer Data can be found in 13.1.4 Point Quantizer Data.

The above predicates "QuantBits = 0" and "QuantBits > 0" refer to the value of the field U8 : Number Of Bits stored in the three components of 13.1.9 Point Quantizer Data. All three of these fields are required to be equal.

I32 : Unique Vertex Count

Vertex Count specifies the count (number of unique) vertices in the Vertex Codes arrays. Identical values are only stored once therefore it may be necessary to smear out the vertices as described in TopoMesh Compressed Rep Data V1 and TopoMesh Topologically Compressed LOD Data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

U8 : Number Components

Number Components specifies the number of vertex components present for each vertex record in the set of vertex records.

VecU32{Int32CDP2, Lag1} : Vertex Coord Exponents

Vertex Coord Exponents is a vector of Floating Point Exponents and Sign for all the ith component values of a set of vertex coordinates. Vertex Coord Exponents uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Vertex Coord Mantissae

Vertex Coord Mantissae is a vector of Floating Point Mantissae for all the ith component values of a set of vertex coordinates. Vertex Coord Mantissae uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Vertex Coord Codes

Vertex Coord Codes is a vector of quantizer "codes" for all the ith component values of a set of vertex coordinates. Vertex Coord Codes uses the Int32 version of the CODEC to compress and encode data.

I32 : Vertex Coordinate Hash

The Vertex Coordinate Hash is the combined hash of the unique vertex coordinate records. If the number of quantization bits is equal to zero the hash value is equal to the combined hash of the vertex coordinate values for each of the component arrays. If the number of quantization bits is greater than 0 the hash value is equal to the combined hash of the vertex coordinates codes for each of the component arrays. Refer to section 14.6 for a more detailed description on hashing.

```
UInt32 uHash    = 0;
uInt32 nUniqVtx = 0;
vecF32 vCoord[nUniqVtx][3];
vecU32 vCodes[3];
...
if ( uQuantBits == 0 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
    for ( int j=0 ; j<nUniqVtx ; j++) {
      uHash = hash32( (UInt32*)(&vCoord[j][i]), 1, uHash );
    }
  }
} else {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vCodes[i], nUniqVtx, uHash );
  }
}
```

13.1.5 Compressed Vertex Normal Array

The Compressed Vertex Normal Array data collection contains the compressed data/representation for a set of vertex normals. Compressed Vertex Normal Array data collection is only present if previously read vertex bindings denote normals are presents (See Vertex Shape LOD Data U64 : Vertex Bindings for complete explanation of the vertex bindings).

A variation of the CODEC developed by Michael Deering at Sun Microsystems is used to encode the normals when quantization is enabled. The variation being that the "Sextants" are arranged differently than in Deering's scheme [4], for better delta encoding. See 13.2.4 Deering Normal CODEC for a complete explanation on the Deering CODEC used.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

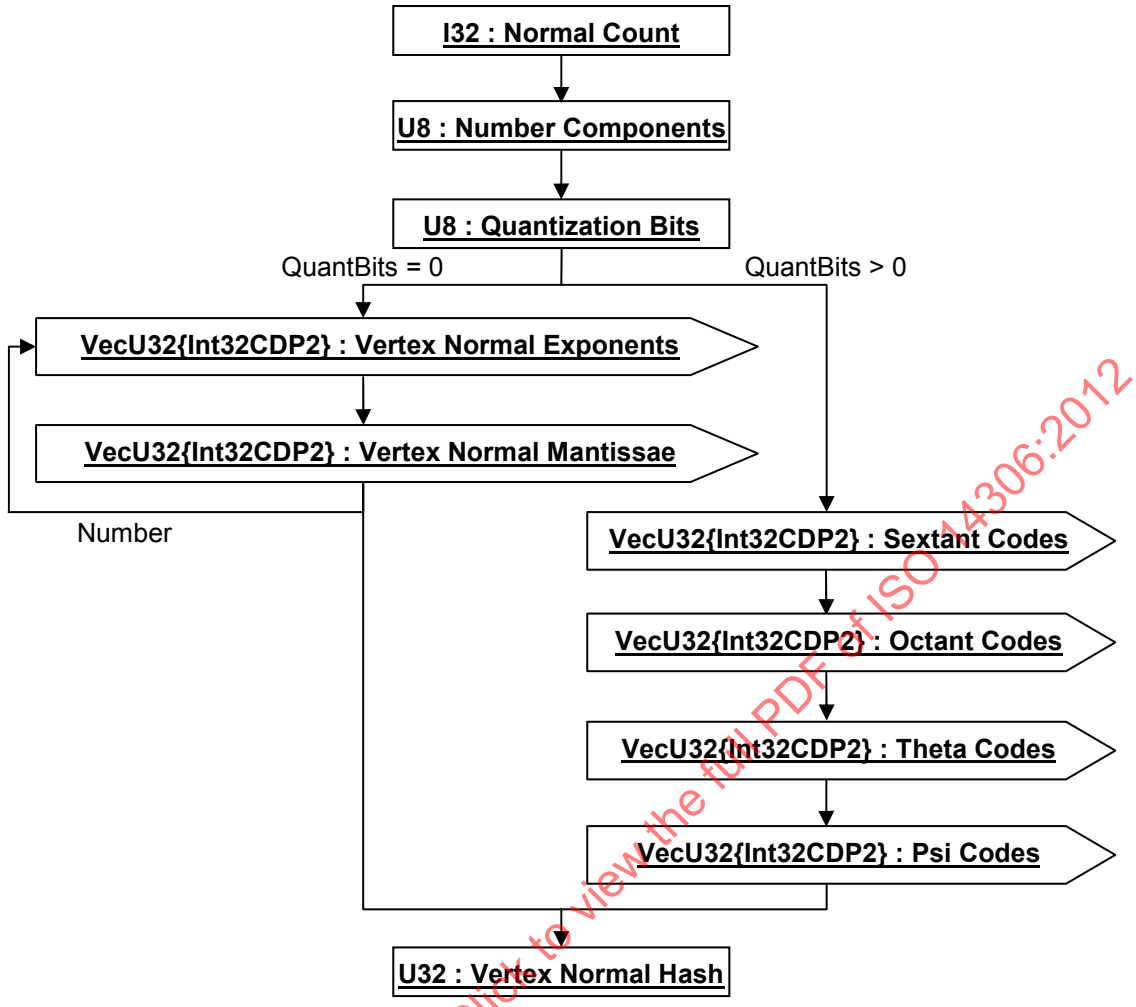


Figure 234 — Compressed Vertex Normal Array data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

I32 : Normal Count

Normal count specifies the number of normals. This number should equal the total number of vertex records.

U8 : Number Components

Number Components specifies the number of normal components present for each vertex record in the set of vertex records.

U8 : Quantization Bits

The number of bits used when the Deering Normal CODEC if quantization is enabled. A value of 0 denotes that quantization is disabled.

VecU32{Int32CDP2} : Vertex Normal Exponents

Vertex Normal Components is a vector of Floating Point Exponents for all the ith component values of a set of vertex coordinates. Vertex Normal Components uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Vertex Normal Mantissae

Vertex Normal Components is a vector of Floating Point Mantissae for all the ith component values of a set of vertex coordinates. Vertex Normal Components uses the Int32 version of the CODEC to compress and encode data.

© ISO 2012 – All rights reserved

VecU32{Int32CDP2} : Sextant Codes

Sextant Codes is a vector of “codes” (one per normal) for a set of normals identifying which Sextant of the corresponding sphere Octant each normal is located in. Sextant Codes uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Octant Codes

Octant Codes is a vector of “codes” (one per normal) for a set of normals identifying which sphere Octant each normal is located in. Octant Codes uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Theta Codes

Theta Codes is a vector of “codes” (one per normal) for a set of normals representing in Sextant coordinates the quantized theta angle for each normal’s location on the unit radius sphere; where theta angle is defined as the angle in spherical coordinates about the Y-axis on a unit radius sphere. Theta Codes uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Psi Codes

Psi Codes is a vector of “codes” (one per normal) for a set of normals representing in Sextant coordinates the quantized Psi angle for each normal’s location on the unit radius sphere; where Psi angle is defined as the longitudinal angle in spherical coordinates from the y = 0 plane on the unit radius sphere. Psi Codes uses the Int32 version of the CODEC to compress and encode data

U32 : Vertex Normal Hash

The Vertex Normal Hash is the combined hash of the vertex normals. If the number of quantization bits is equal to zero the hash value is equal to the combined hash of the vertex normal values for each of the component arrays. If the number of quantization bits is greater than 0 the hash value is equal to the combined hash of the Sextant, Octant, Theta, and Psi Codes for all vertex records. Refer to section 14.6 for a more detailed description on hashing.

```
UInt32 uHash   = 0;
uInt32 nVtxRec = 0;
vecF32 vNorm[nVtxRec][3];
vecU32 vSextant, vOctant. vTheta, vPsi;
...
if ( uQuantBits == 0 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
    for ( int j=0 ; j<nVtxRec ; j++) {
      uHash = hash32( (UInt32*)(&vNorm[j][i]), 1, uHash );
    }
  }
} else {
   uHash = hash32( &vSextant, nVtxRec, uHash );
   uHash = hash32( &vOctant, nVtxRec, uHash );
   uHash = hash32( &vTheta, nVtxRec, uHash );
   uHash = hash32( &vPsi, nVtxRec, uHash );
}
```

13.1.6 Compressed Vertex Texture Coordinate Array

The Compressed Vertex Texture Coordinate Array data collection contains the quantization data/representation for a set of vertex texture coordinates. Compressed Vertex Texture Coordinate Array data collection is only present if previously read vertex bindings denote texture coordinates are presents (See Vertex Shape LOD Data U64 : Vertex Bindings for complete explanation of the vertex bindings).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

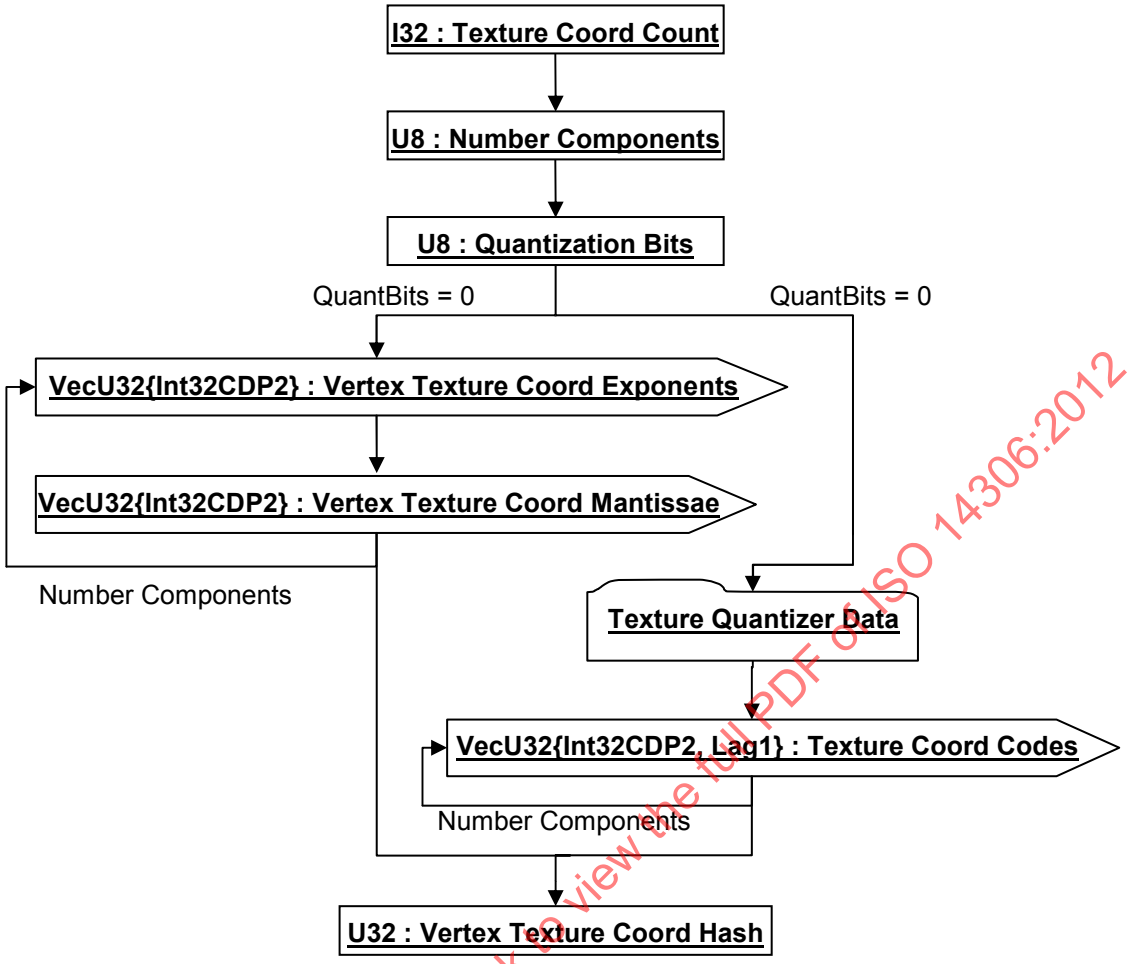


Figure 235 — Compressed Vertex Texture Coordinate Array data collection

Complete description for Texture Quantizer Data can be found in 13.1.10 Texture Quantizer Data.

I32 : Texture Coord Count

Colour count specifies the number of Texture Coordinates. This number should equal the total number of vertex records.

U8 : Number Components

Number Components specifies the number of Texture Coordinate components present for each vertex record in the set of vertex records.

U8 : Quantization Bits

Number of Bits specifies the quantized size (i.e. the number of bits of precision) for each of the components. The actual number of quantization bits used is specified within Texture Quantizer Data. Value shall be within range [0:24] inclusive.

VecU32{Int32CDP2} : Vertex Texture Coord Exponents

Vertex Texture Coordinate Components is a vector of Floating Point Exponents for all the ith component values of a set of vertex coordinates. Vertex Texture Coordinate Components uses the Int32 version of the CODEC to compress and encode data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecU32{Int32CDP2} : Vertex Texture Coord Mantissae

Vertex Texture Coordinate Components is a vector of Floating Point Mantissae for all the ith component values of a set of vertex coordinates. Vertex Texture Coordinate Components uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Texture Coord Codes

V-Texture Coord Codes is a vector of quantizer “codes” for all the nth-component of a set of vertex texture coordinates. V-Texture Coord Codes uses the Int32 version of the CODEC to compress and encode data.

U32 : Vertex Texture Coord Hash

The Vertex Texture Coord Hash is the combined hash of the Vertex Texture Coordinates. If the number of quantization bits is equal to zero the hash value is equal to the combined hash of the vertex texture coordinate values for each of the component arrays. If the number of quantization bits is greater than 0 the hash value is equal to the combined hash of the vertex texture coordinates codes for each of the component arrays. Refer to section 14.6 for a more detailed description on hashing.

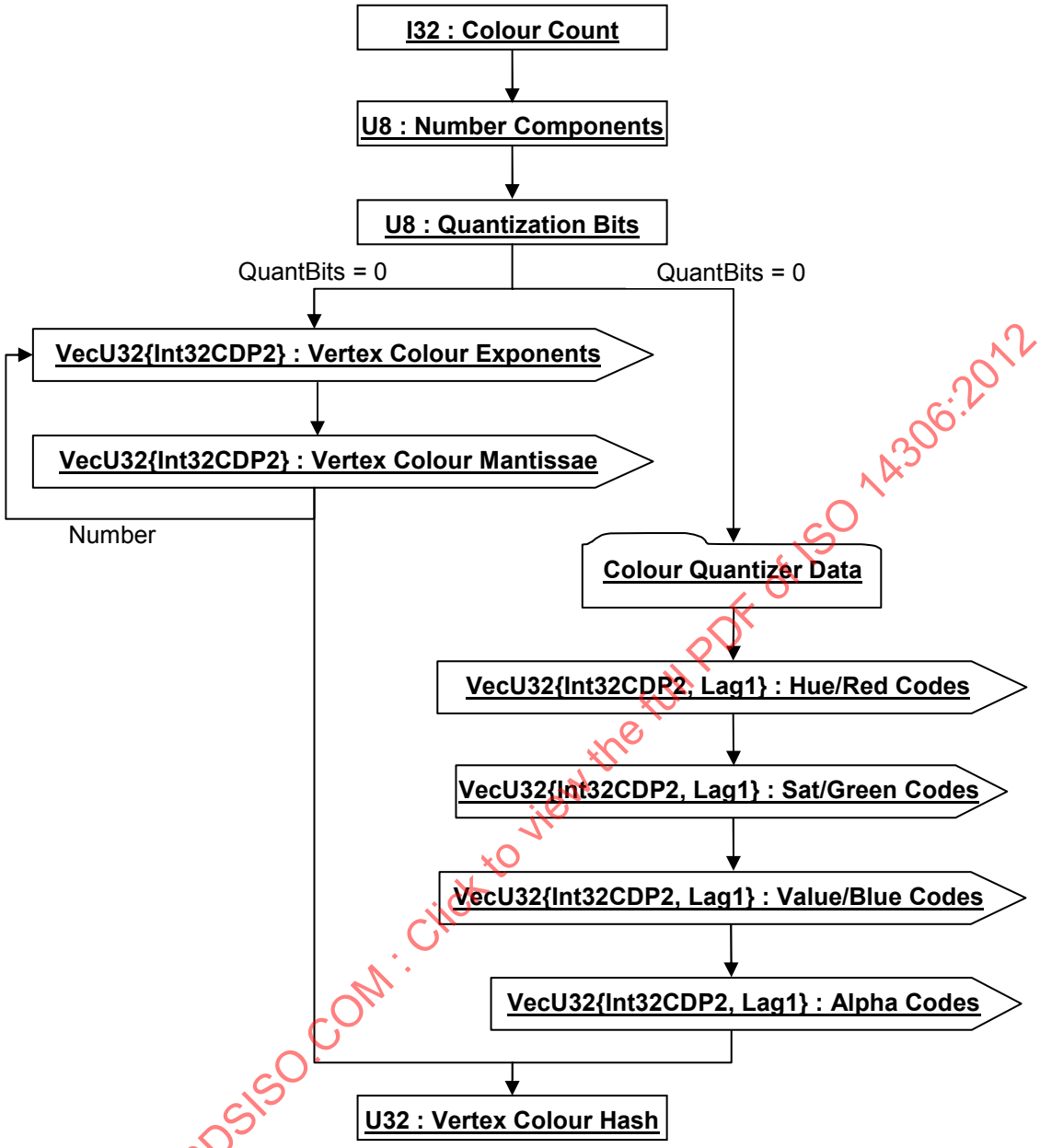
```
UInt32 uHash    = 0;
uInt32 nVtxRec = 0;
vecF32 vTexCoord[nVtxRec][4];
vecU32 vCodes[4];
...
if ( uQuantBits == 0 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
    for ( int j=0 ; j<nVtxRec ; j++) {
      uHash = hash32( (UInt32*)(&vTexCoord[j][i]), 1, uHash );
    }
  }
} else {
  for ( int i=0 ; i<nComp ; i++ ) {
   uHash = hash32( &vCodes[i], nVtxRec, uHash );
  }
}
```

13.1.7 Compressed Vertex Colour Array

The Compressed Vertex Colour Array data collection contains the quantization data/representation for a set of vertex colours. Compressed Vertex Colour Array data collection is only present if previously read Colour Binding value is not equal to zero (See Vertex Shape LOD Data for complete explanation of Colour Binding data field).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 236 — Compressed Vertex Colour Array data collection

Complete description for Colour Quantizer Data can be found in 13.1.11 Colour Quantizer Data.

I32 : Colour Count

Colour count specifies the number of colour records. This number should equal the total number of vertex records.

U8 : Number Components

Number Components specifies the number of Colour components present for each vertex record in the set of vertex records.

U8 : Quantization Bits

Number of Bits specifies the quantized size (i.e. the number of bits of precision) for each of the 3 or 4 colour components. This value shall satisfy the following condition: “0 <= Number Of Bits <= 8”.

© ISO 2012 – All rights reserved

VecU32{Int32CDP2} : Vertex Colour Exponents

Vertex Normal Components is a vector of Floating Point Exponents for all the ith component values of a set of vertex coordinates. Vertex Normal Components uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2} : Vertex Colour Mantissae

Vertex Normal Components is a vector of Floating Point Mantissae for all the ith component values of a set of vertex coordinates. Vertex Normal Components uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Hue/Red Codes

Hue/Red Codes is a vector of quantizer "codes" for all the Hue/Red colour components of a set of vertex colours. Hue/Red Codes uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Sat/Green Codes

Sat/Green Codes is a vector of quantizer "codes" for all the Saturation/Green colour components of a set of vertex colours. Sat/Green Codes uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Value/Blue Codes

Value/Blue Codes is a vector of quantizer "codes" for all the Value/Blue colour components of a set of vertex colours. Value/Blue Codes uses the Int32 version of the CODEC to compress and encode data.

VecU32{Int32CDP2, Lag1} : Alpha Codes

Alpha Codes is a vector of quantizer "codes" for all the Alpha colour components of a set of vertex colours. Alpha Codes uses the Int32 version of the CODEC to compress and encode data.

U32 : Vertex Colour Hash

The Vertex Colour Hash is the combined hash of the vertex colours. If the number of quantization bits is equal to zero the hash value is equal to the combined hash of the vertex colour values for each of the component arrays. If the number of quantization bits is greater than 0 the hash value is equal to the combined hash of the Hue/Red, Sat/Green, Value/Blue, and Alpha Codes for all vertex records. Refer to section 14.6 for a more detailed description on hashing.

```
UInt32 uHash    = 0;
uInt32 nVtxRec = 0;
vecF32 vCol[nVtxRec][3];
vecU32 vHue, vSat, vVal, vAlp;
...
if ( uQuantBits == 0 ) {
  for ( int i=0 ; i<nComp ; i++ ) {
    for ( int j=0 ; j<nVtxRec ; j++) {
      uHash = hash32( (UInt32*)(&vCol[j][i]), nVtxRec, uHash );
    }
  }
} else {
   uHash = hash32( &vHue, nVtxRec, uHash );
   uHash = hash32( &vSat, nVtxRec, uHash );
   uHash = hash32( &vVal, nVtxRec, uHash );
   uHash = hash32( &vAlp, nVtxRec, uHash );
}
```

13.1.8 Compressed Vertex Flag Array

The Compressed Vertex Flag Array data collection contains the quantization data/representation for per vertex flags. Compressed Vertex Flag Array data collection is only present if previously read Vertex Flag Binding value is not equal to zero.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

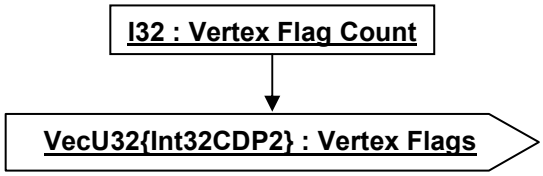


Figure 237 — Compressed Vertex Flag Array data collection

I32 : Vertex Flag Count

Vertex flag count specifies the number of vertex flags. This number should be equal to the total number of vertex records.

VecU32{Int32CDP2} : Vertex Flags

Vertex Flags is a vector of per vertex bit flags encoded as integers with valid values of either 0 (false) or 1 (true). Vertex Flags uses the Int32 version of the CODEC to compress and encode data.

13.1.9 Point Quantizer Data

A Point Quantizer Data collection is made up of three Uniform Quantizer Data collections; there is a separate Uniform Quantizer Data collection for the X, Y, and Z values of point coordinates.

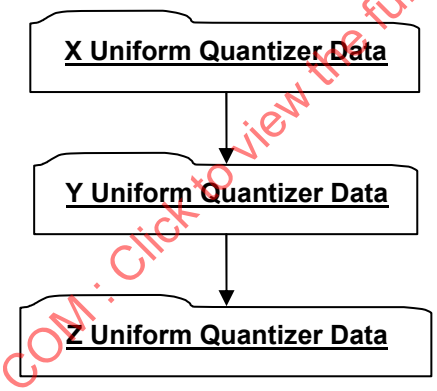


Figure 238 — Point Quantizer Data collection

Complete description for X Uniform Quantizer Data, Y Uniform Quantizer Data and Z Uniform Quantizer Data can be found in 13.1.12 Uniform Quantizer Data.

13.1.10 Texture Quantizer Data

A Texture Quantizer Data collection is made up of n Uniform Quantizer Data collections; there is a separate Uniform Quantizer Data collection for each component of the texture coordinates. The number of components is not specified within the quantizer, but rather is determined by the number of texture components present in the underlying data (See Compressed Vertex Texture Coordinate Arrays U8 : Number Components).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

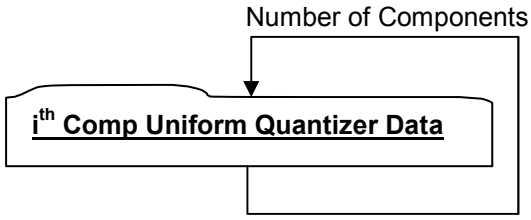


Figure 239 — Texture Quantizer Data collection

Complete description for U Uniform Quantizer Data, and V Uniform Quantizer Data can be found in 13.1.12 Uniform Quantizer Data.

13.1.11 Colour Quantizer Data

A Colour Quantizer Data collection contains the quantizer information for each of the colour components. The Colour Quantizer utilizes a separate Uniform Quantizer Data collection for each of the 4 colour components, but if the HSV colour model is being used, then it is not necessary to store a complete Uniform Quantizer Data Collection.

For the HSV model, since the range values for each colour component are constant, only the Number of Bits of precision for each colour component's Uniform Quantizer is stored. The Uniform Quantizer range values for the HSV colour components should always be assumed to be the following:

Table 104 — Colour Quantizer values

Component	Quantizer Range	
	Min	Max
Hue	0.0	6.0
Saturation	0.0	1.0
Value	0.0	1.0
Alpha	0.0	1.0

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

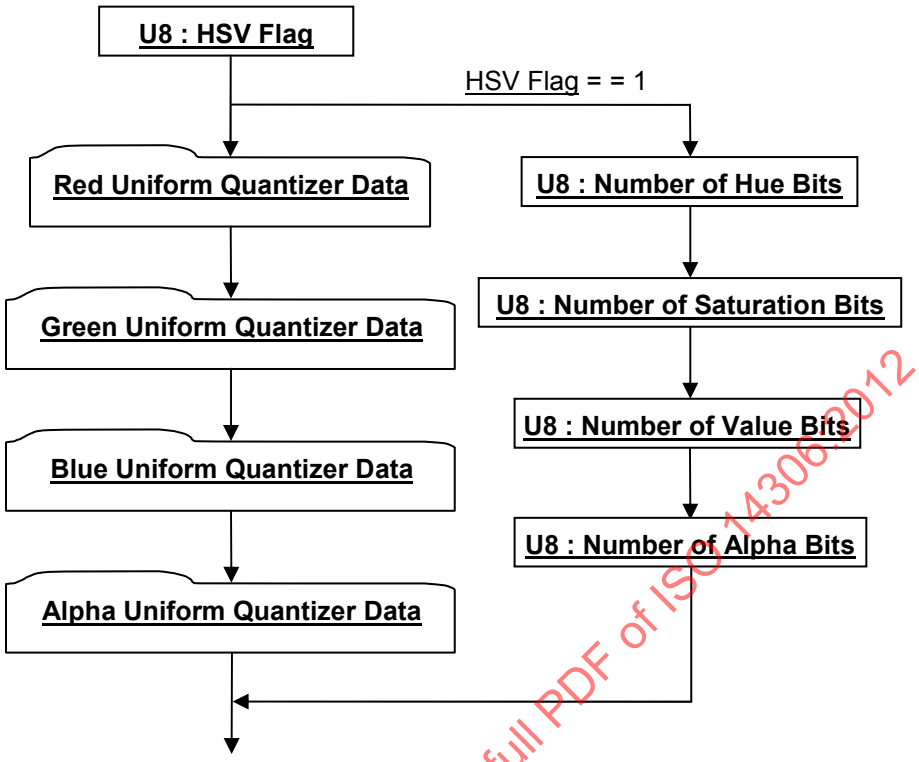


Figure 240 — Colour Quantizer Data collection

Complete descriptions for Red Uniform Quantizer Data, Green Uniform Quantizer Data, Blue Uniform Quantizer Data, and Alpha Uniform Quantizer Data can be found in 13.1.12 Uniform Quantizer Data. These four Uniform Quantizer Data collections are only present when data field HSV Flag = = 0.

U8 : HSV Flag

HSV Flag is a flag indicating whether colour component data is stored in HSV colour model form.

Table 105 — Colour Quantizer HSV Flag values

= 0	Colour component data stored in RGB colour model form.
= 1	Colour component data stored in HSV colour model form.

U8 : Number of Hue Bits

Number of Hue Bits specifies the quantized size (i.e. the number of bits of precision) for the Hue component of the colour. Number of Hue Bits data is only present when data field HSV Flag = = 1.

U8 : Number of Saturation Bits

Number of Saturation Bits specifies the quantized size (i.e. the number of bits of precision) for the Saturation component of the colour. Number of Saturation Bits data is only present when data field HSV Flag = = 1.

U8 : Number of Value Bits

Number of Value Bits specifies the quantized size (i.e. the number of bits of precision) for the Value component of the colour. Number of Value Bits data is only present when data field HSV Flag = = 1.

U8 : Number of Alpha Bits

Number of Alpha Bits specifies the quantized size (i.e. the number of bits of precision) for the Alpha component of the colour. Number of Alpha Bits data is only present when data field HSV Flag = = 1.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

13.1.12 Uniform Quantizer Data

The Uniform Quantizer Data collection contains information that defines a scalar quantizer/dequantizer (encoder/decoder) whose range is divided into levels of equal spacing.

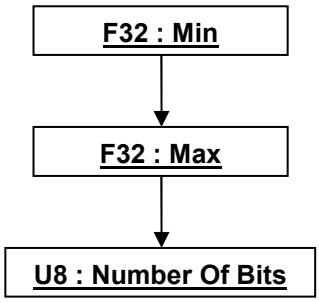


Figure 241 — Uniform Quantizer Data collection

F32 : Min

Min specifies the minimum of the quantized range.

F32 : Max

Max specifies the maximum of the quantized range.

U8 : Number Of Bits

Number of Bits specifies the quantized size (i.e. the number of bits of precision). In general, this value shall satisfy the following condition: "0 <= Number Of Bits <= 32".

13.1.13 Compressed Entity List for Non-Trivial Knot Vector

Compressed Entity List for Non-Trivial Knot Vector data collection specifies index identifiers (i.e. indices to particular entities within a list of entities) for a set of entities that contain Non-Trivial Knot Vectors. The entity types which can contain non-trivial knot vectors include:

JT B-Rep NURBS Surfaces

JT B-Rep PCS NURBS Curves

JT B-Rep MCS NURBS Curves

Wireframe MCS NURBS Curves

Note that any one occurrence of Compressed Entity List for Non-Trivial Knot Vector data collection will only contain index identifiers for one particular type of the above listed entities. The entity type is inferred based on the data collection which includes/references the Compressed Entity List for Non-Trivial Knot Vector.

A trivial knot vector is one which completely satisfies all conditions of at least one of the following cases:

Case-1 for trivial knot vector:

- Number of knots is an even number.
- Knot vector has a [0:1] knot range.
- There are no interior knots (i.e. NumberKnots = = 2 * (NurbsEntityDegree + 1).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Case-2 for trivial knot vector:

— Number of knots is an even number.

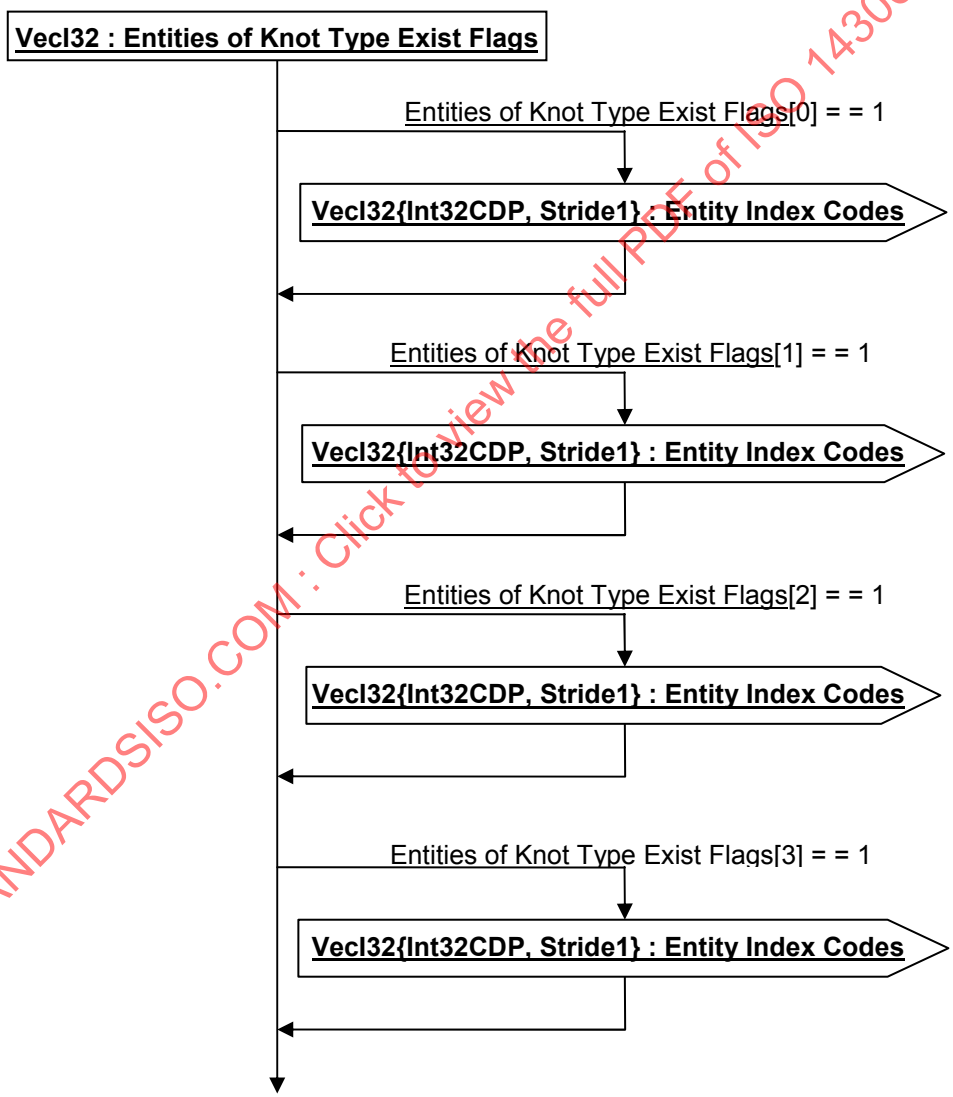
— Knot vector has a [0:1] knot range.

— NurbsEntityDegree < 3.

— Difference between successive non-repeating knots (i.e. KnotDelta) is:

— KnotDelta = 2.0 / (NumberKnots – (2.0 * NurbsEntityDegree)).

Any knot vector which does not satisfy one of the above cases for “trivial knot vector” is classified as a “non-trivial knot vector.”



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 242 — Compressed Entity List for Non-Trivial Knot Vector data collection

Vecl32 : Entities of Knot Type Exist Flags

Entities of Knot Type Exist Flags, is a vector of flags indicating for each knot vector type whether Entity Index ID data collections exist/follow for that knot vector type. Knot Vectors are categorized into types based on the

© ISO 2012 – All rights reserved

following characteristics: whether internal knots occur in *adjacent pairs* and whether the knot range is [0:1] or some other [x_1:x_2] range.

Currently there are four knot vector types, so this Entities of Knot Type Exist Flags vector should be of length four. The four flags have the following meaning:

Table 106 — Knot Type Exist Flag values

[0]	Flag indicating whether Entity IDs data collection exists for "Even Count [0:1] Range" knot type. Knots in this category have their knot range on [0:1], internal knots occur in *adjacent pairs*, *except* when there are no internal knots, in which case Type = 1 instead. = 0 – No Entity IDs data collection exists. = 1 – Entity IDs data collection exists.
[1]	Flag indicating whether Entity IDs data collection exists for "Even Count [x_1:x_2] Range" knot type. Knots in this category have their knot range on [x_1:x_2], and internal knots occur in *adjacent pairs*. = 0 – No Entity IDs data collection exists. = 1 – Entity IDs data collection exists.
[2]	Flag indicating whether Entity IDs data collection exists for "Odd Count [0:1] Range" knot type. Knots of this type have their knot range on [0:1], and are not Type 0. = 0 – No Entity IDs data collection exists. = 1 – Entity IDs data collection exists.
[3]	Flag indicating whether Entity IDs data collection exists for "Odd Count [x_1:x_2] Range" knot type. Knots of this type have their knot range on [x_1:x_2], and are not Type 1. = 0 – No Entity IDs data collection exists. = 1 – Entity IDs data collection exists.

Examples of knot vectors of Type 0:

```
0 0 X X 1 1
0 0 X X Y Y 1 1
0 0 X X Y Y Z Z 1 1
```

Examples of knot vectors of Type 1:

```
0 0 1 1          (Note: This is the exception to Type 0)
X X Y Y
X X Y Y Z Z
X X Y Y Z Z W W
```

Examples of knot vectors of Type 2:

```
0 0 X 1 1
0 0 X Y 1 1
0 0 X Y Z 1 1
0 0 X X X 1 1
0 0 X X Y Z Z 1 1
```

Examples of knot vectors of Type 3:

```
X X Y Z Z
X X Y Z W W
```

With this information in hand, the reader is able to reconstruct complete knot vectors in the following manner. When reconstructing the knot vector, you only take just enough values from the decoded knot value array. This may be as few as one. All the other values are inferred. Here's a sketch of the reconstruction algorithm:

```
// Number of knots in the knot vector
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
cNumKnots = numCtlPts + degree + 1;
// Necessary knot multiplicity at both ends of the knot vector
cClamping = degree + 1;
switch (knotType) {
    // Clamping is 0..1, internal knots occur in ADJACENT PAIRS
    // *EXCEPT* when there are no internal knots, in which case
    // Type = 1 instead.
    case 0: numVals = (cNumKnots - 2 * cClamping)/2;
    // Clamping is X1..X2, internal knots occur in ADJACENT PAIRS
    case 1: numVals = (cNumKnots - 2 * cClamping)/2 + 2;
    // Clamping is 0..1, and not Type 0
    case 2: numVals = (cNumKnots - 2 * cClamping);
    // Clamping is X1..X2, and not Type 1
    case 3: numVals = (cNumKnots - 2 * cClamping) + 2;
}
// numVals is the number of non-inferrable knot values needed
// Let vVals be the knot vector value array
// vKnot will be the final output knot vector
if (knotType is either 0 or 2)
    Set vKnot[0 .. cClamping-1] to 0
    Set vKnot[cNumKnots-cClamping .. cNumKnots-1] to 1
else
    Set vKnot[0 .. cClamping-1] to vVals[0]
    Set vKnot[cNumKnots-cClamping .. cNumKnots-1] to vVals[numVals-1]
Set vKnot[cClamping .. cNumKnots-cClamping-1] from vVals[1 .. numVals-2]
```

VecI32{Int32CDP, Stride1} : Entity Index Codes

Entity Index Codes is a vector of quantizer “codes” representing entity index identifiers for a set of entities (i.e. indices to particular entities within a list of entities). Entity Index Codes uses the Int32 version of the CODEC to compress and encode data.

13.1.14 Compressed Control Point Weights Data

Compressed Control Point Weights Data collection is the compressed and/or encoded representation of weight data for some set of Control Points. All NURBS based geometry use this data collection to compress/encode Control Point Weight data.

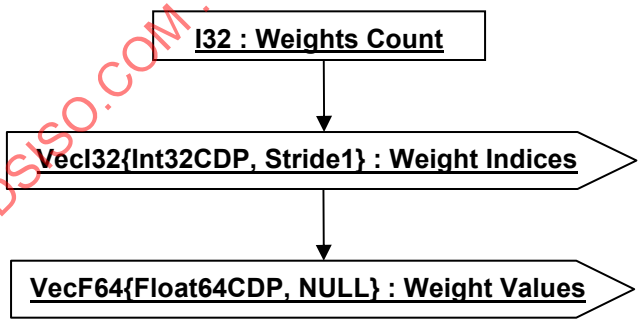


Figure 243 — Compressed Control Point Weights Data collection

I32 : Weights Count

Weights Count specifies the total number of Weights. This count can differ from the Control Point count (see 7.1.4.1.3 NURBS Surface Control Point Counts) because if the Control Point Dimensionality is non-rational (see data field NURBS Surface Control Point Dimensionality in Surfaces Geometric Data), then no Weight values are stored for the particular Control Point. Weights Count value also does not necessarily equate to the actual number of Weights stored, since if a particular Control Point’s Weight values is “1”, then no actual Weight value is stored (i.e. JT file loaders/readers can infer that the Weight Value is “1” for Control Points that don’t have a Weight value stored).

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

VecI32{Int32CDP, Stride1} : Weight Indices

Weight Indices is a vector of indices representing the index identifiers for the conditional set of weights for which an actual Weight Values is stored in Weight Values. Weight Indices uses the Int32 version of the CODEC to compress and encode data.

VecF64{Float64CDP, NULL} : Weight Values

Weight Values is a vector of weight values for the conditional set of weights. Weight Values uses the Float64 version of the CODEC to compress and encode data.

13.1.15 Compressed Curve Data

Compressed Curve Data collection contains JT B-Rep or Wireframe Rep compressed/encoded geometric Curve data. Currently only NURBS Curve types are supported as part of this data collection. Complete documentation for JT B-Rep and Wireframe Rep can be found in sections 7.1 JT B-Rep Element and 9.1 Wireframe Rep Element respectively.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

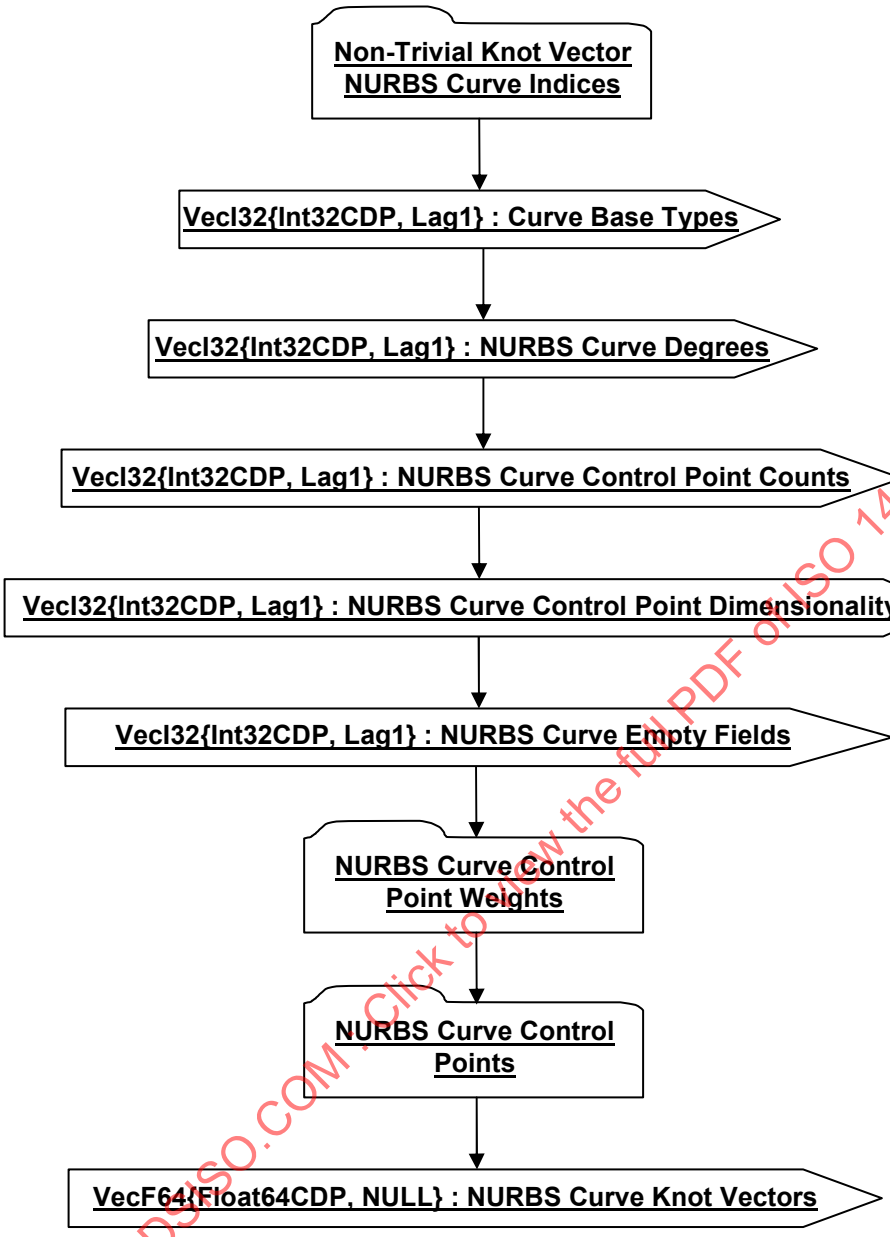


Figure 244 — Compressed Curve Data collection

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

VecI32{Int32CDP, Lag1} : Curve Base Types

Each Curve is assigned a base type identifier. Curve Base Types is a vector of base type identifiers for each Curve in a list of Curves. Currently only NURBS Curve Base Type is supported, but a type identifier is still included in the specification to allow for future expansion of the JT Format to support other curve types.

In an uncompressed/decoded form the Curves base type identifier values have the following meaning:

Table 107 — Compressed Curve Base Type values

= 1	Curve is a NURBS curve

Curve Base Types uses the Int32 version of the CODEC to compress and encode data.

© ISO 2012 – All rights reserved

VecI32{Int32CDP, Lag1} : NURBS Curve Degrees

NURBS Curve Degrees is a vector of Curve degree values for each NURBS Curve in a list of Curves (there is a stored value for each NURBS Curve in the list). NURBS Curve Degrees uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : NURBS Curve Control Point Counts

NURBS Curve Control Point Counts is a vector of control point counts for each NURBS Curve in a list of curves (there is a stored value for each NURBS Curve in the list). NURBS Curve Control Point Counts uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : NURBS Curve Control Point Dimensionality

NURBS Curve Control Point Dimensionality is a vector of control point dimensionality values for each NURBS Curve in a list of Curve s(i.e. there is a stored values for each NURBS Curve in the list).

In an uncompressed/decoded form the control point dimensionality values meaning is dependent upon the NURBS Entity type.

For NURBS UV Curve entities the dimensionality value has the following definition:

Table 108 — NURB UV Curve entity dimensionality values

= 2	Non-Rational (each control point has 2 coordinates)
= 3	Rational (each control point has 3 coordinates)

For NURBS XYZ Curve entities the dimensionality value has the following definition:

Table 109 — NURB XYZ Curve entity dimensionality values

= 3	Non-Rational (each control point has 3 coordinates)
= 4	Rational (each control point has 4 coordinates)

NURBS Curve Control Point Dimensionality uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP, Lag1} : NURBS Curve Empty Fields

NURBS Curve Empty Fields is a vector of data. Each NURBS Curve in a list of Curves has one reserved data field entry in this NURBS Curve Empty Fields vector. NURBS Curve Empty Fields uses the Int32 version of the CODEC to compress and encode data. Refer to best practice 14.4 Empty Field

VecF64{Float64CDP, NULL} : NURBS Curve Knot Vectors

NURBS Curve Knot Vectors is a list of knot vector values for each NURBS Curve having non-trivial knot vectors in a list of Curves (i.e. there are stored values for each non-trivial knot vector NURBS Curve in the list). All these NURBS Curve non-trivial knot vectors are accumulated into this single list in the same order as the Curve appears in the Curve list (i.e. Curve-N Non-Trivial Knot Vector, Curve-M Non-Trivial Knot Vector, etc.). The NURBS Curves for which knot vectors are stored (i.e. those containing non-trivial knot vectors) are identified in data collection Non-Trivial Knot Vector NURBS Curve Indices documented in Non-Trivial Knot Vector NURBS Curve Indices. NURBS Curve Knot Vectors uses the Float64 version of the CODEC to compress and encode data.

Non-Trivial Knot Vector NURBS Curve Indices

Non-Trivial Knot Vector NURBS Curve Indices data collection specifies the Curve index identifiers (i.e. indices to particular NURBS Curves within a list of Curves) for all NURBS Curves containing non-trivial knot vectors. A description/definition for “non-trivial knot vector” can be found in 13.1.13 Compressed Entity List for Non-Trivial Knot Vector.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

This Curve index data is stored in a compressed format.

Figure 245 — Non-Trivial Knot Vector NURBS Curve Indices data collection

Complete description for Compressed Entity List for Non-Trivial Knot Vector can be found in 13.1.13 Compressed Entity List for Non-Trivial Knot Vector.

NURBS Curve Control Point Weights

NURBS Curve Control Point Weights data collection defines the Weight values for a conditional set of Control Points for a list of NURBS Curves. The storing of the Weight value for a particular Control Point is conditional, because if NURBS Curve Control Point Dimension is "non-rational" or the actual Control Point's Weight value is "1", then no Weight value is stored for the Control Point (i.e. Weight value can be inferred to be "1").

The NURBS Curve Control Point Weights data is stored in a compressed format.

Figure 246 — NURBS Curve Control Point Weights data collection

Complete description for Compressed Control Point Weights Data can be found in 13.1.14 Compressed Control Point Weights Data.

NURBS Curve Control Points

NURBS Curve Control Points is the compressed and/or encoded representation of the Control Point coordinates for each NURBS Curve in a list of Curves (i.e. there are stored values for each NURBS Curve in the list). Note that these are non-homogeneous coordinates (i.e. Control Point coordinates have been divided by the corresponding Control Point Weight values).

Figure 247 — NURBS Curve Control Points data collection

VecF64{Float64CDP, NULL} : Control Points

Control Points is a vector of Control Point coordinates for all the NURBS Curves in a list of Curves. All the NURBS Curve Control Point coordinates are accumulated into this single vector in the same order as the Curve appears in the Curve list (i.e. Curve-1 Control Points, Curve-2 Control Points, etc.). Control Points uses the Float64 version of the CODEC to compress and encode data in a "lossless" manner.

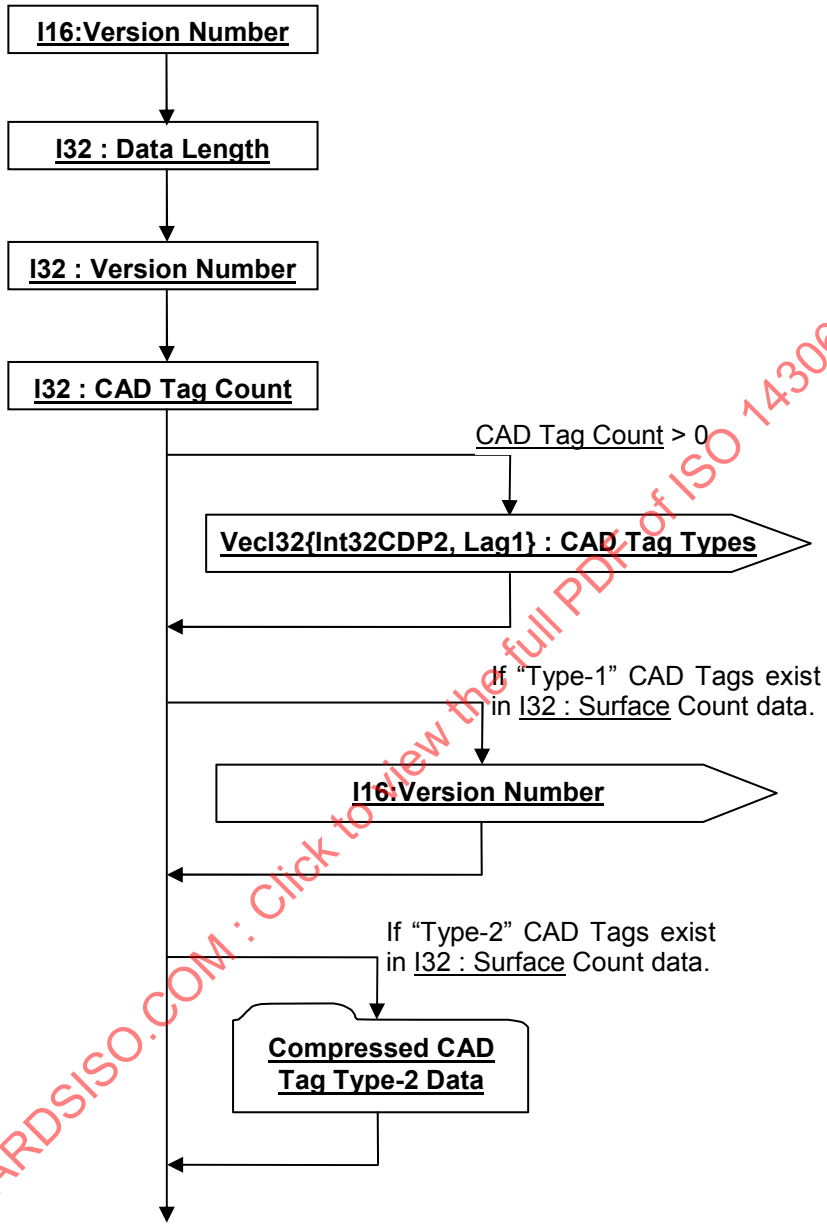
13.1.16 Compressed CAD Tag Data

The Compressed CAD Tag Data collection contains the persistent IDs, as defined in the CAD System, to uniquely identify individual CAD entities (e.g. Faces and Edges of a JT B-Rep, PMI, etc.). Exactly what CAD entity types have CAD Tags and what order they are stored in Compressed CAD Tag Data is defined by users of this data collection (e.g. 7.1.6 B-Rep CAD Tag Data, 10.2.7 PMI CAD Tag Data)

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

What constitutes a CAD Tag is outside the scope of the JT File format and is indeed part of the CAD system. The JT File format simply provides a way to store any kind of CAD Tag as provided by the CAD system which produced the CAD entity.



STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

Figure 248 — Compressed CAD Tag Data collection

I16:Version Number

Version Number is the version identifier for the CADTag element. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : Data Length

Data Length specifies the length in bytes of the Compressed CAD Tag Data collection. A JT file loader/reader may use this information to compute the end position of the Compressed CAD Tag Data within the JT file and thus skip reading the remaining Compressed CAD Tag Data.

© ISO 2012 – All rights reserved

I32 : Version Number

Version Number is the local version identifier for the Compressed CAD Tag Data. For informaiton on local version numbers see best practice 14.5 Local version numbers

I32 : CAD Tag Count

CAD Tag Count specifies the number of CAD Tags

VecI32{Int32CDP2, Lag1} : CAD Tag Types

CAD Tag Types is a vector of type identifiers for a list of CAD Tags (where each CAD Tag in the list has a type identifier value).

In an uncompressed/decoded form the CAD Tag type identifier values have the following meaning:

Table 110 — Compressed CAD Tag Type values

= 1	32 Bit Integer CAD Tag Type
= 2	64 Bit Integer CAD Tag Type

CAD Tag Types uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP2, Lag1} : CAD Tags Type-1

CAD Tags Type-1 is a vector of the Type-1 (i.e. 32 Bit Integer Type) CAD Tags for a list of CAD Tags. CAD Tags Type-1 uses the Int32 version of the CODEC to compress and encode data. CAD Tags Type-1 is only present if there are Type-1 CAD Tags in the CAD Tag Types vector. Thus a loader/reader of JT file shall first uncompress/decode and evaluate the previously read CAD Tag Types to determine if there are any Type-1 CAD Tags and if so, then the CAD Tags Type-1 data vector is present.

Compressed CAD Tag Type-2 Data

Compressed CAD Tag Type-2 Data collection contains the Type-2 (i.e. 64 Bit integer Type) CAD Tag data for a list of CAD Tags.

The Compressed CAD Tag Type-2 Data collection is only present if there are Type-2 CAD Tags in the CAD Tag Types vector. Thus a loader/reader of JT file shall first uncompress/decode and evaluate the previously read CAD Tag Types vector to determine if there are any Type-2 CAD Tags and if so, then the Compressed CAD Tag Type-2 Data collection is present.

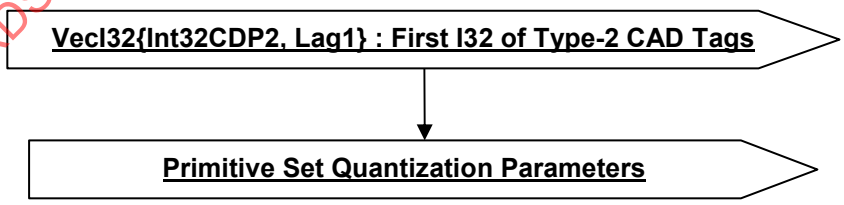


Figure 249 — Compressed CAD Tag Type-2 Data collection

VecI32{Int32CDP2, Lag1} : First I32 of Type-2 CAD Tags

First I32 of Type-2 CAD Tags is a vector of the first 32 bits of each Type-2 CAD Tag in the list of CAD Tags. First I32 Of Type-2 CAD Tags uses the Int32 version of the CODEC to compress and encode data.

VecI32{Int32CDP2, Lag1} : Second I32 of Type-2 CAD Tags

Second I32 of Type-2 CAD Tags is a vector of the second 32 bits of each Type-2 CAD Tag in the list of CAD Tags. Second I32 Of Type-2 CAD Tags uses the Int32 version of the CODEC to compress and encode data.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

13.2 Encoding Algorithms

The following sections give a brief technical overview/descriptions of the various encoding algorithms used in the JT format. A sample implementation of the encoding and decoding portion of each algorithm can be found in Annex B.

13.2.1 Uniform Data Quantization

Uniform Data Quantization is a lossy encoding algorithm in which a continuous set of input values (floating point data) is approximated with integral multiples (i.e. integers) of a common factor. How close the quantization output approximates the original input data is dependent upon the quantization data range and the number of bits specified to hold the resulting integer value.

The quantization is considered “uniform” because the algorithm divides the data input range into levels of equal spacing (i.e. a uniform scale). The form of Uniform Data Quantization used by the JT format is also considered scalar in nature, in that each input value is treated separately in producing the output integer value.

Given the following definitions:

```
inputVal:     Input floating point data to quantize
outputval:       Resulting quantized output integer value
minInputRange:   Specified minimum value of input data range
maxInputRange:   Specified maximum value of input data  range
nBits:        Specified number of bits of precision (quantized size)
```

The basic algorithm (using C++ style syntax) for Uniform Data Quantization is as follows:

```
UInt32 iMaxCode = (nBits < 32) ? (0x1 << nBits) - 1 : 0xffffffff;
Float64 encodeMultiplier = Float64(iMaxCode) / (maxInputRange – minInputRange);
UInt32 outputVal = UInt32( (inputVal - minInputRange) * encodeMultiplier + 0.5 );
```

Note: For reasons of robustness, “outputVal” shall also be explicitly clamped to the range [0,iMaxCode]. This is because floating-point roundoff error in the calculation of “encodeMultiplier” can otherwise cause “outputVal” to sometimes come out equal to “iMaxCode + 1”.

Note that all compression algorithms in the following sections operate on quantized integer data.

13.2.2 Bitlength CODEC

This is a very simple compression algorithm that runs an adaptive-width bit field encoding for each value. As each input value is encountered, the number of bits needed to represent it is calculated and compared to the current "field width". The current field width is then adjusted upwards or downwards by a constant “step_size” number of bits (i.e. 2 bits for the JT format) to accommodate the input value storage. This increment or decrement of the current field width is indicated for each encoded value by a prefix code stored with each value.

The prefix code will be one of the following two forms:

A single '0' bit to denote the same (i.e. current) field width is to be used for the next value.

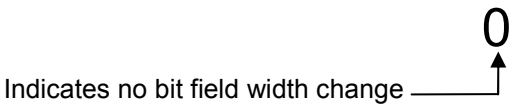
A '1' bit followed by a series of one or more bits where each bit indicates whether the field width is to be incremented (a '1' bit) or decremented (a '0' bit) by the field step_size, followed by a single terminator bit (which is complement of the previous increment/decrement bit). Note that there can only be increments or decrements in a given prefix code, never both, and that is why the prefix code terminator bit can be recognized as bits are read by simply looking for the complement of the previous increment/decrement bit.

Some examples of prefix codes and their interpretation are as follows:

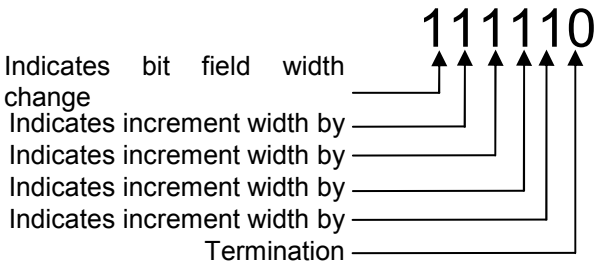
STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

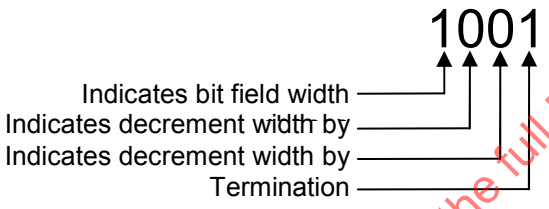
Example 1: Prefix code to maintain same (current) field width.



Example 2: Prefix code to increment field width four times (8 bits).



Example 3: Prefix code to decrement field width two times.



A pseudo-code sample implementation of bit length decoding is available in Annex C.

13.2.3 Arithmetic CODEC

In 1948, Claude Shannon of Bell Laboratories published his seminal paper “A mathematical theory of communication” that launched the new field of Information Theory. In that same year, two Doctoral students at the Massachusetts Institute of Technology (MIT) made breakthroughs in the coding of information. The first to press was David Huffman, whose coding scheme we now know as Huffman Coding. In that same class with Huffman was Peter Elias who reportedly developed the first articulation of arithmetic coding, but it lay unpublished until 1976, when Jorma Rissanen and Richard Pasco, of IBM, refined it into a practically useful algorithm.

Arithmetic encoding is a lossless compression algorithm that replaces an input stream of symbols or bytes with a single fixed point output number (i.e. only the mantissa bits to the right of the binary point are output from MSB to LSB). The total number of bits needed in the output number is dependent upon the length/complexity of the input message (i.e. the longer the input message the more bits needed in the output number). This single fixed point number output from an arithmetic encoding process shall be uniquely decodable to create the exact stream of input symbols that were used to create it.

Initially all symbols being encoded have a probability value assigned to them based on the likelihood that the symbol will occur next in the input stream (i.e. the frequency of the symbol in the input stream). Given probability value assignments, each individual symbol is then assigned an interval range along a nominal 0 to 1 “probability line”, where the size of each range corresponds to the symbol’s probability value. Note that a particular symbol owns all values within its assigned range up to, but not including, the range high value, and that it does not matter which symbols are assigned which segment of the range as long it is done in the same manner by both the encoder and the decoder.

Given the above described input stream probability and interval range assignments, a high level description of the arithmetic encoding process is as follows:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Begin with a “current interval” initialized to [0,1). Note, that in interval range notation (i.e. “[0,1)”), the “[“ symbol indicates inclusive of the interval low limit and “)” symbol indicates exclusive of the interval high limit.

Sequentially for each symbol of the input stream, perform two steps

Subdivide the current interval into subintervals based on the input stream symbol probability values as described above.

Select the subinterval corresponding to the current input stream symbol being sequentially processed and make it the new “current interval”.

After all input stream symbols have been sequentially processed; output enough bits to distinguish the final “current interval” from all other possible final intervals.

In pseudo code form, the algorithm to accomplish the above described arithmetic encoding for an input stream message of any length could look as follows:

```
Set low to 0.0
Set high to 1.0
While there are still input symbols do
    cur_symbol = get next input symbol
    range = high - low
    high = low + range * high_range(cur_symbol)
    low = low + range * low_range(cur_symbol)
End of While
Output low
```

So the arithmetic encoding process is simply one in which we narrow the range of possible numbers with every new sequentially processed input symbol; where the new narrowed range is proportional to the predefined probability values assigned to each symbol in the input stream.

The arithmetic decoding process is the inverse procedure, where the range is expanded in proportion to the probability of each symbol as it is extracted. For the arithmetic decoding process we find the first symbol in the message by seeing which symbol owns the interval range that our encoded message falls in. Then, since we know the low and high range limit values of the first symbol we can remove their effects by reversing the process that put them in.

In pseudo code form, the algorithm for decoding the incoming number could look as follows:

```
Get encoded_number
Do
    find symbol whose range straddles the encoded_number
    output the symbol
    range = symbol_high_value - symbol_low_value
    encoded_number = encoded_number - symbol_low_value
    encoded_number = encoded_number / range
until no more symbols
```

Example

Following is an example to demonstrate in practice the basic principles of arithmetic coding.

Suppose you want to compress, using arithmetic coding, the following sequence/array of integer data:

{2, 9, 12, 12, 0, 7, 1, 20, 5, 19}

For this input stream of data, the assigned probability values will be as follows:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 111 — Example assigned probability values

Number	Probability
0	1/10
1	1/10
2	1/10
5	1/10
7	1/10
9	1/10
12	2/10
19	1/10
20	1/10

Then based on each input numbers probability value, an interval range along a 0 to 1 "probability line" can be assigned to each input number as follows:

Table 112 — Example "probability line" values

Number	Probability	Range
0	1/10	[0.00, 0.10)
1	1/10	[0.10, 0.20)
2	1/10	[0.20, 0.30)
5	1/10	[0.30, 0.40)
7	1/10	[0.40, 0.50)
9	1/10	[0.50, 0.60)
12	2/10	[0.60, 0.80)
19	1/10	[0.80, 0.90)
20	1/10	[0.90, 1.00)

Now proceeding with encoding the example input integer sequence {2, 9, 12, 12, 0, 7, 1, 20, 5, 19}, the first number to be encoded is "2"; so the final encoded value will be a number that is greater than or equal to 0.20 and less than 0.30. Now as each subsequent number in the input stream is sequentially processed for encoding, the possible range of the output number is further restricted. In our example the next number to be encoded is "9" which owns the range [0.50, 0.60) within the new sub-range of [0.20, 0.30); which now further restricts our output number to the range [0.25, 0.26). If we continue this logic for the complete input integer sequence we end up with the following:

Table 113 — Example input integer sequence values

New integer number	Low value	High value
	0.0	1.0
2	0.2	0.3
9	0.25	0.26
12	0.256	0.258
12	0.2572	0.2576
0	0.25720	0.25724
7	0.257216	0.257220
1	0.2572164	0.2572168
20	0.25721676	0.2572168
5	0.257216772	0.257216776
19	**0.2572167752**	0.2572167756

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

From the above table, are final low values is “0.2572167752” which is the output number that uniquely encodes the integer number sequence {2, 9, 12, 12, 0, 7, 1, 20, 5, 19}.

Given this encoding scheme, the decoding would simply follow the process previously described. We find the first number in the sequence by looking up in the probability range for the value, whose range, our encoded number “0.2572167752” falls within. In our example this equates to the value “2” and so our first decoded value shall be “2”. Then we apply the previously described decoding subtraction and division steps to arrive at a new encoded value of “0.572167752”. Using this new “0.572167752” encoded value and the same logic of the first step, the second decoded value will be “9”. We continue this process until there are no more numbers to decode.

In practice, due to floating point size (i.e. number of bits) restrictions and possible differences in floating point formats on machines, arithmetic encoding is best implemented using 16 bit or 32 bit integer math. Using 16 bit or 32 bit integer math, an incremental transmission scheme can be implemented, where fixed size integer state variables receive new bits in at the low end and shift them out the high end, forming a single number that can be as many bits long as are available on the computer’s storage medium.

Using our example as a guide, define the starting range [0.0, 1.0) to instead be 0 to 0.999 (which is .111 in binary). Then in order to use integer registers to store these numbers, justify the values so that the implied decimal point is at the left hand side of the word. Now load the initial range values based on the word size we are using. In the case of a 16 bit implementation the initial range values will be low equals 0x0000 and high equals 0xFFFF. Since we know these values will go on forever (e.g. 0.999… will continue with FFs) we can shift those extra bits in as needed with no detrimental effects.

Going back to our example and using a 5 digit register, we start with the range:

High: 99999

Low: 00000

Applying the previously described encoding algorithm we first calculate the range between the low and high values; which in this case is 100000 (not 9999 since we assume the high value has an infinite number of 9’s). Next, we calculate the new high value which in this example will be 30000. But before we store the new high value we shall decrement it to account for the implied digits appended to it; so new high value will be 29999. Applying similar logic to computing the new low value results in a new range of:

High: 29999 (999…)

Low: 20000 (000…)

In looking at the newly computed high and low range values, it can be seen that the most significant digits of high and low match. A property of arithmetic coding is that as this encoding process continues, the high and low values will continue to get closer, but will never match exactly. Given this property, once the most significant digit of high and low match, it will never change, and thus we can output this most significant digit as the first number in the coded word and continue working with just 16 bit high and low values. This output process is accomplished by shifting both the high and low values left by one digit and shifting in a “9” in the least significant digit of the high value.

Applying the previously described encoding algorithm and continuing the above described process of shifting out most significant digit into the coded word as high and low continually grow closer together looks as follows for encoding our example integer number sequence {2, 9, 12, 12, 0, 7, 1, 20, 5, 19}:

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 114 — Example integer number sequence values

	High	Low	Range	Cumulative output
Initial State	99999	00000	100000	
Encode “2” [0.2, 0.3)	29999	20000		
Shift out 2	99999	00000	100000	.2
Encode “9” [0.5, 0.6)	59999	50000		.2
Shift out 5	99999	00000	100000	.25
Encode “12” [0.6, 0.8)	79999	60000	20000	.25
Encode “12” [0.6, 0.8)	75999	72000		.25
Shift out 7	59999	20000	40000	.257
Encode “0” [0.0, 0.1)	23999	20000		.257
Shift out 2	39999	00000	40000	.2572
Encode “7” [0.4, 0.5)	19999	16000		.2572
Shift out 1	99999	60000	40000	.25721
Encode “1” [0.1, 0.2)	67999	64000		.25721
Shift out 6	79999	40000	40000	.257216
Encode “20” [0.9, 1.0)	79999	76000		.257216
Shift out 7	99999	60000	40000	.2572167
Encode “5” [0.3, 0.4)	75999	72000		.2572167
Shift out 7	59999	20000	40000	.25721677
Encode “19” [0.8, 0.9)	55999	52000		.25721677
Shift out 5	59999	20000	40000	.257216775
Shift out 2				.2572167752
Shift out 0				.25721677520

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

As can be seen in the above table, after all values in the input stream have been encoded and any final matching most significant digit has been output, the arithmetic coding algorithm requires that two extra digits be shifted out of either the high or low value to finish up the cumulative output word.

Although the above example incrementally encodes very nicely with the arithmetic coding algorithm, there are certain cases where the computed high and low values get closer, but never actually converge to one value in the most significant digit (e.g. High = 0.300001, Low = 0.29992). Thus after a few iterations the difference between high and low becomes so small that 16 bits is not sufficient to represent any difference between the values (i.e. all calculations return the same values). This conditions is known as “underflow” and special logic shall added to the arithmetic coding algorithm to recognize that “underflow” is occurring and thus head it off before the computations reach an impasse.

The additional logic for recognizing that “underflow” is occurring would be executed after each recalculation of High and Low value set, and in pseudo code form this logic would look as follows:

underflow = FALSE

if((High and Low value’s significant digits don’t match but are on adjacent numbers) &&

(2^{nd} most significant digit of High is “0” and the 2^{nd} most significant digit of low is “9”))

{

underflow = TRUE

}

When/If it is identified that “underflow” is occurring, the encoding algorithm shall perform the following steps to stop the current “underflow”:

© ISO 2012 – All rights reserved

Delete the 2nd most significant digit from both the High and Low value.

Shift the other digits (those to the right of the deleted 2nd digit) to the left to fill up the space (note that the most significant digit stays in place).

Increment a counter to remember that we threw away a digit and don't know whether it was going to converge to "0" or "9".

A before and after example of performing the above steps to the High and Low values when 'underflow" occurs is as follows:

	Before	After
High	40344	43449
Low	39810	38100
Underflow_counter	0	1

Now as the encoding algorithm continues and the most significant digit of High and Low values once again converge to a common value, then that value shall be output to the coded word along with "Underflow_counter" number of "underflow" digits that were previously deleted. The underflow digits output to the coded word will either be all 9s or 0s, depending on whether the High and Low value converged to the higher or lower value.

A pseudo-code sample implementation of arithmetic decoding is available in Annex C.

13.2.4 Deering Normal CODEC

Michael Deering first published his work on geometry compression in 1995 [2] and later helped present a course on the subject at SIGGRAPH'99 [3]. Although Deering's approach to geometric compression involves compression of vertices, colours and normals, the description detailed here will focus solely on compression of normals since this is the only component of Deering's approach used in the JT format.

Through both theoretical examination and empirical testing, Deering found that an angular density of 0.01 radians between normals (about 100,000 normalized normals distributed over unit sphere) gave results that were not visually distinguishable from results obtained from finer normal representations. This observation reduced the problem of having to "exactly" represent any general surface normal, to only having to represent about 100,000 specific normals (i.e. general surface normal replaced by the appropriate one of the 100,000 specific normals).

If there were no run-time memory concerns and no concerns for on disk footprint size, these specific 100,000 normals could be simply represented in a table that is indexed into, to reference a particular normal. Instead, Deering's approach leverages symmetrical properties of the unit sphere to reduce the size of the table and allow any normal to be represented by, at max, an 18 bit index as summarized below:

- All normals are normalized (i.e. can be represented as points on the surface of the unit sphere).
- Unit sphere is divided into eight symmetrical octants based on sign bits of normal's X,Y,Z rectilinear representation (see Figure 250). Using three bits to represent the three sign bits of the normals XYZ components reduces the problem space to one eighth of the unit sphere.
- Each octant of the unit sphere is divided into six identical sextants by folding about the planes of symmetry; x=y, x=z, and y=z (see Figure 250). The particular sextant can be encoded using another three bits. So now unit sphere is divided into 48 identically shaped triangle patches reducing the normal look-up table to about 2000 entries (i.e. 100000/48).

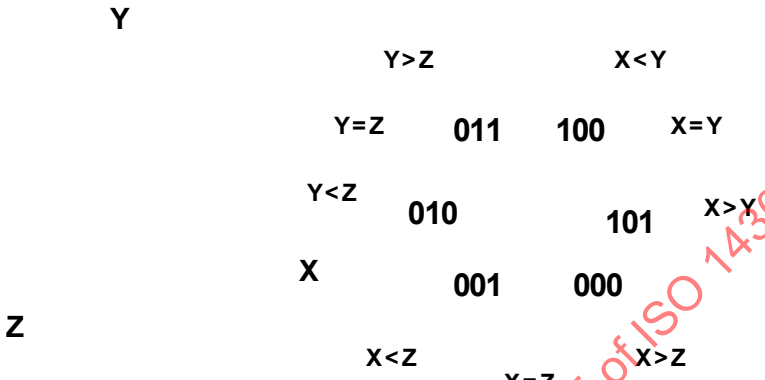
STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

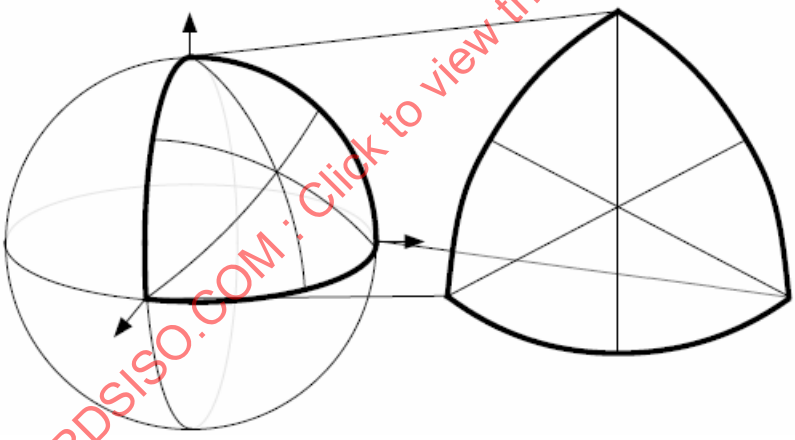
— Then, a local rectangular orthogonal two dimensional grid is created on the sextant and all normals within the sextant are represented as two n-bit angular addresses (i.e. a quantization of two angular values along the unit sphere) where “n” is in the range from 0 to 6 bits.

— Resulting in a max grand total of 18 bits (3 + 3 + 6 + 6) to represent any normal on the unit sphere.

In the figure below, the sphere is divided into eight octants and each octant is divided into six sextants. Each



sextant is assigned an identifying three bit code.



Note that th igure 250) are slightly modified fro

The represe summarized above, is based on th

— In sphe s, θ and φ; where θ is the ai

— Mappin

$$x = \cos\theta * \cos\varphi \qquad y = \sin\varphi \qquad z = \sin\theta * \cos\varphi.$$

— All encoding takes place in the positive octant.

— Angles θ and φ can be quantized into two n-bit integers θ'_n and φ'_n (where “n” is in the range of 0 to 6) and the relationship between these n-bit integers and angles θ and φ for a given “n” is:

$$\theta\,(\theta'_n) = \text{asin tan}\,(\varphi_{max} * (n - \theta'_n) / 2n\,)$$
$$\varphi\,(\varphi'_n) = \varphi_{max} * \varphi'_n / 2n.$$

Thus to encode (i.e. quantize) a given normal **N** into θ'_n and φ'_n:

— N shall be first represented (see Figure 250) in the positive octant and appropriate sextant within that octant, resulting in N’.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

— Then N' shall be dotted with all quantized normals in the sextant.

— For a fixed "n", the corresponding θ'n and φ'n values of the quantized sextant normal that result in the largest (nearest unity) dot product defines the proper θ'n and φ'n encoding of N.

With this encoding of normal **N** into θ'_n and φ'_n n-bit integers the complete bit representation of normal **N** can now be defined as follows:

— Uppermost three bits specify the octant.

— Next three bits specify the sextant code as defined in Figure 250.

— Next two n-bit fields specify θ'n and φ'n values respectively.

13.3 zlib compression

Zlib compression is a lossless data compression algorithm and is essentially the same as that in gzip and Zip. Zlib's compression method, called deflation, creates compressed data as a sequence of blocks. The JT format uses the following functionality from Version 1.1.2 of the zlib compression library. These provide the basic interface to zlib compression functionality. A link to the web location for Version 1.1.2 of the zlib compression library source code can be found in the bibliography of this document.

— zlibVersion();
— deflate();
— deflateReset();
— deflateEnd();
— inflateInit();
— inflate();
— inflateReset();
— inflateEnd();
— zlib_compress();
— zlib_compress2().

14 Best Practices

The proceeding sections of this International Standard specify the mandatory clauses for creating a reference compliant Version 9.5 JT file. This "Best Practices" section focusing on documenting format conventions that although not required to have a reference compliant JT file, have become commonplace within JT format translators to the point where these conventions are considered *best practices* for constructing JT files.

14.1 Late-Loading Data

The JT format was designed and structured to load entities from a JT file on a deferred or as-needed basis.. This concept is referred to within this International Standard as "late-loading data". The JT format has many structures in support of this and it is recommended as a best practice that writers/loaders of JT data leverage these capabilities.

Initial loading only requires the Table of Contents and the LSG. All Meta Data Node Elements, JT B-Rep Elements, XT B-Rep Elements, Wireframe Rep Elements, PMI Manager Meta Data Elements, JT ULP Elements, JT LWPA Elements, and Shape LOD Element

s may be ignored until they are actually needed. These Late-Loaded data containers are accessed via a Late Loaded Property Atom Element which appears in a LSG Node's Property list. Contained in this Property is

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

the GUID associated with the segment to be loaded. This GUID can be looked up in the TOC Segment, which will give the location in the JT from which to load the actual Element via the Data Segment convention.

14.2 TOC Segment Location

Although there are no JT format compliance rules about where the TOC Segment shall be located within the file, it is best placed immediately following the file header.

14.3 Bit Fields

All bits fields that are not defined as in use should be set to “0”.

14.4 Empty Field

In the 4 File Format section of this reference some data fields may be named/documented “Empty Field” (e.g. 5.1.1.7.1LOD Node Data ”Empty Field” field). Best practice suggests that these fields should be treated as follows:

If you are writing a JT file whose data did not originate from reading a previous JT file, then Empty Fields should be set to a value a “0” when writing the field to a JT file.

If you are writing a JT file whose data originated from reading a previous JT file (i.e. rewriting a JT File), then “Empty Fields” should be written with the same value that was read from the originating JT file.

14.5 Local version numbers

The version numbers seen throughout the data collections are version numbers local to those data types. They provide a simple means by which those data collections can be extended. All version information for 9.5 JT data is included within this document.

For each data collection, data for each local version should be written in sequence. When reading the data, the local version number allows readers to read up to the maximum local version they support and then use the segment length that was read in the Segment Header to skip over additional data. For example when version one and two data is present the user can choose to read only the version one data or additionally read the version two data as required.

Local version numbers are used for conditional branching as depicted in the element figures.

14.5.1 Version numbers

Local version number values defined at the publication of this standard are “0x0001” and “0x0002”. Listed below are the data collections that use local version numbers by value.

1. “0x0001” is the only valid value for Version 9.5 JT files for the listed data elements;

   5.1.1.1 Base Node Data,

   5.1.1.3.1 Group Node Data,

   5.1.1.4 Instance Node Element,

   5.1.1.5 Part Node Element,

   5.1.1.6.1 Meta Data Node Data,

   5.1.1.7.1 LOD Node Data,

   5.1.1.8 Range LOD Node Element,

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

5.1.1.9 Switch Node Element,

Base Shape Data,

5.1.1.10.7 NULL Shape Node Element,

5.1.1.10.8 Primitive Set Shape Node Element,

5.1.2.1.1 Base Attribute Data,

5.1.2.4 Draw Style Attribute Element,

5.1.2.5 Light Set Attribute Element.

5.1.2.6.1 Base Light Data.

5.1.2.8 Linestyle Attribute Element,

5.1.2.9 Pointstyle Attribute Element,

5.1.2.10 Geometric Transform Attribute Element,

5.1.2.11 Shader Effects Attribute Element,

5.1.2.12 Texture Coordinate Generator Attribute Element,

Mapping Plane Element,

Mapping Cylinder Element,

Mapping Sphere Element,

Mapping TriPlanar Element,

Base Property Atom Data,

5.2.2 String Property Atom Element,

5.2.3 Integer Property Atom Element,

5.2.4 Floating Point Property Atom Element,

5.2.5 JT Object Reference Property Atom Element,

5.2.6 Date Property Atom Element,

5.2.7 Late Loaded Property Atom Element,

5.2.8 Vector4f Property Atom Element,

5.3 Property Table,

6.1.1.1 Base Shape LOD Data,

6.1.2.1 Vertex Shape LOD Data,

6.1.2.5 TopoMesh Compressed Rep Data V2,

6.1.3 Tri-Strip Set Shape LOD Element,

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

6.1.4 Polyline Set Shape LOD Element,

6.1.5 Point Set Shape LOD Element,

6.1.6 Null Shape LOD Element,

6.2 Primitive Set Shape Element,

7.1 JT B-Rep Element,

10.1 Property Proxy Meta Data Element,

13.1.16 Compressed CAD Tag Data - I16:Version Number
10.2.8 PMI Polygon Data.

12.3 JT LWPA Element.

13.1.16 Compressed CAD Tag Data - I32 : Version Number.

9.1 Wireframe Rep Element

2. “0x0002” is the only valid value for Version 9.5 JT files for the listed data elements;;

   Vertex Shape Data,

   5.1.1.10.4 Polyline Set Shape Node Element,

   5.1.1.10.5 Point Set Shape Node Element.

3. “0x0001” and “0x0002” are valid values for:

   6.1.2.2 TopoMesh LOD Data

   6.1.2.2.1 TopoMesh Compressed LOD Data.

   6.1.2.2.2 TopoMesh Topologically Compressed LOD Data.

   10.2 PMI Manager Meta Data Element.

   12.1 JT ULP Element.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

14.6 Hash Value

Hashing is a means by which a large chunk of values can be represented by single value through the use of a mathematical function that provides a distinctive value for each unique set of ordered values. The hash function used within the v9.x format was published by Bob Jenkins in Dr Dobbs back in 1997 and its implementation is provided in Annex C.

The hash function takes a pointer to a set of values, the number of values, and a seed hash value. It returns the resulting hash value. Initially the seed value is set to 0, however when hashing multiple data fields together the hash of previous data field is used as the seed hash value of the next data field:

```
UInt32 uHash = 0;
uHash = hash32( pVal0, nVal0, uHash );
uHash = hash32( pVal1, nVal1, uHash );
```

The order that individual fields are hashed is extremely important since v9.x readers are strongly encouraged to assert that the stored hash value matches the calculated hash value of the corresponding fields after

© ISO 2012 – All rights reserved

reading in all the corresponding data. To this end each hash value stored within the v9.x format carefully documents which fields it encompasses and the order in which they should be hashed.

14.7 Scene graph construction

The following guidlines apply for scenegraph construction;

1. use a 5.1.1.6 Meta Data Node Element to denote a CAD “Assembly”,
2. use a 5.1.1.5 Part Node Element to denote a CAD “part”.

Below is an example of a fully-fleshed out small assembly of a three-wheeled motorcycle:

```
Partition              ( 5.1.1.2 Partition Node Element )          Root node of JT file
  MetaDataNode   ( 5.1.1.6 Meta Data Node Element )        “Three-wheeler” Top-level assembly
    Instance           ( 5.1.1.4 Instance Node Element )          “Front wheel” assembly
      Partition        ( 5.1.1.2 Partition Node Element )         External reference to JT file for “Wheel” part
        PartNode     ( 5.1.1.5 Part Node Element )              “Wheel” generic part
          RangeLOD   ( 5.1.1.8 Range LOD Node Element )         Level-of-detail node
                Group          ( 5.1.1.3 Group Node Element )              High-LOD group node
                  TriStripSet  ( 6.1.3 Tri-Strip Set Shape LOD Element ) Rim geometry
                  TriStripSet  ( 6.1.3 Tri-Strip Set Shape LOD Element ) Tire geometry
                  TriStripSet  ( 6.1.3 Tri-Strip Set Shape LOD Element ) Low-LOD geometry
            MetaDataNode    ( 5.1.1.6 Meta Data Node Element )        “Rear Axle” assembly
              Instance           ( 5.1.1.4 Instance Node Element )          “Left rear wheel” assembly
                [“Wheel” PartNode above]        This instance node’s child is same as the one above.
              Instance           ( 5.1.1.4 Instance Node Element )          “Right rear wheel” assembly
                [“Wheel” PartNode above]        This instance node’s child is same as the one above.
```

5.1.1.4 Instance Node Element are used when referring to an *instanced* part but are not fundamentally different from a 5.1.1.3 Group Node Element having only one child.

14.8 Metadata Conventions

Although there are really no restrictions/limits/requirements on what metadata (i.e. properties) can/shall be attached to nodes in the LSG in order to have a reference compliant JT file, there are some conventions that have been generally followed in the industry when translating CAD data to the JT file format. See Property Atom Elements section of this International Standard for complete description of the file Elements used to attach this property information to nodes.

14.8.1 CAD Properties

The following table lists the conventions that CAD data translators typically (although not always) follow in placing CAD information in a JT file as properties on various LSG nodes. Some of these properties are considered required in order for the data in the file to be interpreted correctly while other properties are optional. See flowing sub-sections for additional information on required versus optional properties.

The convention is to place these Units properties on every Part and Assembly grouping node in the LSG. By following this convention, JT file format readers/writers are provided maximum flexibility in understanding/indicating the appropriate JT data unit processing for both, monolithic and shattered JT file Assembly structures.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 115 — CAD Property Conventions

JT Property Key	Meaning	JT File Data Type	Encoded Data Type	Valid Values	Required / Optional
JT_PROP_MEASUREMENT_UNITS	Model Units	MbString	MbString	millimeters centimeters meters inches feet yards micrometers decimeters kilometers mils miles	Required
CAD_MASS_UNITS	Units of mass	MbString	MbString	micrograms milligrams grams kilograms ounces pounds	Required
CAD_SURFACE_AREA	Surface area of solids within part.	MbString	F64	numeric	Optional
CAD_VOLUME	Volume of solids within part	MbString	F64	numeric	Optional
CAD_DENSITY	Density of solids within part (6)	MbString	F64	numeric	Optional
CAD_MASS	Mass or weight of solids within part	MbString	F64	numeric	Optional
CAD_CENTRE_OF_GRAVITY	Centre of gravity of solids within part	MbString	3 space separated F64	3 numeric values	Optional
CAD_PROP_MATERIAL_THICKNESS	Sheet thickness within part	MbString	F64	numeric	Optional
CAD_PART_NAME	Component name from translator	MbString	MbString	<string>	Optional
CAD_SOURCE	CAD program the Part originated from	MbString	MbString	<string>	Optional

14.8.1.1 Required Properties

The required unit properties are really necessary for viewers of JT file data to properly interpret numeric data for analysis operations (e.g. measure) and support the building of assemblies through the reading of multiple JT files in disparate units. There are two units of measure that are relevant, units of distance and units of weight.

The JT_PROP_MEASURMENT_UNITS property is used to specify units of distance. The CAD_MASS_UNITS property is used to specify units for weight. JT_PROP_MEASURMENT_UNITS property is strictly required, while CAD_MASS_UNITS property is "optionally required". By “optionally required”, we mean, it is required if other optional metadata intends to specify properties that would depend on these units of measure (e.g. CAD_DENSITY and CAD_MASS). Notice that the Mass units are specified, instead of the Density units, since Density is a derived unit of Mass/Volume.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

14.8.1.2 Optional Properties

Optional properties can be provided, but if the property is a units based value, then the value shall be in units that are consistent with the JT_PROP_MEASURMENT_UNITS and CAD_MASS_UNITS properties. Thus the units for the optional units based properties shall be as follows:

Table 116 — CAD Optional Property Units

Optional Property	Units
CAD_SURFACE_AREA	$(JT_PROP_MEASUREMENT_UNITS)^2$
CAD_VOLUME	$(JT_PROP_MEASUREMENT_UNITS)^3$
CAD_DENSITY	$CAD_MASS_UNITS/(JT_PROP_MEASUREMENT_UNITS)^3$
CAD_MASS	CAD_MASS_UNITS
CAD_CENTRE_OF_GRAVITY	JT_PROP_MEASUREMENT_UNITS
CAD_PROP_MATERIAL_THICKNESS	JT_PROP_MEASUREMENT_UNITS

Note of caution regarding the node placement for the CAD_DENSITY property. Following the recommended convention for the placing of CAD properties (see description in 14.8.1 CAD Properties) implies that all solids within a single JT part are of a uniform density, which may not be true in all cases.

14.8.2 PMI Properties

The table below lists properties that can be found in JT parts that contain Generic PMI Entities (see 10.2.6 Generic PMI Entities). The column titled Supported PMI_Type values defines the expected Generic PMI Entity Types the property will be found with (reference Enitity Type). The column Value Format / Expected Values defines the expected representation of the property data. The Property Key and Value Format / Expected Values data forms a key value pair for a standard JT property (see PMI Property Atom).

Table 123 – PMI Properties makes use of C programming string format specifiers in description of values supported by the Property Key. The following values are used
"%d" - Signed 32-bit integer
"%.16g" - 64-bit floating-point number with 16 digit precision
"%s" - Null-terminated array of 8-bit unsigned characters

Note: The recommended best practice for working with PMI in a JT file is to use the descriptions found in 10.2 PMI Manager Meta Data Element as opposed to working with PMI Data Segment.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Table 117 — PMI Properties

Property Key	Supported PMI_Type values	Value Format / Expected Values	Prerequisite Key	Additional Note
accountabilityId	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		
aftDistance	0x0242	"%.16g"	type	Expects type property to be closed
aftThroughAll	0x0242	"%d"	type	Expects type property to be closed
allAround	0x0041 0x0081 0x0008	"%d"		
allOver	0x0081	"%d"		
AltText[%d]	0x0100 0x0235 0x0244	NULL		
AltText[%d].bold	0x0100 0x0235 0x0244	"%d"	AltText[%d]	
AltText[%d].font	0x0100 0x0235 0x0244	"%s"	AltText[%d]	
AltText[%d].italic	0x0100 0x0235 0x0244	"%d"	AltText[%d]	
AltText[%d].italicAngle	0x0100 0x0235 0x0244	"%.16g"	AltText[%d]	
AltText[%d].Item[%d]	0x0100 0x0235 0x0244	"%s"	AltText[%d]	
AltText[%d].Item[%d].bold	0x0100 0x0235 0x0244	"%d"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].font	0x0100 0x0235 0x0244	"%s"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].format	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	AltText[%d].Item[%d]	
AltText[%d].Item[%d].italic	0x0100 0x0235 0x0244	"%d"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].italicAngle	0x0100 0x0235 0x0244	"%.16g"	AltText[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

AltText[%d].Item[%d].justification	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	AltText[%d].Item[%d]	
AltText[%d].Item[%d].language	0x0100 0x0235 0x0244	"%s"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].lineFactor	0x0100 0x0235 0x0244	"%.16g"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].outlineType	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	AltText[%d].Item[%d]	
AltText[%d].Item[%d].spaceFactor	0x0100 0x0235 0x0244	"%.16g"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].string	0x0100 0x0235 0x0244	"%s"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].symbol	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc,	AltText[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		"17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface		
AltText[%d].Item[%d].textAspect	0x0100 0x0235 0x0244	"%.16g"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].textColour	0x0100 0x0235 0x0244	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	AltText[%d].Item[%d]	
AltText[%d].Item[%d].textHeight	0x0100 0x0235 0x0244	"%.16g"	AltText[%d].Item[%d]	
AltText[%d].Item[%d].textThickness	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	AltText[%d].Item[%d]	
AltText[%d].Item[%d].underline	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=over, "1"=under	AltText[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

AltText[%d].justification	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	AltText[%d]	
AltText[%d].language	0x0100 0x0235 0x0244	"%s"	AltText[%d]	
AltText[%d].lineFactor	0x0100 0x0235 0x0244	"%.16g"	AltText[%d]	
AltText[%d].name	0x0100 0x0235 0x0244	"%s"	AltText[%d]	
AltText[%d].outlineType	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	AltText[%d]	
AltText[%d].spaceFactor	0x0100 0x0235 0x0244	"%.16g"	AltText[%d]	
AltText[%d].textAspect	0x0100 0x0235 0x0244	"%.16g"	AltText[%d]	
AltText[%d].textColour	0x0100 0x0235 0x0244	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	AltText[%d]	
AltText[%d].textHeight	0x0100 0x0235 0x0244	"%.16g"	AltText[%d]	
AltText[%d].textOrientation	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=horizontal, "1"=vertical	AltText[%d]	
AltText[%d].textThickness	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	AltText[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

AltText[%d].underline	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=over, "1"=under	AltText[%d]	
angle	0x0306	"%.16g"		
angleFormat	0x0082 0x0081	String representing enumeration such that: "0"=degrees, "1"=seconds, "2"=minutes		
angleNumerator	0x0082	"%.16g"		
approachDirection	0x0004	"%.16g %.16g %.16g"		
Area[%d]	0x0246	NULL		
Area[%d].ConstraintPlane.origin	0x0246	"%.16g %.16g %.16g"	Area[%d].ConstraintPlane	
Area[%d].ConstraintPlane.xaxis	0x0246	"%.16g %.16g %.16g"	Area[%d].ConstraintPlane	
Area[%d].ConstraintPlane.zaxis	0x0246	"%.16g %.16g %.16g"	Area[%d].ConstraintPlane	
Area[%d].curveRef[%d]	0x0246	"%d %d %d %s" such that first in is the dst id, second the dst type, third the reason(optional) and the string the path. This property formating is used to semantically tie a singlar semantic aspect with an entity.	Area[%d]	
Area[%d].diameter	0x0246	"%.16g"	Area[%d]	
Area[%d].height	0x0246	"%.16g"	Area[%d]	
Area[%d].innerDiameter	0x0246	"%.16g"	Area[%d]	
Area[%d].insidePoint	0x0246	"%.16g %.16g %.16g"	Area[%d]	
Area[%d].length	0x0246	"%.16g"	Area[%d]	
Area[%d].originAnchor	0x0246	String representing enumeration such that: "0"=topleft, "1"=topcentre, "2"=topright, "3"=middleleft, "4"=middlecentre, "5"=middleright, "6"=bottomleft, "7"=bottomcentre, "8"=bottomright	Area[%d]	
Area[%d].type	0x0246	String representing enumeration such that: "0"=rectangular, "1"=circular, "2"=annular,	Area[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		"3"=cylindrical, "4"=general		
Area[%d].width	0x0246	"%.16g"	Area[%d]	
areaShape	0x0088	String representing enumeration such that: "0"=diameter, "1"=sphericaldiameter, "2"=square		
arrowSideFinishSymbol	0x0008	String representing enumeration such that: "0"=unset, "1"=unspecified, "2"=machining, "3"=grinding, "4"=chipping, "5"=none, "6"=hammering, "7"=peening, "8"=rolling		
arrowSideGrooveAngle	0x0008	"%s"		
arrowSideGrooveGap	0x0008	"%s"		
arrowSideLongitudinalSize	0x0008	"%s"		
arrowSideMainSize	0x0008	"%s"		
arrowSideStaggeredSize	0x0008	"%s"		
arrowSideSupplementalSymbol	0x0008	String representing enumeration such that: "0"=unset, "1"=convex, "2"=toesblended, "3"=concave, "4"=flush		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

arrowSideSymbol	0x0008	String representing enumeration such that: "0"=unset, "1"=jisseam, "2"=singlebevelgroove, "3"=singleugroove, "4"=edgeflange, "5"=surface, "6"=steepflankedsinglebevel, "7"=flaresinglevgroove, "8"=surfacejoint, "9"=singlevgroove, "10"=inclinedjoint, "11"=plug, "12"=fillet, "13"=squaregroove, "14"=spot, "15"=singlej, "16"=square, "17"=overlay, "18"=singleu, "19"=singlejgroove, "20"=doubleflange, "21"=singleflange, "22"=spotprojected, "23"=plugandslot, "24"=singlebevelbroadrootface, "25"=edge, "26"=jisstaggeredfillet1, "27"=singlev, "28"=flaresinglebevelgroove, "29"=singlevbroadrootface, "30"=steepflankedsinglev, "31"=removablebackingstrip, "32"=permanentbackingstrip, "33"=jisfillet, "34"=singlebevel, "35"=backing, "36"=foldjoint, "37"=jisstaggeredfillet2, "38"=seam, "39"=bead		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

attachmentType	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=angulardimension, "1"=ongeometry, "2"=lineardimension, "3"=leader, "4"=stacked, "5"=ordinatedimension, "6"=radialdimension, "7"=diametraldimension, "8"=noleader		
AttributeName[%d]	0x0240 0x0241	"%s"		
axisDirection	0x0230 0x0238	"%.16g %.16g %.16g"		
balloonId	0x0235	"%s"		
basic	0x0082	"%d"		
blanked	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		
bold	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		
bottomExtensionLine	0x0041	"%d"		
category	0x0100 0x0235 0x0244	"%s"		
causality	0x0082	String representing enumeration such that: "0"=key, "1"=functional, "2"=reference, "3"=associated, "4"=pmi		
centrecrossSize	0x0306	"%.16g"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

centrelineType	0x0306	String representing enumeration such that: "0"=centreline2d, "1"=centreline3d, "2"=centremark, "3"=circularcentreline, "4"=boltcirclecentreline, "5"=symmetriccentreline, "6"=offsetcentreline		
changeLevel	0x0118	String representing enumeration such that: "0"=soft, "1"=medium, "2"=hard		
characteristic	0x0081	String representing enumeration such that: "0"=profileofaline, "1"=circularrunout, "2"=perpendicularity, "3"=position, "4"=totalrunout, "5"=profileofasurface, "6"=cylindricity, "7"=symmetry, "8"=angularity, "9"=parallelism, "10"=concentricity, "11"=flatness, "12"=circularity, "13"=straightness		
clampingDirection	0x0004	"%.16g %.16g %.16g"		
CLIP_CAPPING	0x0305	"%d"		
CLIP_CAPPING_COLOUR	0x0305	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string		
CLIP_CURVE	0x0305	"%d"		
CLIP_CURVE_COLOUR	0x0305	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string		
CLIP_NORMAL	0x0305	"%.16g,%.16g,%.16g"		
CLIP_POSITION	0x0305	"%.16g,%.16g,%.16g"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CLIP_RENDER	0x0305	String representing enumeration such that: "0"=shaded, "1"=wireframe		
CLIP_STYLE	0x0305	Enumerated string supporting values "Both", "Near", "Far" and "None"		
commaAsDecimal	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		
conformsToFaces	0x0246	"%d"		
ConstraintPlane.origin	0x0242	"%.16g %.16g %.16g"	ConstraintPlane	
ConstraintPlane.xaxis	0x0242	"%.16g %.16g %.16g"	ConstraintPlane	
ConstraintPlane.zaxis	0x0242	"%.16g %.16g %.16g"	ConstraintPlane	
CoordinatedEntity[%d]	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	NULL		
CoordinatedEntity[%d].bold	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%d"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].font	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].italic	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%d"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].italicAngle	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].Item[%d]	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].Item[%d].bold	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%d"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].font	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CoordinatedEntity[%d].Item[%d].format	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].italic	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%d"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].italicAngle	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].justification	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].language	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].lineFactor	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].outlineType	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].spaceFactor	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].string	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CoordinatedEntity[%d].Item[%d].symbol	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].textAspect	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CoordinatedEntity[%d].Item[%d].textColour	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].textHeight	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].textThickness	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].Item[%d].underline	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=over, "1"=under	CoordinatedEntity[%d].Item[%d]	
CoordinatedEntity[%d].justification	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	CoordinatedEntity[%d]	
CoordinatedEntity[%d].language	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].lineFactor	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].name	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%s"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].outlineType	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	CoordinatedEntity[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CoordinatedEntity[%d].spaceFactor	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].textAspect	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].textColour	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	CoordinatedEntity[%d]	
CoordinatedEntity[%d].textHeight	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	"%.16g"	CoordinatedEntity[%d]	
CoordinatedEntity[%d].textOrientation	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=horizontal, "1"=vertical	CoordinatedEntity[%d]	
CoordinatedEntity[%d].textThickness	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	CoordinatedEntity[%d]	
CoordinatedEntity[%d].underline	0x0230 0x0004 0x0008 0x0042 0x0118 0x0238	String representing enumeration such that: "0"=over, "1"=under	CoordinatedEntity[%d]	
coordinatePlane	0x0118	String representing enumeration such that: "0"=y, "1"=xz, "2"=xy, "3"=yz, "4"=z, "5"=xyz, "6"=x		
coordinationDirection	0x0042	"%.16g %.16g %.16g"		
counterbored	0x0230	"%d"		
CpsLetters	0x0242	NULL		
CpsLetters.bold	0x0242	"%d"	CpsLetters	
CpsLetters.font	0x0242	"%s"	CpsLetters	
CpsLetters.italic	0x0242	"%d"	CpsLetters	
CpsLetters.italicAngle	0x0242	"%.16g"	CpsLetters	
CpsLetters.Item[%d]	0x0242	"%s"	CpsLetters	
CpsLetters.Item[%d].bold	0x0242	"%d"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].font	0x0242	"%s"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].format	0x0242	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list,	CpsLetters.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		"10"=datetime		
CpsLetters.Item[%d].italic	0x0242	"%d"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].italicAngle	0x0242	"%.16g"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].justification	0x0242	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	CpsLetters.Item[%d]	
CpsLetters.Item[%d].language	0x0242	"%s"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].lineFactor	0x0242	"%.16g"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].outlineType	0x0242	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	CpsLetters.Item[%d]	
CpsLetters.Item[%d].spaceFactor	0x0242	"%.16g"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].string	0x0242	"%s"	CpsLetters.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CpsLetters.Item[%d].symbol	0x0242	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	CpsLetters.Item[%d]	
CpsLetters.Item[%d].textAspect	0x0242	"%.16g"	CpsLetters.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CpsLetters.Item[%d].textColour	0x0242	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	CpsLetters.Item[%d]	
CpsLetters.Item[%d].textHeight	0x0242	"%.16g"	CpsLetters.Item[%d]	
CpsLetters.Item[%d].textThickness	0x0242	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	CpsLetters.Item[%d]	
CpsLetters.Item[%d].underline	0x0242	String representing enumeration such that: "0"=over, "1"=under	CpsLetters.Item[%d]	
CpsLetters.justification	0x0242	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	CpsLetters	
CpsLetters.language	0x0242	"%s"	CpsLetters	
CpsLetters.lineFactor	0x0242	"%.16g"	CpsLetters	
CpsLetters.name	0x0242	"%s"	CpsLetters	
CpsLetters.outlineType	0x0242	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	CpsLetters	
CpsLetters.spaceFactor	0x0242	"%.16g"	CpsLetters	
CpsLetters.textAspect	0x0242	"%.16g"	CpsLetters	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

CpsLetters.textColour	0x0242	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	CpsLetters	
CpsLetters.textHeight	0x0242	"%.16g"	CpsLetters	
CpsLetters.textOrientation	0x0242	String representing enumeration such that: "0"=horizontal, "1"=vertical	CpsLetters	
CpsLetters.textThickness	0x0242	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	CpsLetters	
CpsLetters.underline	0x0242	String representing enumeration such that: "0"=over, "1"=under	CpsLetters	
CrosshatchPattern.angle	0x0243 0x0246	"%.16g"	CrosshatchPattern	
CrosshatchPattern.pattern	0x0243 0x0246	String representing enumeration such that: "0"=unset, "1"=rubber, "2"=lead, "3"=user, "4"=steel, "5"=brass, "6"=solid, "7"=iron, "8"=aluminum, "9"=glass, "10"=electricalwinding, "11"=refractory, "12"=thermalinsulation	CrosshatchPattern	
CrosshatchPattern.rotate	0x0243 0x0246	"%d"	CrosshatchPattern	
CrosshatchPattern.spacing	0x0243 0x0246	"%.16g"	CrosshatchPattern	
Curve[%d]	0x0305	"%d %d %d %s" such that first in is the dst id, second the dst type, third the reason(optional) and the string the path. This property formating is used to semantically tie a singlar semantic aspect with an entity.		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

curveRef	0x0242	"%d %d %d %s" such that first in is the dst id, second the dst type, third the reason(optional) and the string the path. This property formating is used to semantically tie a singlar semantic aspect with an entity.	type	Expects type property to not be plane
datumOnLeader	0x0081 0x0084 0x0100 0x0235 0x0244	String representing enumeration such that: "0"=none, "1"=solid, "2"=filled		
delta	0x0004 0x0008	"%d"		
depth[%d]	0x0230	"%.16g"		
Description	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
descriptiveModifier	0x0233	"%s"		
deviation	0x0082	"%s"		
diameter	0x0088 0x0004 0x0238	"%.16g"		
diameter[%d]	0x0230	"%.16g"		
diameterOverride	0x0238	"%d"		
diameterPlacement	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after		
dimensionLeadingZero	0x0082	"%d"		
DimensionText[%d]	0x0082	NULL		DimensionText[%d] index expected to range from 0-3 inclusive.Each instance of this is expected to have a different position such that DimensionText[x].position != DimensionText[y].position for any combination of x and y

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DimensionText[%d].position	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after	DimensionText[%d]	
DimensionText[%d].Text	0x0082	NULL	DimensionText[%d]	
DimensionText[%d].Text.bold	0x0082	"%d"	DimensionText[%d].Text	
DimensionText[%d].Text.font	0x0082	"%s"	DimensionText[%d].Text	
DimensionText[%d].Text.italic	0x0082	"%d"	DimensionText[%d].Text	
DimensionText[%d].Text.italicAngle	0x0082	"%.16g"	DimensionText[%d].Text	
DimensionText[%d].Text.Item[%d]	0x0082	"%s"	DimensionText[%d].Text	
DimensionText[%d].Text.Item[%d].bold	0x0082	"%d"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].font	0x0082	"%s"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].format	0x0082	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].italic	0x0082	"%d"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].italicAngle	0x0082	"%.16g"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].justification	0x0082	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].language	0x0082	"%s"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].lineFactor	0x0082	"%.16g"	DimensionText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DimensionText[%d].Text.Item[%d].outlineType	0x0082	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].spaceFactor	0x0082	"%.16g"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].string	0x0082	"%s"	DimensionText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DimensionText[%d].Text.Item[%d].symbol	0x0082	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].textAspect	0x0082	"%.16g"	DimensionText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DimensionText[%d].Text.Item[%d].textColour	0x0082	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].textHeight	0x0082	"%.16g"	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].textThickness	0x0082	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.Item[%d].underline	0x0082	String representing enumeration such that: "0"=over, "1"=under	DimensionText[%d].Text.Item[%d]	
DimensionText[%d].Text.justification	0x0082	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	DimensionText[%d].Text	
DimensionText[%d].Text.language	0x0082	"%s"	DimensionText[%d].Text	
DimensionText[%d].Text.lineFactor	0x0082	"%.16g"	DimensionText[%d].Text	
DimensionText[%d].Text.name	0x0082	"%s"	DimensionText[%d].Text	
DimensionText[%d].Text.outlineType	0x0082	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	DimensionText[%d].Text	
DimensionText[%d].Text.spaceFactor	0x0082	"%.16g"	DimensionText[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DimensionText[%d].Text.textAspect	0x0082	"%.16g"	DimensionText[%d].Text	
DimensionText[%d].Text.textColour	0x0082	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	DimensionText[%d].Text	
DimensionText[%d].Text.textHeight	0x0082	"%.16g"	DimensionText[%d].Text	
DimensionText[%d].Text.textOrientation	0x0082	String representing enumeration such that: "0"=horizontal, "1"=vertical	DimensionText[%d].Text	
DimensionText[%d].Text.textThickness	0x0082	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	DimensionText[%d].Text	
DimensionText[%d].Text.underline	0x0082	String representing enumeration such that: "0"=over, "1"=under	DimensionText[%d].Text	
dimensionTrailingZero	0x0082	"%d"		
direction	0x0082 0x0041 0x0081	"%.16g %.16g %.16g"		
dislayClearance	0x0307	"%d"		
displayOption	0x0306	String representing enumeration such that: "0"=none, "1"=centreline, "2"=centrelinewithextension, "3"=centrepoint		
displayTolerance	0x0307	"%d"		
displayType	0x0230 0x0004 0x0100 0x0235 0x0244 0x0042 0x0238	String representing enumeration such that: "0"=urface, "1"=creen		
documentation	0x0082	"%d"		
DualDimensionText[%d]	0x0082	NULL		DualDimensionText[%d] index expected to range from 0-3 inclusive.Each instance of this is expected to have a different position such that DualDimensionText[x].position != DualDimensionText[y].position for any combination of x and y

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DualDimensionText[%d].position	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after	DualDimensionText[%d]	
DualDimensionText[%d].Text	0x0082	NULL	DualDimensionText[%d]	
DualDimensionText[%d].Text.bold	0x0082	"%d"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.font	0x0082	"%s"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.italic	0x0082	"%d"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.italicAngle	0x0082	"%.16g"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.Item[%d]	0x0082	"%s"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.Item[%d].bold	0x0082	"%d"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].font	0x0082	"%s"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].format	0x0082	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].italic	0x0082	"%d"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].italicAngle	0x0082	"%.16g"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].justification	0x0082	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].language	0x0082	"%s"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].lineFactor	0x0082	"%.16g"	DualDimensionText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DualDimensionText[%d].Text.Item[%d].outlineType	0x0082	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].spaceFactor	0x0082	"%.16g"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].string	0x0082	"%s"	DualDimensionText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DualDimensionText[%d].Text.Item[%d].symbol	0x0082	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].textAspect	0x0082	"%.16g"	DualDimensionText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DualDimensionText[%d].Text.Item[%d].textColour	0x0082	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].textHeight	0x0082	"%.16g"	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].textThickness	0x0082	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.Item[%d].underline	0x0082	String representing enumeration such that: "0"=over, "1"=under	DualDimensionText[%d].Text.Item[%d]	
DualDimensionText[%d].Text.justification	0x0082	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.language	0x0082	"%s"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.lineFactor	0x0082	"%.16g"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.name	0x0082	"%s"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.outlineType	0x0082	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	DualDimensionText[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

DualDimensionText[%d].Text.spaceFactor	0x0082	"%.16g"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.textAspect	0x0082	"%.16g"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.textColour	0x0082	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.textHeight	0x0082	"%.16g"	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.textOrientation	0x0082	String representing enumeration such that: "0"=horizontal, "1"=vertical	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.textThickness	0x0082	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	DualDimensionText[%d].Text	
DualDimensionText[%d].Text.underline	0x0082	String representing enumeration such that: "0"=over, "1"=under	DualDimensionText[%d].Text	
dualLowerDeltaDenominator	0x0082	"%d"		
dualPlacement	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after		
dualPrecision	0x0082	"%d"		
dualTolerancePrecision	0x0082	"%d"		
dualType	0x0082	String representing enumeration such that: "0"=positional, "1"=bracket		
dualUnits	0x0082	"%s"		
dualUnitText	0x0082	"%d"		
dualUpperDeltaDenominator	0x0082	"%d"		
dualValueDenominator	0x0082	"%d"		
fastenerSubType	0x0230	"%s"		
FCF[%d].	0x0308	"%.16g %.16g %.16g"	FCF[%d]	
FCF[%d].Text.bold	0x0308	"%d"	FCF[%d].Text	
FCF[%d].Text.font	0x0308	"%s"	FCF[%d].Text	
FCF[%d].Text.italic	0x0308	"%d"	FCF[%d].Text	
FCF[%d].Text.italicAngle	0x0308	"%.16g"	FCF[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

FCF[%d].Text.Item[%d]	0x0308	"%s"	FCF[%d].Text	
FCF[%d].Text.Item[%d].bold	0x0308	"%d"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].font	0x0308	"%s"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].format	0x0308	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].italic	0x0308	"%d"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].italicAngle	0x0308	"%.16g"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].justification	0x0308	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].language	0x0308	"%s"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].lineFactor	0x0308	"%.16g"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].outlineType	0x0308	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	FCF[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

FCF[%d].Text.Item[%d].spaceFactor	0x0308	"%.16g"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].string	0x0308	"%s"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item[%d].symbol	0x0308	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeat	FCF[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		ure, "50"=spotface		
FCF[%d].Text.Item [%d].textAspect	0x0308	"%.16g"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item [%d].textColour	0x0308	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item [%d].textHeight	0x0308	"%.16g"	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item [%d].textThickness	0x0308	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	FCF[%d].Text.Item[%d]	
FCF[%d].Text.Item [%d].underline	0x0308	String representing enumeration such that: "0"=over, "1"=under	FCF[%d].Text.Item[%d]	
FCF[%d].Text.justification	0x0308	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	FCF[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

FCF[%d].Text.language	0x0308	"%s"	FCF[%d].Text	
FCF[%d].Text.lineFactor	0x0308	"%.16g"	FCF[%d].Text	
FCF[%d].Text.name	0x0308	"%s"	FCF[%d].Text	
FCF[%d].Text.outlineType	0x0308	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	FCF[%d].Text	
FCF[%d].Text.spaceFactor	0x0308	"%.16g"	FCF[%d].Text	
FCF[%d].Text.textAspect	0x0308	"%.16g"	FCF[%d].Text	
FCF[%d].Text.textColour	0x0308	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	FCF[%d].Text	
FCF[%d].Text.textHeight	0x0308	"%.16g"	FCF[%d].Text	
FCF[%d].Text.textOrientation	0x0308	String representing enumeration such that: "0"=horizontal, "1"=vertical	FCF[%d].Text	
FCF[%d].Text.textThickness	0x0308	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	FCF[%d].Text	
FCF[%d].Text.underline	0x0308	String representing enumeration such that: "0"=over, "1"=under	FCF[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.bold	0x0308	"%d"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.font	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.italic	0x0308	"%d"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.italicAngle	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].bold	0x0308	"%d"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].font	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].format	0x0308	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].italic	0x0308	"%d"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].ital	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

icAngle				
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].justification	0x0308	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].language	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].lineFactor	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].outlineType	0x0308	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].spaceFactor	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].string	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].symbol	0x0308	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency,	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		"49"=continuousfeature, "50"=spotface		
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textAspect	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textColour	0x0308	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textHeight	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].tex	0x0308	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

tThickness				
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].underline	0x0308	String representing enumeration such that: "0"=over, "1"=under	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.justification	0x0308	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.language	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.lineFactor	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.name	0x0308	"%s"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.outlineType	0x0308	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.spaceFactor	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textAspect	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textColour	0x0308	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textHeight	0x0308	"%.16g"	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textOrientation	0x0308	String representing enumeration such that: "0"=horizontal, "1"=vertical	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textThickness	0x0308	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.underline	0x0308	String representing enumeration such that: "0"=over, "1"=under	FCF[%d].ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	
fieldWeld	0x0008	"%d"		
filename	0x0114	"%s"		
fitGrade	0x0082	"%d"		
flag	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

font	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
foreDistance	0x0242	"%.16g"	type	Expects type property to be closed
foreThroughAll	0x0242	"%d"	type	Expects type property to be closed
fraction	0x0082	"%d"		
fractionSize	0x0082	String representing enumeration such that: "0"=full, "1"=twothirds, "2"=threequarters		
FreeState	0x0082	NULL		
fullCircle	0x0306	"%d"		
functionalSubscript	0x0118	"%s"		
GeneralText	0x0230	NULL		
GeneralText.bold	0x0230	"%d"	GeneralText	
GeneralText.font	0x0230	"%s"	GeneralText	
GeneralText.italic	0x0230	"%d"	GeneralText	
GeneralText.italicAngle	0x0230	"%.16g"	GeneralText	
GeneralText.Item[%d]	0x0230	"%s"	GeneralText	
GeneralText.Item[%d].bold	0x0230	"%d"	GeneralText.Item[%d]	
GeneralText.Item[%d].font	0x0230	"%s"	GeneralText.Item[%d]	
GeneralText.Item[%d].format	0x0230	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	GeneralText.Item[%d]	
GeneralText.Item[%d].italic	0x0230	"%d"	GeneralText.Item[%d]	
GeneralText.Item[%d].italicAngle	0x0230	"%.16g"	GeneralText.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

GeneralText.Item[%d].justification	0x0230	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	GeneralText.Item[%d]	
GeneralText.Item[%d].language	0x0230	"%s"	GeneralText.Item[%d]	
GeneralText.Item[%d].lineFactor	0x0230	"%.16g"	GeneralText.Item[%d]	
GeneralText.Item[%d].outlineType	0x0230	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset "20"=scoredrectangle, "21"=parallelogram	GeneralText.Item[%d]	
GeneralText.Item[%d].spaceFactor	0x0230	"%.16g"	GeneralText.Item[%d]	
GeneralText.Item[%d].string	0x0230	"%s"	GeneralText.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

GeneralText.Item[%d].symbol	0x0230	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	GeneralText.Item[%d]	
GeneralText.Item[%d].textAspect	0x0230	"%.16g"	GeneralText.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

GeneralText.Item[%d].textColour	0x0230	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	GeneralText.Item[%d]	
GeneralText.Item[%d].textHeight	0x0230	"%.16g"	GeneralText.Item[%d]	
GeneralText.Item[%d].textThickness	0x0230	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	GeneralText.Item[%d]	
GeneralText.Item[%d].underline	0x0230	String representing enumeration such that: "0"=over, "1"=under	GeneralText.Item[%d]	
GeneralText.justification	0x0230	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	GeneralText	
GeneralText.language	0x0230	"%s"	GeneralText	
GeneralText.lineFactor	0x0230	"%.16g"	GeneralText	
GeneralText.name	0x0230	"%s"	GeneralText	
GeneralText.outlineType	0x0230	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	GeneralText	
GeneralText.spaceFactor	0x0230	"%.16g"	GeneralText	
GeneralText.textAspect	0x0230	"%.16g"	GeneralText	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

GeneralText.textColour	0x0230	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	GeneralText	
GeneralText.textHeight	0x0230	"%.16g"	GeneralText	
GeneralText.textOrientation	0x0230	String representing enumeration such that: "0"=horizontal, "1"=vertical	GeneralText	
GeneralText.textThickness	0x0230	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	GeneralText	
GeneralText.underline	0x0230	String representing enumeration such that: "0"=over, "1"=under	GeneralText	
grade	0x0082	"%d"		
group	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
groupId	0x0004	"%s"		
holeDeviation	0x0307	"%s"		
holeGrade	0x0307	"%d"		
holeLowerDetla	0x0307	"%.16g"		
holeUpperDelta	0x0307	"%.16g"		
hotSpotPosition	0x0118	"%.16g %.16g %.16g"		
identifier	0x0231 0x0232 0x0233 0x0100 0x0235 0x0244	"%s"		
includeI	0x0239	"%d"		
includeJ	0x0239	"%d"		
includeK	0x0239	"%d"		
includeLabel	0x0239	"%d"		
includeLabels	0x0240 0x0241	"%d"		
includeLevel	0x0239	"%d"		
includeX	0x0239	"%d"		
includeY	0x0239	"%d"		
includeZ	0x0239	"%d"		
index	0x0088	"%d"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

indexDirection	0x0230	"%.16g %.16g %.16g"		
individualExtensions	0x0306	"%d"		
individuallyApplied	0x0084	"%d"		
innerDiameter	0x0088	"%.16g"		
invertText	0x0041	"%d"		
iPrefix	0x0239	"%s"		
isReference	0x0082	"%d"		
iSuffix	0x0239	"%s"		
italic	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		
italicAngle	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		
itemName	0x0233	"%s"		
jisArrowSidePartialPenetrationGrooveWeld	0x0008	"%d"		
jisOtherSidePartialPenetrationGrooveWeld	0x0008	"%d"		
jisStaggeredFilletWeld	0x0008	"%d"		
jointId	0x0004	"%s"		
jPrefix	0x0239	"%s"		
jSuffix	0x0239	"%s"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

justification	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=right, "1"=centre, "2"=left		
Keyword[%d]	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
kPrefix	0x0239	"%s"		
kSuffix	0x0239	"%s"		
label	0x0084 0x0088	"%s"		
labelPrefix	0x0239	"%s"		
labelSuffix	0x0239	"%s"		
language	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
lay	0x0041	String representing enumeration such that: "0"=parallel, "1"=multidirectional, "2"=perpendicular, "3"=circular, "4"=particulate, "5"=angularboth, "6"=radial		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

layer	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"		
Leader[%d]	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	NULL		
Leader[%d].arrow Angle	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	
Leader[%d].arrow Colour	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Leader[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].arrow Length	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	
Leader[%d].arrow LineThickness	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Leader[%d]	
Leader[%d].arrow LineType	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=unset, "1"=longdashed, "2"=dotted, "3"=dotteddashed, "4"=phantom, "5"=dashed, "6"=solid, "7"=centreline	Leader[%d]	
Leader[%d].arrow OutSideLength	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].arrow Placement	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=out, "1"=in	Leader[%d]	
Leader[%d].arrow Type	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=closed, "1"=open, "2"=filled, "3"=none, "4"=origin, "5"=cross, "6"=dot, "7"=filleddot, "8"=wedge, "9"=plus, "10"=x, "11"=closedsolid, "12"=closeddouble, "13"=closeddoublesolid, "14"=openddouble, "15"=filleddouble, "16"=integral, "17"=box, "18"=filledbox, "19"=datum, "20"=filleddatum, "21"=solidorigin, "22"=filledorigin, "23"=xorigin, "24"=solidbox	Leader[%d]	
Leader[%d].colour	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Leader[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].dotDiameter	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	
Leader[%d].extensionLineColour	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Leader[%d]	
Leader[%d].extensionLineExtension	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	
Leader[%d].extensionLineGap	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].extensionLineThickness	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Leader[%d]	
Leader[%d].extensionLineType	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=unset, "1"=longdashed, "2"=dotted, "3"=dotteddashed, "4"=phantom, "5"=dashed, "6"=solid, "7"=centreline	Leader[%d]	
Leader[%d].lineTextGap	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	
Leader[%d].lineType	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=unset, "1"=longdashed, "2"=dotted, "3"=dotteddashed, "4"=phantom, "5"=dashed, "6"=solid, "7"=centreline	Leader[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].radius ToCentre	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d"	Leader[%d]	
Leader[%d].Reference	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%d %d %d %s" such that first in is the dst id, second the dst type, third the reason(optional) and the string the path. This property formating is used to semantically tie a singlar semantic aspect with an entity.	Leader[%d]	
Leader[%d].stubDirection	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=left, "1"=right	Leader[%d]	
Leader[%d].stubLength	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].terminator	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g %.16g %.16g"	Leader[%d]	
Leader[%d].thickness	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Leader[%d]	
Leader[%d].tParm	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"	Leader[%d]	
Leader[%d].uvParm.U	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		Expect to also have Leader[%d].uvParam.V.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Leader[%d].uvParam.V	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		Expect to also have Leader[%d].uvParam.U.
leadingZero	0x0081	"%d"		
length	0x0088	"%.16g"		
level	0x0042	"%s"		
levelPrefix	0x0239	"%s"		
levelSuffix	0x0239	"%s"		
limitFitOrder	0x0082 0x0307	String representing enumeration such that: "0"=valuelimitfittolerance, "1"=tolerancevaluelimitfit, "2"=valuetolerancelimitfit		
limitFitParenthesis	0x0082 0x0307	String representing enumeration such that: "0"=none, "1"=valuelimitfit, "2"=limitfit, "3"=tolerance		
linearPlacement	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after		
lineFactor	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		
lineStyle	0x0114	String representing enumeration such that: "0"=unset, "1"=longdashed, "2"=dotted, "3"=dotteddashed, "4"=phantom,		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		"5"=dashed, "6"=solid, "7"=centreline		
locationOnCurve	0x0241	"%.16g"		
lowerDelta	0x0082 0x0041	"%.16g"		
lowerDeltaDenominator	0x0082	"%d"		
machiningAllowance	0x0041	"%s"		
mainEdgeType	0x0118	String representing enumeration such that: "0"=ce, "1"=e, "2"=e, "3"=ek, "4"=co		
mainHoleType	0x0118	String representing enumeration such that: "0"=dl, "1"=h, "2"=hk, "3"=ch, "4"=hk, "5"=ac, "6"=h, "7"=bl, "8"=ch		
mainRectSlotFreeformType	0x0118	String representing enumeration such that: "0"=dl, "1"=h, "2"=hk, "3"=bl, "4"=hk, "5"=ac, "6"=h, "7"=ch		
mainSurfaceType	0x0118	String representing enumeration such that: "0"=sk, "1"=co, "2"=j, "3"=s, "4"=sl, "5"=sk, "6"=cs, "7"=s, "8"=ck		
materialSide	0x0242	"%d"		
maxBonus	0x0081	"%d"		
maxBonusPrecision	0x0081	"%d"		
maxBonusValue	0x0081	"%.16g"		
maxRoughness	0x0041	"%s"		
maxRoughness2	0x0041	"%s"		
measurementDirection	0x0042	"%.16g %.16g %.16g"		
minRoughness	0x0041	"%s"		
minRoughness2	0x0041	"%s"		
modifier	0x0041	"%d"		
movable	0x0088	"%d"		
movableTargetStubDirection	0x0088	String representing enumeration such that: "0"=left, "1"=right		
movableTargetStubLength	0x0088	"%.16g"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

nameModifier	0x0233	"%s"		
Nomenclature	0x0231 0x0232	NULL		
Nomenclature.bold	0x0231 0x0232	"%d"	Nomenclature	
Nomenclature.font	0x0231 0x0232	"%s"	Nomenclature	
Nomenclature.italic	0x0231 0x0232	"%d"	Nomenclature	
Nomenclature.italicAngle	0x0231 0x0232	"%.16g"	Nomenclature	
Nomenclature.Item[%d]	0x0231 0x0232	"%s"	Nomenclature	
Nomenclature.Item[%d].bold	0x0231 0x0232	"%d"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].font	0x0231 0x0232	"%s"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].format	0x0231 0x0232	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	Nomenclature.Item[%d]	
Nomenclature.Item[%d].italic	0x0231 0x0232	"%d"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].italicAngle	0x0231 0x0232	"%.16g"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].justification	0x0231 0x0232	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Nomenclature.Item[%d]	
Nomenclature.Item[%d].language	0x0231 0x0232	"%s"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].lineFactor	0x0231 0x0232	"%.16g"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].outlineType	0x0231 0x0232	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota,	Nomenclature.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		"18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram		
Nomenclature.Item[%d].spaceFactor	0x0231 0x0232	"%.16g"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].string	0x0231 0x0232	"%s"	Nomenclature.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Nomenclature.Item[%d].symbol	0x0231 0x0232	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	Nomenclature.Item[%d]	
Nomenclature.Item[%d].textAspect	0x0231 0x0232	"%.16g"	Nomenclature.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Nomenclature.Item[%d].textColour	0x0231 0x0232	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Nomenclature.Item[%d]	
Nomenclature.Item[%d].textHeight	0x0231 0x0232	"%.16g"	Nomenclature.Item[%d]	
Nomenclature.Item[%d].textThickness	0x0231 0x0232	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Nomenclature.Item[%d]	
Nomenclature.Item[%d].underline	0x0231 0x0232	String representing enumeration such that: "0"=over, "1"=under	Nomenclature.Item[%d]	
Nomenclature.justification	0x0231 0x0232	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Nomenclature	
Nomenclature.language	0x0231 0x0232	"%s"	Nomenclature	
Nomenclature.lineFactor	0x0231 0x0232	"%.16g"	Nomenclature	
Nomenclature.name	0x0231 0x0232	"%s"	Nomenclature	
Nomenclature.outlineType	0x0231 0x0232	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Nomenclature	
Nomenclature.spaceFactor	0x0231 0x0232	"%.16g"	Nomenclature	
Nomenclature.textAspect	0x0231 0x0232	"%.16g"	Nomenclature	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Nomenclature.textColour	0x0231 0x0232	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Nomenclature	
Nomenclature.textHeight	0x0231 0x0232	"%.16g"	Nomenclature	
Nomenclature.textOrientation	0x0231 0x0232	String representing enumeration such that: "0"=horizontal, "1"=vertical	Nomenclature	
Nomenclature.textThickness	0x0231 0x0232	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Nomenclature	
Nomenclature.underline	0x0231 0x0232	String representing enumeration such that: "0"=over, "1"=under	Nomenclature	
normalDirection	0x0004 0x0042 0x0118 0x0238	"%.16g %.16g %.16g"		
Note	0x0118	NULL		
Note.bold	0x0118	"%d"	Note	
Note.font	0x0118	"%s"	Note	
Note.italic	0x0118	"%d"	Note	
Note.italicAngle	0x0118	"%.16g"	Note	
Note.Item[%d]	0x0118	"%s"	Note	
Note.Item[%d].bold	0x0118	"%d"	Note.Item[%d]	
Note.Item[%d].font	0x0118	"%s"	Note.Item[%d]	
Note.Item[%d].format	0x0118	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	Note.Item[%d]	
Note.Item[%d].italic	0x0118	"%d"	Note.Item[%d]	
Note.Item[%d].italicAngle	0x0118	"%.16g"	Note.Item[%d]	
Note.Item[%d].justification	0x0118	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Note.Item[%d]	
Note.Item[%d].language	0x0118	"%s"	Note.Item[%d]	
Note.Item[%d].lineFactor	0x0118	"%.16g"	Note.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Note.Item[%d].outlineType	0x0118	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Note.Item[%d]	
Note.Item[%d].spaceFactor	0x0118	"%.16g"	Note.Item[%d]	
Note.Item[%d].string	0x0118	"%s"	Note.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Note.Item[%d].symbol	0x0118	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	Note.Item[%d]	
Note.Item[%d].textAspect	0x0118	"%.16g"	Note.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Note.Item[%d].textColour	0x0118	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Note.Item[%d]	
Note.Item[%d].textHeight	0x0118	"%.16g"	Note.Item[%d]	
Note.Item[%d].textThickness	0x0118	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Note.Item[%d]	
Note.Item[%d].underline	0x0118	String representing enumeration such that: "0"=over, "1"=under	Note.Item[%d]	
Note.justification	0x0118	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Note	
Note.language	0x0118	"%s"	Note	
Note.lineFactor	0x0118	"%.16g"	Note	
Note.name	0x0118	"%s"	Note	
Note.outlineType	0x0118	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Note	
Note.spaceFactor	0x0118	"%.16g"	Note	
Note.textAspect	0x0118	"%.16g"	Note	
Note.textColour	0x0118	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are	Note	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		encoded in the string		
Note.textHeight	0x0118	"%.16g"	Note	
Note.textOrientation	0x0118	String representing enumeration such that: "0"=horizontal, "1"=vertical	Note	
Note.textThickness	0x0118	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Note	
Note.underline	0x0118	String representing enumeration such that: "0"=over, "1"=under	Note	
notToScale	0x0082	"%d"		
OpenField	0x0231 0x0232	NULL		
OpenField.bold	0x0231 0x0232	"%d"	OpenField	
OpenField.font	0x0231 0x0232	"%s"	OpenField	
OpenField.italic	0x0231 0x0232	"%d"	OpenField	
OpenField.italicAngle	0x0231 0x0232	"%.16g"	OpenField	
OpenField.Item[%d]	0x0231 0x0232	"%s"	OpenField	
OpenField.Item[%d].bold	0x0231 0x0232	"%d"	OpenField.Item[%d]	
OpenField.Item[%d].font	0x0231 0x0232	"%s"	OpenField.Item[%d]	
OpenField.Item[%d].format	0x0231 0x0232	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	OpenField.Item[%d]	
OpenField.Item[%d].italic	0x0231 0x0232	"%d"	OpenField.Item[%d]	
OpenField.Item[%d].italicAngle	0x0231 0x0232	"%.16g"	OpenField.Item[%d]	
OpenField.Item[%d].justification	0x0231 0x0232	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	OpenField.Item[%d]	
OpenField.Item[%d].language	0x0231 0x0232	"%s"	OpenField.Item[%d]	
OpenField.Item[%d].lineFactor	0x0231 0x0232	"%.16g"	OpenField.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

OpenField.Item[%d].outlineType	0x0231 0x0232	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	OpenField.Item[%d]	
OpenField.Item[%d].spaceFactor	0x0231 0x0232	"%.16g"	OpenField.Item[%d]	
OpenField.Item[%d].string	0x0231 0x0232	"%s"	OpenField.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

OpenField.Item[%d].symbol	0x0231 0x0232	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	OpenField.Item[%d]	
OpenField.Item[%d].textAspect	0x0231 0x0232	"%.16g"	OpenField.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

OpenField.Item[%d].textColour	0x0231 0x0232	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	OpenField.Item[%d]	
OpenField.Item[%d].textHeight	0x0231 0x0232	"%.16g"	OpenField.Item[%d]	
OpenField.Item[%d].textThickness	0x0231 0x0232	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	OpenField.Item[%d]	
OpenField.Item[%d].underline	0x0231 0x0232	String representing enumeration such that: "0"=over, "1"=under	OpenField.Item[%d]	
OpenField.justification	0x0231 0x0232	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	OpenField	
OpenField.language	0x0231 0x0232	"%s"	OpenField	
OpenField.lineFactor	0x0231 0x0232	"%.16g"	OpenField	
OpenField.name	0x0231 0x0232	"%s"	OpenField	
OpenField.outlineType	0x0231 0x0232	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	OpenField	
OpenField.spaceFactor	0x0231 0x0232	"%.16g"	OpenField	
OpenField.textAspect	0x0231 0x0232	"%.16g"	OpenField	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

OpenField.textColour	0x0231 0x0232	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	OpenField	
OpenField.textHeight	0x0231 0x0232	"%.16g"	OpenField	
OpenField.textOrientation	0x0231 0x0232	String representing enumeration such that: "0"=horizontal, "1"=vertical	OpenField	
OpenField.textThickness	0x0231 0x0232	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	OpenField	
OpenField.underline	0x0231 0x0232	String representing enumeration such that: "0"=over, "1"=under	OpenField	
optDirection	0x0230	"%.16g %.16g %.16g"		
origin	0x0082	String representing enumeration such that: "0"=first, "1"=second		
otherSideFinishSymbol	0x0008	String representing enumeration such that: "0"=unset, "1"=unspecified, "2"=machining, "3"=grinding, "4"=chipping, "5"=none, "6"=hammering, "7"=peening, "8"=rolling		
otherSideGrooveAngle	0x0008	"%s"		
otherSideGrooveGap	0x0008	"%s"		
otherSideLongitudinalSize	0x0008	"%s"		
otherSideMainSize	0x0008	"%s"		
otherSideStaggeredSize	0x0008	"%s"		
otherSideSupplementalSymbol	0x0008	String representing enumeration such that: "0"=unset, "1"=convex, "2"=toesblended, "3"=concave, "4"=flush		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

otherSideSymbol	0x0008	String representing enumeration such that: "0"=unset, "1"=jisseam, "2"=singlebevelgroove, "3"=singleugroove, "4"=edgeflange, "5"=surface, "6"=steepflankedsinglebevel, "7"=flaresinglevgroove, "8"=surfacejoint, "9"=singlevgroove, "10"=inclinedjoint, "11"=plug, "12"=fillet, "13"=squaregroove, "14"=spot, "15"=singlej, "16"=square, "17"=overlay, "18"=singleu, "19"=singlejgroove, "20"=doubleflange, "21"=singleflange, "22"=spotprojected, "23"=plugandslot, "24"=singlebevelbroadrootface, "25"=edge, "26"=jisstaggeredfillet1, "27"=singlev, "28"=flaresinglebevelgroove, "29"=singlevbroadrootface, "30"=steepflankedsinglev, "31"=removablebackingstrip, "32"=permanentbackingstrip, "33"=jisfillet, "34"=singlebevel, "35"=backing, "36"=foldjoint, "37"=jisstaggeredfillet2, "38"=seam, "39"=bead		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

outlineType	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram		
overrun	0x0306	"%.16g"		
parenthesis	0x0041	String representing enumeration such that: "0"=none, "1"=left, "2"=right, "3"=both		
PartNumber	0x0118	NULL		
PartNumber.bold	0x0118	"%d"	PartNumber	
PartNumber.font	0x0118	"%s"	PartNumber	
PartNumber.italic	0x0118	"%d"	PartNumber	
PartNumber.italicAngle	0x0118	"%.16g"	PartNumber	
PartNumber.Item[%d]	0x0118	"%s"	PartNumber	
PartNumber.Item[%d].bold	0x0118	"%d"	PartNumber.Item[%d]	
PartNumber.Item[%d].font	0x0118	"%s"	PartNumber.Item[%d]	
PartNumber.Item[%d].format	0x0118	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	PartNumber.Item[%d]	
PartNumber.Item[%d].italic	0x0118	"%d"	PartNumber.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

PartNumber.Item[%d].italicAngle	0x0118	"%.16g"	PartNumber.Item[%d]	
PartNumber.Item[%d].justification	0x0118	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	PartNumber.Item[%d]	
PartNumber.Item[%d].language	0x0118	"%s"	PartNumber.Item[%d]	
PartNumber.Item[%d].lineFactor	0x0118	"%.16g"	PartNumber.Item[%d]	
PartNumber.Item[%d].outlineType	0x0118	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	PartNumber.Item[%d]	
PartNumber.Item[%d].spaceFactor	0x0118	"%.16g"	PartNumber.Item[%d]	
PartNumber.Item[%d].string	0x0118	"%s"	PartNumber.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

PartNumber.Item[%d].symbol	0x0118	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	PartNumber.Item[%d]	
PartNumber.Item[%d].textAspect	0x0118	"%.16g"	PartNumber.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

PartNumber.Item[%d].textColour	0x0118	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	PartNumber.Item[%d]	
PartNumber.Item[%d].textHeight	0x0118	"%.16g"	PartNumber.Item[%d]	
PartNumber.Item[%d].textThickness	0x0118	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	PartNumber.Item[%d]	
PartNumber.Item[%d].underline	0x0118	String representing enumeration such that: "0"=over, "1"=under	PartNumber.Item[%d]	
PartNumber.justification	0x0118	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	PartNumber	
PartNumber.language	0x0118	"%s"	PartNumber	
PartNumber.lineFactor	0x0118	"%.16g"	PartNumber	
PartNumber.name	0x0118	"%s"	PartNumber	
PartNumber.outlineType	0x0118	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	PartNumber	
PartNumber.spaceFactor	0x0118	"%.16g"	PartNumber	
PartNumber.textAspect	0x0118	"%.16g"	PartNumber	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

PartNumber.textColour	0x0118	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	PartNumber	
PartNumber.textHeight	0x0118	"%.16g"	PartNumber	
PartNumber.textOrientation	0x0118	String representing enumeration such that: "0"=horizontal, "1"=vertical	PartNumber	
PartNumber.textThickness	0x0118	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	PartNumber	
PartNumber.underline	0x0118	String representing enumeration such that: "0"=over, "1"=under	PartNumber	
pinDirection	0x0118	"%.16g %.16g %.16g"		
pitchDiaDeviation	0x0082 0x0307	"%s"		
pitchDiaGrade	0x0082 0x0307	"%d"		
planar	0x0088	"%d"		
point1	0x0088	"%.16g %.16g %.16g"		
point2	0x0088	"%.16g %.16g %.16g"		
position	0x0230 0x0004 0x0042 0x0238	"%.16g %.16g %.16g"		
Position[%d]	0x0306	"%.16g %.16g %.16g"		
precision	0x0082 0x0230 0x0041 0x0088 0x0004 0x0008 0x0118 0x0238 0x0239 0x0241 0x0307	"%d"		
ProcessText	0x0041 0x0004 0x0008	NULL		
ProcessText.bold	0x0041 0x0004 0x0008	"%d"	ProcessText	
ProcessText.font	0x0041 0x0004 0x0008	"%s"	ProcessText	
ProcessText.italic	0x0041 0x0004 0x0008	"%d"	ProcessText	
ProcessText.italicAngle	0x0041 0x0004 0x0008	"%.16g"	ProcessText	
ProcessText.Item[%d]	0x0041 0x0004 0x0008	"%s"	ProcessText	
ProcessText.Item[%d].bold	0x0041 0x0004 0x0008	"%d"	ProcessText.Item[%d]	
ProcessText.Item[%d].font	0x0041 0x0004 0x0008	"%s"	ProcessText.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ProcessText.Item[%d].format	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	ProcessText.Item[%d]	
ProcessText.Item[%d].italic	0x0041 0x0004 0x0008	"%d"	ProcessText.Item[%d]	
ProcessText.Item[%d].italicAngle	0x0041 0x0004 0x0008	"%.16g"	ProcessText.Item[%d]	
ProcessText.Item[%d].justification	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	ProcessText.Item[%d]	
ProcessText.Item[%d].language	0x0041 0x0004 0x0008	"%s"	ProcessText.Item[%d]	
ProcessText.Item[%d].lineFactor	0x0041 0x0004 0x0008	"%.16g"	ProcessText.Item[%d]	
ProcessText.Item[%d].outlineType	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	ProcessText.Item[%d]	
ProcessText.Item[%d].spaceFactor	0x0041 0x0004 0x0008	"%.16g"	ProcessText.Item[%d]	
ProcessText.Item[%d].string	0x0041 0x0004 0x0008	"%s"	ProcessText.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ProcessText.Item[%d].symbol	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	ProcessText.Item[%d]	
ProcessText.Item[%d].textAspect	0x0041 0x0004 0x0008	"%.16g"	ProcessText.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ProcessText.Item[%d].textColour	0x0041 0x0004 0x0008	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	ProcessText.Item[%d]	
ProcessText.Item[%d].textHeight	0x0041 0x0004 0x0008	"%.16g"	ProcessText.Item[%d]	
ProcessText.Item[%d].textThickness	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	ProcessText.Item[%d]	
ProcessText.Item[%d].underline	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=over, "1"=under	ProcessText.Item[%d]	
ProcessText.justification	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	ProcessText	
ProcessText.language	0x0041 0x0004 0x0008	"%s"	ProcessText	
ProcessText.lineFactor	0x0041 0x0004 0x0008	"%.16g"	ProcessText	
ProcessText.name	0x0041 0x0004 0x0008	"%s"	ProcessText	
ProcessText.outlineType	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	ProcessText	
ProcessText.spaceFactor	0x0041 0x0004 0x0008	"%.16g"	ProcessText	
ProcessText.textAspect	0x0041 0x0004 0x0008	"%.16g"	ProcessText	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ProcessText.textColour	0x0041 0x0004 0x0008	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	ProcessText	
ProcessText.textHeight	0x0041 0x0004 0x0008	"%.16g"	ProcessText	
ProcessText.textOrientation	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=horizontal, "1"=vertical	ProcessText	
ProcessText.textThickness	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	ProcessText	
ProcessText.underline	0x0041 0x0004 0x0008	String representing enumeration such that: "0"=over, "1"=under	ProcessText	
profileType	0x0081	String representing enumeration such that: "0"=bilateral, "1"=unilateralinside, "2"=unilateraloutside, "3"=bilateralunequal		
profileValue	0x0081	"%.16g"		
profileValue2	0x0081	"%.16g"		
projected	0x0082	"%d"		
projectedPrecision	0x0081	"%d"		
punchDirection	0x0118	"%.16g %.16g %.16g"		
radialPlacement	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after		
radius	0x0235	"%.16g"		
referenceDisplay	0x0082	String representing enumeration such that: "0"=parenthesis, "1"=reference, "2"=matched		
referenceSign	0x0008	"%d"		
revision	0x0231 0x0232 0x0233 0x0100 0x0235 0x0244	"%s"		
roughness	0x0041	"%s"		
roughness2	0x0041	"%s"		
roughnessCutoff	0x0041	"%s"		
roughnessSpacing	0x0041	"%s"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

SafetyClassification	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
scale	0x0114	"%.16g"		
shaftDeviation	0x0307	"%s"		
shaftGrade	0x0307	"%d"		
shaftLowerDelta	0x0307	"%.16g"		
shaftUpperDelta	0x0307	"%.16g"		
shapeTextOutsideSymbol	0x0088	"%d"		
shortDash	0x0306	"%.16g"		
singleLine	0x0041 0x0307	"%d"		
size	0x0088	"%s"		
spaceFactor	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		
standard	0x0082 0x0081 0x0084 0x0088 0x0100 0x0235 0x0244 0x0239 0x0240 0x0241	String representing enumeration such that: "0"=asme_y145m_1994, "1"=jis, "2"=iso, "3"=bs, "4"=ansi_y145m_1982, "5"=asme_y1441m_2003, "6"=din, "7"=gm_addendum_1994, "8"=asme_y145_2009		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

standard	0x0041	String representing enumeration such that: "0"=jis, "1"=ansi_y1436_1993, "2"=iso, "3"=asme_y1436m_1996, "4"=din, "5"=gb, "6"=iso_1302_2002, "7"=eskd, "8"=din_en_iso_1302_2002		
standard	0x0008	String representing enumeration such that: "0"=ansiaws_a24-98 "1"=iso_2556 "2"=jis_z_3021 "3"=din "4"=eskd "5"=gb		
Statistical	0x0082	NULL		
statisticalPlacement	0x0082	String representing enumeration such that: "0"=before, "1"=above, "2"=below, "3"=after		
style	0x0082	String representing enumeration such that: "0"=lineardiametral, "1"=radial, "2"=controlledradial, "3"=diametral, "4"=limits, "5"=ordinate, "6"=sphericalradial, "7"=sphericaldiametral		
subType	0x0118	String representing enumeration such that: "0"=mechanicallyfastened, "1"=manufacturingandassembly, "2"=diesmolds, "3"=fixing, "4"=temporarytransferred		
symbol	0x0118	String representing enumeration such that: "0"=rectangle, "1"=edge, "2"=hole, "3"=freeform, "4"=slot, "5"=surface		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

symbolColour	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string		
symbolType	0x0230	"%s"		
Text	0x0081	NULL		
Text.bold	0x0081	"%d"	Text	
Text.font	0x0081	"%s"	Text	
Text.italic	0x0081	"%d"	Text	
Text.italicAngle	0x0081	"%.16g"	Text	
Text.Item[%d]	0x0081	"%s"	Text	
Text.Item[%d].bold	0x0081	"%d"	Text.Item[%d]	
Text.Item[%d].font	0x0081	"%s"	Text.Item[%d]	
Text.Item[%d].format	0x0081	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	Text.Item[%d]	
Text.Item[%d].italic	0x0081	"%d"	Text.Item[%d]	
Text.Item[%d].italicAngle	0x0081	"%.16g"	Text.Item[%d]	
Text.Item[%d].justification	0x0081	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Text.Item[%d]	
Text.Item[%d].language	0x0081	"%s"	Text.Item[%d]	
Text.Item[%d].lineFactor	0x0081	"%.16g"	Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text.Item[%d].outlineType	0x0081	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Text.Item[%d]	
Text.Item[%d].spaceFactor	0x0081	"%.16g"	Text.Item[%d]	
Text.Item[%d].string	0x0081	"%s"	Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text.Item[%d].symbol	0x0081	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	Text.Item[%d]	
Text.Item[%d].textAspect	0x0081	"%.16g"	Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text.Item[%d].textColour	0x0081	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Text.Item[%d]	
Text.Item[%d].textHeight	0x0081	"%.16g"	Text.Item[%d]	
Text.Item[%d].textThickness	0x0081	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Text.Item[%d]	
Text.Item[%d].underline	0x0081	String representing enumeration such that: "0"=over, "1"=under	Text.Item[%d]	
Text.justification	0x0081	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Text	
Text.language	0x0081	"%s"	Text	
Text.lineFactor	0x0081	"%.16g"	Text	
Text.name	0x0081	"%s"	Text	
Text.outlineType	0x0081	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=coredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Text	
Text.spaceFactor	0x0081	"%.16g"	Text	
Text.textAspect	0x0081	"%.16g"	Text	
Text.textColour	0x0081	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are	Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		encoded in the string		
Text.textHeight	0x0081	"%.16g"	Text	
Text.textOrientation	0x0081	String representing enumeration such that: "0"=horizontal, "1"=vertical	Text	
Text.textThickness	0x0081	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Text	
Text.underline	0x0081	String representing enumeration such that: "0"=over, "1"=under	Text	
Text[%d]	0x0100 0x0235 0x0244	NULL		
Text[%d].bold	0x0100 0x0235 0x0244	"%d"	Text[%d]	
Text[%d].font	0x0100 0x0235 0x0244	"%s"	Text[%d]	
Text[%d].italic	0x0100 0x0235 0x0244	"%d"	Text[%d]	
Text[%d].italicAngle	0x0100 0x0235 0x0244	"%.16g"	Text[%d]	
Text[%d].Item[%d]	0x0100 0x0235 0x0244	"%s"	Text[%d]	
Text[%d].Item[%d].bold	0x0100 0x0235 0x0244	"%d"	Text[%d].Item[%d]	
Text[%d].Item[%d].font	0x0100 0x0235 0x0244	"%s"	Text[%d].Item[%d]	
Text[%d].Item[%d].format	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	Text[%d].Item[%d]	
Text[%d].Item[%d].italic	0x0100 0x0235 0x0244	"%d"	Text[%d].Item[%d]	
Text[%d].Item[%d].italicAngle	0x0100 0x0235 0x0244	"%.16g"	Text[%d].Item[%d]	
Text[%d].Item[%d].justification	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Text[%d].Item[%d]	
Text[%d].Item[%d].language	0x0100 0x0235 0x0244	"%s"	Text[%d].Item[%d]	
Text[%d].Item[%d].lineFactor	0x0100 0x0235 0x0244	"%.16g"	Text[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text[%d].Item[%d].outlineType	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Text[%d].Item[%d]	
Text[%d].Item[%d].spaceFactor	0x0100 0x0235 0x0244	"%.16g"	Text[%d].Item[%d]	
Text[%d].Item[%d].string	0x0100 0x0235 0x0244	"%s"	Text[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text[%d].Item[%d].symbol	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	Text[%d].Item[%d]	
Text[%d].Item[%d].textAspect	0x0100 0x0235 0x0244	"%.16g"	Text[%d].Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text[%d].Item[%d].textColour	0x0100 0x0235 0x0244	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Text[%d].Item[%d]	
Text[%d].Item[%d].textHeight	0x0100 0x0235 0x0244	"%.16g"	Text[%d].Item[%d]	
Text[%d].Item[%d].textThickness	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Text[%d].Item[%d]	
Text[%d].Item[%d].underline	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=over, "1"=under	Text[%d].Item[%d]	
Text[%d].justification	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	Text[%d]	
Text[%d].language	0x0100 0x0235 0x0244	"%s"	Text[%d]	
Text[%d].lineFactor	0x0100 0x0235 0x0244	"%.16g"	Text[%d]	
Text[%d].name	0x0100 0x0235 0x0244	"%s"	Text[%d]	
Text[%d].outlineType	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	Text[%d]	
Text[%d].spaceFactor	0x0100 0x0235 0x0244	"%.16g"	Text[%d]	
Text[%d].textAspect	0x0100 0x0235 0x0244	"%.16g"	Text[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Text[%d].textColour	0x0100 0x0235 0x0244	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	Text[%d]	
Text[%d].textHeight	0x0100 0x0235 0x0244	"%.16g"	Text[%d]	
Text[%d].textOrientation	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=horizontal, "1"=vertical	Text[%d]	
Text[%d].textThickness	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	Text[%d]	
Text[%d].underline	0x0100 0x0235 0x0244	String representing enumeration such that: "0"=over, "1"=under	Text[%d]	
textAspect	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		
textBox	0x0239	"%d"		
textColour	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string		
textDirection	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243	"%.16g %.16g %.16g"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

	0x0244 0x0246 0x0306 0x0307			
textHeight	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g"		
textOrigin	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%.16g %.16g %.16g"		
textStubDirection	0x0088	String representing enumeration such that: "0"=left, "1"=right		
textStubLength	0x0088	"%.16g"		
textThickness	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin		
thickness	0x0004	"%.16g"		
thicknesses	0x0004	"%d"		
threadClass	0x0082	"%d"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

threaded	0x0230	"%d"		
ToleranceCompartment[%d]	0x0081	NULL		
ToleranceCompartment[%d].commonZone	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].FreeState	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].maxBonus	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].maxBonusValue	0x0081	"%.16g"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].modifier	0x0081	String representing enumeration such that: "0"=lmc, "1"=mmc, "2"=rfs	ToleranceCompartment[%d]	
ToleranceCompartment[%d].precision	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].PrimaryDatum.Reference[%d]	0x0081	NULL	ToleranceCompartment[%d].PrimaryDatum	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum.
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].datumRef	0x0081	"%d %d %d %s" such that first in is the dst id, second the dst type, third the reason(optional) and the string the path. This property formating is used to semantically tie a singlar semantic aspect with an entity.	ToleranceCompartment[%d].PrimaryDatum.Reference[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum.
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].FreeState	0x0081	"%d"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum.
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].label	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum.
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].modifier	0x0081	String representing enumeration such that: "0"=lmc, "1"=mmc, "2"=rfs	ToleranceCompartment[%d].PrimaryDatum.Reference[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum.
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	0x0081	NULL	ToleranceCompartment[%d].PrimaryDatum.Reference[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum.
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.bold	0x0081	"%d"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.font	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.italic	0x0081	"%d"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.italicAngle	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].bold	0x0081	"%d"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].font	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].format	0x0081	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].italic	0x0081	"%d"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].italicAngle	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].justification	0x0081	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLa	PrimaryDatum can be replaced with SecondaryDatum

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

%d].TextLabel.Item[%d].language			bel.Item[%d]	or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].lineFactor	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].outlineType	0x0081	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].spaceFactor	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].string	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].symbol	0x0081	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textAspect	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textColour	0x0081	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textHeight	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].textThickness	0x0081	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d].underline	0x0081	String representing enumeration such that: "0"=over, "1"=under	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.Item[%d]	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.justification	0x0081	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.language	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.lineFactor	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.name	0x0081	"%s"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.outlineType	0x0081	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.spaceFactor	• 0x0081	• "%.16g"	• ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	• PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textAspect	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textColour	0x0081	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textHeight	0x0081	"%.16g"	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textOrientation	0x0081	String representing enumeration such that: "0"=horizontal, "1"=vertical	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.textThickness	0x0081	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel.underline	0x0081	String representing enumeration such that: "0"=over, "1"=under	ToleranceCompartment[%d].PrimaryDatum.Reference[%d].TextLabel	PrimaryDatum can be replaced with SecondaryDatum or TertiaryDatum. no notes
ToleranceCompartment[%d].projected	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].projectedValue	0x0081	"%s"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].projectionVector	0x0081	"%.16g %.16g %.16g"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].Statistical	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].tangentPlane	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].unitBasis	0x0081	"%d"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].unitBasisValue	0x0081	"%s"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].value	0x0081	"%s"	ToleranceCompartment[%d]	
ToleranceCompartment[%d].zoneShape	0x0081	String representing enumeration such that: "0"=diameter, "1"=sphericaldiameter, "2"=square	ToleranceCompartment[%d]	
toleranceDisplay	0x0082 0x0041 0x0307	String representing enumeration such that: "0"=minuslimit1, "1"=bilateral, "2"=pluslimit1, "3"=pluslimit2, "4"=equalbilateral, "5"=minuslimit2		
toleranceLeadingZero	0x0082 0x0041 0x0307	"%d"		
tolerancePrecision	0x0082 0x0307	"%d"		
toleranceTrailingZero	0x0082 0x0041 0x0307	"%d"		
trailingZero	0x0081	"%d"		
twoArrows	0x0242	"%d"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

type	0x0082	String representing enumeration such that: "0"=curvelength, "1"=linear, "2"=angular, "3"=radial		
type	0x0230	"%s"		
type	0x0041	String representing enumeration such that: "0"=basic, "1"=mrr, "2"=mrp		
type	0x0088	String representing enumeration such that: "0"=cylindrical, "1"=line, "2"=arbitrary, "3"=circular, "4"=point, "5"=annular, "6"=area, "7"=rectangular, "8"=spherical		
type	0x0004	String representing enumeration such that: "0"=unset, "1"=resistance, "2"=projection		
type	0x0241	String representing enumeration such that: "0"=bundlenote, "1"=dressingnote		
type	0x0242	String representing enumeration such that: "0"=plane, "1"=open, "2"=closed		
type	0x0244	String representing enumeration such that: "0"=exportcontrol, "1"=productproprietaryinfo, "2"=governmentsecurityinfo		
underline	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=over, "1"=under		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

units	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=metres, "1"=millimetres, "2"=inches		
unitText	0x0082	"%d"		
upperDelta	0x0082 0x0041	"%.16g"		
upperDeltaDenominator	0x0082	"%d"		
uriRefs[%d]	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
usage	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
UserDefinedSymbolText[%d]	0x0114	NULL		
UserDefinedSymbolText[%d].Text	0x0114	NULL	UserDefinedSymbolText[%d]	
UserDefinedSymbolText[%d].Text.bold	0x0114	"%d"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.font	0x0114	"%s"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.italic	0x0114	"%d"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.ita	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

licAngle				
UserDefinedSymbolText[%d].Text.Item[%d]	0x0114	"%s"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.Item[%d].bold	0x0114	"%d"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].font	0x0114	"%s"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].format	0x0114	String representing enumeration such that: "0"=int, "1"=ints, "2"=real, "3"=reals, "4"=boolean, "5"=booleans, "6"=string, "7"=reference, "8"=enum, "9"=list, "10"=datetime	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].italic	0x0114	"%d"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].italicAngle	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].justification	0x0114	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].language	0x0114	"%s"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].lineFactor	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].outlineType	0x0114	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpar	UserDefinedSymbolText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

		t, "19"=symmetricalset , "20"=scoredrectangl e, "21"=parallelogram		
UserDefinedSymbolText[%d].Text.Item[%d].spaceFactor	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].string	0x0114	"%s"	UserDefinedSymbolText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

UserDefinedSymbolText[%d].Text.Item[%d].symbol	0x0114	String representing enumeration such that: "0"=centreline, "1"=partingline, "2"=depth, "3"=countersink, "4"=conicaltaper, "5"=slope, "6"=counterbore, "7"=square, "8"=phi, "9"=plusminus, "10"=degrees, "11"=between, "12"=arclength, "13"=leftparenthesis, "14"=rightparenthesis, "15"=projectedtolerance, "16"=mmc, "17"=lmc, "18"=freestate, "19"=ohm, "20"=circularrunout, "21"=totalrunout, "22"=profileofasurface, "23"=profileofaline, "24"=flatness, "25"=straightness, "26"=symmetry, "27"=perpendicularity, "28"=parallelism, "29"=cylindricity, "30"=concentricity, "31"=circularity, "32"=angularity, "33"=micro, "34"=position, "35"=envelope, "36"=rfs, "37"=tangentplane, "38"=lessthanorequal, "39"=greaterthanorequal, "40"=threadprefix, "41"=triangle, "42"=statistical, "43"=radius, "44"=circledu, "45"=fitfunction, "46"=safetycompliance, "47"=quantity, "48"=independency, "49"=continuousfeature, "50"=spotface	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].textAspect	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text.Item[%d]	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

UserDefinedSymbolText[%d].Text.Item[%d].textColour	0x0114	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].textHeight	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].textThickness	0x0114	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.Item[%d].underline	0x0114	String representing enumeration such that: "0"=over, "1"=under	UserDefinedSymbolText[%d].Text.Item[%d]	
UserDefinedSymbolText[%d].Text.justification	0x0114	String representing enumeration such that: "0"=right, "1"=centre, "2"=left	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.language	0x0114	"%s"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.lineFactor	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.name	0x0114	"%s"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.outlineType	0x0114	String representing enumeration such that: "0"=box, "1"=triangle, "2"=circle, "3"=ellipse, "4"=underline, "5"=square, "6"=scoredcircle, "7"=diamond, "8"=flagright, "9"=flagleft, "10"=flagboth, "11"=oblong, "12"=oblongright, "13"=oblongleft, "14"=sticking, "15"=set, "16"=fixedsupport, "17"=nota, "18"=symmetricalpart, "19"=symmetricalset, "20"=scoredrectangle, "21"=parallelogram	UserDefinedSymbolText[%d].Text	

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

UserDefinedSymbolText[%d].Text.spaceFactor	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.textAspect	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.textColour	0x0114	"0x00%02x%02x%02x" such that the values of blue, green and red for the colour are encoded in the string	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.textHeight	0x0114	"%.16g"	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.textOrientation	0x0114	String representing enumeration such that: "0"=horizontal, "1"=vertical	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.textThickness	0x0114	String representing enumeration such that: "0"=normal, "1"=thick, "2"=thin	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].Text.underline	0x0114	String representing enumeration such that: "0"=over, "1"=under	UserDefinedSymbolText[%d].Text	
UserDefinedSymbolText[%d].textDirection	0x0114	"%.16g %.16g %.16g"	UserDefinedSymbolText[%d]	
UserDefinedSymbolText[%d].textOrigin	0x0114	"%.16g %.16g %.16g"	UserDefinedSymbolText[%d]	
valid	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	String representing enumeration such that: "0"=true, "1"=false, "2"=unknown		
value	0x0082 0x0307	"%.16g"		
valueDenominator	0x0082	"%d"		
valueLeadingZero	0x0307	"%d"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

ValueToCustomer	0x0082 0x0230 0x0041 0x0081 0x0084 0x0088 0x0004 0x0008 0x0231 0x0232 0x0233 0x0245 0x0100 0x0042 0x0118 0x0114 0x0238 0x0239 0x0240 0x0241 0x0242 0x0243 0x0244 0x0246 0x0306 0x0307	"%s"		
valueTrailingZero	0x0307	"%d"		
wavinessHeight	0x0041	"%s"		
wavinessSpacing	0x0041	"%s"		
width	0x0088	"%.16g"		
xPrefix	0x0239	"%s"		
xSuffix	0x0239	"%s"		
yPrefix	0x0239	"%s"		
ySuffix	0x0239	"%s"		
zeroToleranceDisplay	0x0082 0x0041 0x0307	String representing enumeration such that: "0"=displayzero, "1"=blank, "2"=suppresstrailing zero, "3"=omit		
zPrefix	0x0239	"%s"		
zSuffix	0x0239	"%s"		

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

14.8.3 Tessellation Properties

When dealing with facetted graphical representations (i.e. LODs) of precise models (e.g. JT B-Rep), depending on the desired use it is often useful/necessary to know what tessellation tolerances were used to generate the facetted representation. To that end, two properties are typically stored on Part Node Elements (if part also has precise model) to indicate the tessellation tolerances used to generate each LOD. These two tessellation properties are as follows

© ISO 2012 – All rights reserved

Table 118 —Tessellatin Proptery values

JT Property Key	Meaning	JT File Data Type	Encoded Data Type	Valid Values
Chordal::	Chordal deviation tessellation tolerance in MCS units for each LOD. Measure of maximum allowable distance a linear approximation for a curve/surface may deviate from the true curve/surface. Encoded value string would look as follows for the case of two LODs: “0.045603 0.069245”	MbString	space separated F32 values	Numeric
Angular::	Angular tessellation tolerance for each LOD in degrees. Two consecutive segments in a linear approximation of a curve/surface form an angle; this value specifies the maximum angle allowed. Encoded value string would look as follows for the case of two LODs: “30.000000 40.000000”	MbString	space separated F32 values	Numeric

14.8.4 Miscellaneous Properties

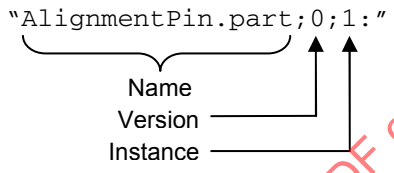
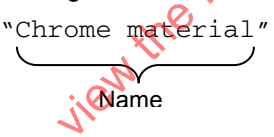
The below table documents some miscellaneous properties often placed on various nodes in the LSG to communicate specific information about the node or its contents.

Table 119 — Miscellaneous Proptery values

JT Property Key	Meaning	JT File Data Type	Encoded Data Type	Valid Values
PMI_TYPE_TABLE	May be attached to Part Node Element to indicate the list of PMI type values and associated names for all PMI types (basically equivalent to the Entity Type field documented in Generic PMI Entities). The string is a “.” and “,” delimited string of the following form: “10.Groove Weld,11.Fillet Weld,12.Plug/Slot Weld,14.Edge Weld”	MbString	<string>	
JT_PROP_SHAPE_DATA_TYPE	May be attached to Shape Node Elements to indicate what geometry type the shape data represents.	MbString	<string>	“Surface” “Wire”
~~JT_PROP_TRISTRIP_DATA_LAYOUT~~	This property is deprecated, and is no longer used.			
JT_PROP_ORIGINATING_BREPTYPE	May be attached to Part Node Element to indicate the type of B-Rep associated with the Part.	MbString	<string>	“None” “JtBrep” “XTBrep”

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

JT Property Key	Meaning	JT File Data Type	Encoded Data Type	Valid Values
JT_PROP_NAME	May be attached to any form of node or attribute with which one wants to associate a textual name (e.g. Part/Assembly/Instance name, Material name, Light Set name, etc.). For Part/Assembly/Instance names this string has the following encoded form where ";" is a delimiter and ":' is a terminator:  For attribute names this string has the following encoded form:  	MbString	<string>	

14.9 LSG Attribute Accumulation Semantics

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

Although each attribute type defines its own application and accumulation LSG semantics (the details of which can be found in each attribute type sub-section under 5.1.2 Attribute Elements), there are some general rules which apply:

Attributes at lower level in the LSG take precedence and replace or accumulate with attributes set at higher levels. When multiple Attributes of the same type are present on a Node, they accumulate in the order they are specified (i.e. from the front of the Attribute list toward the back).

Nodes with no associated attributes inherit those of their parents.

Attributes are inherited only from a node's parents. Thus a given node's attributes do not affect those on the node's siblings.

The root of a partition inherits the attributes in effect at the referring partition node.

Attributes can be marked "final", which terminates accumulation of that attribute type at that marked attribute and propagates the accumulated value at that point to all descendants of the associated node. Descendants can override a "final" attribute using the "force" flag. Note that "force" does not turn OFF "final" – it is simply a one-shot override of "final" for the specific attribute marked as "forcing." Multiple attributes of the same type may be marked as "forcing" and in this case, the last one wins. Both of these flags are OFF by default. An

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

analogy for this “force” and “final” interaction is that “final” is a back-door in the attribute accumulation semantics, and that “force” is the doggy-door in the back-door!

14.10 LSG Part Structure

The JT Format Reference does not mandate that a particular node hierarchy be used for modeling physical Parts within a LSG structure. In fact there are many node hierarchies for representing Parts in LSG that will function correctly in most JT enabled applications. Still, there is a convention that most JT translators follow (and some JT enabled applications may assume exists) for modeling Parts within a LSG. The convention is to model each Part within a LSG structure with the following node hierarchy:

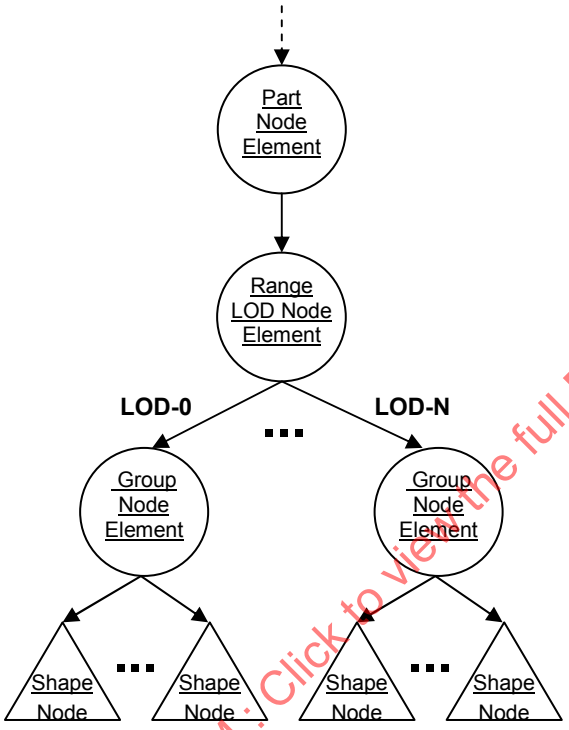


Figure 251 — JT Format Convention for Modeling each Part in LSG

14.11 Range LOD Node Alternative Rep Selection

Best practices suggest that LSG traversers apply the following strategy, at Range LOD Nodes (see 5.1.1.8 Range LOD Node Element), when making alternative representation selection decisions based on Range Limits: The first alternate representation is valid when the world coordinate 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.

14.12 Brep Face Group Associations

The original purpose of the face group concept was to provide associativity between Brep faces and geometry. Exactly how a Brep face associates to a face group number is the topic of this section. An implicit scheme has been chosen for face group associativity, rather than storing some kind of explicit data on either the Vertex Shape LOD Data or the Brep. The primary motivation for this implicit scheme is to keep the JT files simple and small; additional association information would not only be redundant, but also wasteful. Tessellators shall exercise this policy when producing Vertex Shape LOD Data from Breps, grouping the triangles into face groups according to its rules. Tristrips may not cross face groups. Applications shall be able to count on this policy so that, for example, they can map a picking action back to its corresponding Brep face reliably.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

JTBrep/ULP: In the case of JtBrep and ULP reps, the mapping is simple. These Reps have a consistent, sequential, index origin-0 numbering scheme for their regions, shells, and faces. So the Brep faces are simply assigned sequentially to face group by increasing region and shell. For example, suppose we have a JTBrep with 2 regions, each with 2 shells, each with 2 faces. The Face Group ⇔ Region/Shell/Face mapping will be as follows:

```
FG0 ⇔ R0 S0 F0
FG1 ⇔ R0 S0 F1
FG2 ⇔ R0 S1 F0
FG3 ⇔ R0 S1 F1
FG4 ⇔ R1 S0 F0
FG5 ⇔ R1 S0 F1
FG6 ⇔ R1 S1 F0
FG7 ⇔ R1 S1 F1
```

JtXTBrep: In the case of JtXTBrep, the mapping is based on an identifier of each XT face that is persisted on disk. The identifier is unique within each XT body, but it is not an index. XTBrep maintains a zero-based contiguous index of XT face based on increasing identifier value within the same XT body. In the case when multiple bodies are present in JtXTBrep, face index is assigned sequentially by increasing XT body index. For example, suppose we have a JtXTBrep with 2 bodies, each with 2 faces, then the Face Group to Body/Face mapping will be as follows:

```
FG0 ⇔ B0 F0
FG1 ⇔ B0 F1
FG2 ⇔ B1 F0
FG3 ⇔ B1 F1
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Annex A
(informative)

Object Type Identifiers

All objects stored in a JT file are classified by type and thus include an object type identifier as part of their persisted data. The data format for these Object Type identifiers is a GUID. These Object Type identifiers are consistent for all objects, of a particular type, in all Version 8.1 JT files.

Table A.1 — Object Type Identifiers lists the assigned identifier for each Object Type that can exist in a Version 9.5 JT file.

Table A.1 — Object Type Identifiers

GUID	Object Type
0xffffffff, 0xffff, 0xffff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff	Identifier to signal End-Of-Elements
Types Stored Within LSG Segment (Segment Type = 1)	
0x10dd1035, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Base Node Element
0x10dd101b, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Group Node Element
0x10dd102a, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Instance Node Element
0x10dd102c, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	LOD Node Element
0xce357245, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1	Meta Data Node Element
0xd239e7b6, 0xdd77, 0x4289, 0xa0, 0x7d, 0xb0, 0xee, 0x79, 0xf7, 0x94, 0x94	NULL Shape Node Element
0xce357244, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1	Part Node Element
0x10dd103e, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Partition Node Element
0x10dd104c, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Range LOD Node Element
0x10dd10f3, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Switch Node Element
0x10dd1059, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Base Shape Node Element
0x98134716, 0x0010, 0x0818, 0x19, 0x98, 0x08, 0x00, 0x09, 0x83, 0x5d, 0x5a	Point Set Shape Node Element
0x10dd1048, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Polygon Set Shape Node Element
0x10dd1046, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Polyline Set Shape Node Element

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

0xe40373c1, 0x1ad9, 0x11d3, 0x9d, 0xaf, 0x0, 0xa0, 0xc9, 0xc7, 0xdd, 0xc2	Primitive Set Shape Node Element
0x10dd1077, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Tri-Strip Set Shape Node Element
0x10dd107f, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Vertex Shape Node Element
0x10dd1001, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Base Attribute Data
0x10dd1014, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Draw Style Attribute Element
0x10dd1083, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Geometric Transform Attribute Element
0x10dd1028, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Infinite Light Attribute Element
0x10dd1096, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Light Set Attribute Element
0x10dd10c4, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Linestyle Attribute Element
0x10dd1030, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Material Attribute Element
0x10dd1045, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Point Light Attribute Element
0x8d57c010, 0xe5cb, 0x11d4, 0x84, 0xe, 0x00, 0xa0, 0xd2, 0x18, 0x2f, 0x9d	Pointstyle Attribute Element
0xaa1b831d, 0x6e47, 0x4fee, 0xa8, 0x65, 0xcd, 0x7e, 0x1f, 0x2f, 0x39, 0xdb	Shader Effects Attribute Element
0x10dd1073, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Texture Image Attribute Element
0xaa1b831d, 0x6e47, 0x4fee, 0xa8, 0x65, 0xcd, 0x7e, 0x1f, 0x2f, 0x39, 0xdc	Texture Coordinate Generator Attribute Element
0xa3cfb921, 0xbdeb, 0x48d7, 0xb3, 0x96, 0x8b, 0x8d, 0xe, 0xf4, 0x85, 0xa0	Mapping Plane Element
0x3e70739d, 0x8cb0, 0x41ef, 0x84, 0x5c, 0xa1, 0x98, 0xd4, 0x0, 0x3b, 0x3f	Mapping Cylinder Element
0x72475fd1, 0x2823, 0x4219, 0xa0, 0x6c, 0xd9, 0xe6, 0xe3, 0x9a, 0x45, 0xc1	Mapping Sphere Element
0x92f5b094, 0x6499, 0x4d2d, 0x92, 0xaa, 0x60, 0xd0, 0x5a, 0x44, 0x32, 0xcf	Mapping TriPlanar Element
0x10dd104b, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Base Property Atom Element
0xce357246, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1	Date Property Atom Element
0x10dd102b, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Integer Property Atom Element
0x10dd1019, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Floating Point Property Atom Element
0xe0b05be5, 0xfbbd, 0x11d1, 0xa3, 0xa7, 0x00, 0xaa, 0x00, 0xd1, 0x09, 0x54	Late Loaded Property Atom Element

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

0x10dd1004, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	JT Object Reference Property Atom Element
0x10dd106e, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	String Property Atom Element
Types Stored Within JT B-Rep Segment (Segment Type = 2)	
0x873a70c0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	JT B-Rep Element
Types Stored Within Meta Data Segment (Segment Type = 4)	
0xce357249, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1	PMI Manager Meta Data Element
0xce357247, 0x38fb, 0x11d1, 0xa5, 0x6, 0x0, 0x60, 0x97, 0xbd, 0xc6, 0xe1	Property Proxy Meta Data Element
Types Stored Within Shape LOD Segment (Segment Type = 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)	
0x3e637aed, 0x2a89, 0x41f8, 0xa9, 0xfd, 0x55, 0x37, 0x37, 0x3, 0x96, 0x82	Null Shape LOD Element
0x98134716, 0x0011, 0x0818, 0x19, 0x98, 0x08, 0x00, 0x09, 0x83, 0x5d, 0x5a	Point Set Shape LOD Element
0x10dd10a1, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Polyline Set Shape LOD Element
0xe40373c2, 0x1ad9, 0x11d3, 0x9d, 0xaf, 0x0, 0xa0, 0xc9, 0xc7, 0xdd, 0xc2	Primitive Set Shape Element
0x10dd10ab, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Tri-Strip Set Shape LOD Element
0x10dd10b0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Vertex Shape LOD Element
Types Stored Within XT B-Rep Segment (Segment Type = 17)	
0x873a70e0, 0x2ac9, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	XT B-Rep Element
Types Stored Within Wireframe Segment (Segment Type = 18)	
0x873a70d0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97	Wireframe Rep Element
Types Stored Within JT ULP Segment (Segment Type = 20)	
0xf338a4af, 0xd7d2, 0x41c5, 0xbc, 0xf2, 0xc5, 0x5a, 0x88, 0xb2, 0x1e, 0x73	JT ULP Element
Types Stored Within JT LWPA Segment (Segment Type = 24)	
0xd67f8ea8, 0xf524, 0x4879, 0x92, 0x8c, 0x4c, 0x3a, 0x56, 0x1f, 0xb9, 0x3a	JT LWPA Element

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Annex B
(informative)
Coding Algorithms – An Implementation

This Appendix provides a sample C++ implementation for the encoding and decoding portion of the various compression CODECs (as detailed in 13.2 Encoding Algorithms) used in the JT format. This sample code is not intended to be fully functional encoder/decoder class implementations, but is instead intended to demonstrate the fundamentals of implementing the encoding/decoding portion of the CODEC algorithms used in the JT format.

B.1 Common classes

The following sub-sections define some general classes used by all the coding algorithms.

B.1.1 CntxEntry class

```
//
// Type used to build probability context tables.
// Used by ProbabilityContext class.
//
class CntxEntry
{
public:

    Int32 iSym;          // Symbol
    Int32 cCount;        // Number of occurrences
    Int32 cCumCount;     // Cumulative number of occurrences
    Int32 iNextCntx = 0; // Next context if this symbol seen
};
```

B.1.2 CntxEntryBase2 class

```
//
// Type used to build probability context 2 tables.
// Used by ProbabilityContext 2 class.
//
class CntxEntryBase2
{
  public:

    // ---------- Housekeeping ----------
    typedef enum { CEBUninitialized = -1,
                   CEBEscape        = -2,
                   CEBDeleted       = -3
    } CEBSpecialSym;
    CntxEntryBase2() : _iSym(CEBUninitialized), _cCount(-1), _cCumCount(-1) {};
    CntxEntryBase2(Int32 iSym, Int32 cCount)
        : _iSym(iSym), _cCount(cCount), _cCumCount(-1) {};
    CntxEntryBase2(const CntxEntryBase2 &rhs) { *this = rhs; }
    ~CntxEntryBase2() {};
    CntxEntryBase2 &operator=(const CntxEntryBase2 &rhs)
      { _iSym      = rhs._iSym;
        _cCount    = rhs._cCount;
        _cCumCount = rhs._cCumCount;
        return *this;
      }

    // ---------- Operations Interface ----------
    Bool isEscape() const
      { return (_iSym == CEBEscape); }
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    Bool makeEscape(Bool bEsc = True)
      { _iSym = (bEsc ? CEBEscape : CEBUninitialized); return True; }
    Int32 operator==(const CntxEntryBase2 &rhs) const
      { return (_iSym == rhs._iSym); };

  public:

    // ---------- Member Variables ----------
    Int32         _iSym;      // Symbol
    Int32         _cCount;    // Number of occurrences
    Int32         _cCumCount; // Cumulative number of occurrences
};

class CntxEntry2 : public CntxEntryBase2
{
  public:

    // ---------- Housekeeping ----------
    CntxEntry2() : CntxEntryBase2(), _val(0) {};

    CntxEntry2( Int32 iSym, Int32 cCount, const Int32 &val ):
        CntxEntryBase2(iSym, cCount), _val(val) {};

    CntxEntry2( const CntxEntry2 &rhs ) { *this = rhs; }
    ~CntxEntry2() {};
    CntxEntry2 &operator=(const CntxEntry2 &rhs)
        { _val = rhs._val;
         CntxEntryBase2::operator= (rhs);
         return *this;
        }
    Int32 operator==(const CntxEntry2 &rhs) const
        { return (_iSym == rhs._iSym); };

  public:

    // ---------- Member Variables ----------
    Int32           _val;     // Associated value
};
```

B.1.3 ProbabilityContext class

```
//
// Type used to build probability context tables.
// Used by CodecDriver class.
//
class ProbabilityContext
{
public:

    // Returns total cumulative count for all context entries
    Int32 totalCount();

    // Returns number of context entries
    Int32 numEntries();

    // Returns context entry of index iEntry
    Bool getEntry(Int32 iEntry, CntxEntry& rpEntry);

    // Looks up the next context field given by the context entry
    // with input symbol 'iSymbol'
    Bool lookupNextContext(Int32 iSymbol, Int32& iNextContext);

    // Looks up the index of the context entry for the given
    // input symbol 'iSymbol'
    Bool lookupSymbol(Int32 iSymbol, Int32& iOutEntry);

    // Looks up the index of the context entry that falls just above
    // the accumulated count.
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    Bool lookupEntryByCumCount(Int32 iCount, Int32& iOutEntry);
};
```

B.1.4 ProbContex2 class

```
//
// Type used to build probability context 2 tables.
// Used by CodecDriver2 class.
//
class ProbContex2
{
  public:
    // ---------- Housekeeping ----------
    ProbContext2();
    ProbContext2(const ProbContext2 &rhs);
    ~ProbContext2();
    ProbContext2 &operator=(const ProbContext2 &rhs);
    Bool operator==(const ProbContext2 &rhs) const;
    enum { cMaxCntxCount = 8192 };

    // ---------- Accessor Interface ----------
    Int32 totalCount() const
        { return _cTotalCount; }
    Int32 numEntries() const
        { return _vEntries.length(); }
    Bool getEntry(Int32 iEntry, const CntxEntryBase2 *&rpEntry) const
        { const CntxEntry2 *aEntries = _vEntries.ptr();
          rpEntry = &aEntries[iEntry]; return True; }
    Bool getEntryV(Int32 iEntry, const CntxEntry2 *&rpEntry) const
        { const CntxEntry2 *aEntries =  _vEntries.ptr();
          rpEntry = &aEntries[iEntry]; return True; }
    Bool getEntryV(Int32 iEntry, CntxEntry2 *&rpEntry)
        { CntxEntry2 *aEntries = _vEntries.ptr();
          rpEntry = &aEntries[iEntry]; return True; }

    // ---------- Lookup Interface ----------
    Bool lookupValue(const Int32 &rValue, const CntxEntry2 *&opCntxEntry) const;
    Bool lookupEntryByCumCount(Int32 iCount, const CntxEntry2 *&opCntxEntry) const;

    // ---------- Reorganizing Interface ----------
    Bool accumulateCounts();
    Bool sortByValue();

  protected:
    Vec< CntxEntry2 >  _vEntries;
    Int32              _cTotalCount;
    Int32              _iEscPosCache;

    static int _compareCounts(const void *pVal1,
                              const void *pVal2,
                              const void *uData);
};

Bool
ProbContext2::lookupValue( const Int32      &rValue,
                           const CntxEntry2 *&opCntxEntry ) const
{
    // If we do not find the value, then return NULL for the entry
    opCntxEntry = NULL;

    // If the escape position is not cached, sort the context by value
    // and then set it.  Then, we can binary search for values.  We do
    // this because translateValuesToSymbols() will call this method in
    // a tight loop.  Anything is better than linear search!
    ProbContext2 *pThis = (ProbContext2*) this;
    CntxEntry2 *pEntries = pThis->_vEntries.ptr();
   Int32 nEntries = _vEntries.length();
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    if (_iEscPosCache == -1) {
        // Search for the escape symbol
        Bool bFoundEsc = False;
        for (Int32 i = 0 ; i < nEntries ; i++) {
            if (pEntries[i]._iSym == CntxEntryBase2::CEBEscape) {
                // Move the escape symbol to context slot 0
                ::swap(pEntries[0], pEntries[i]);
                bFoundEsc = True;
                break;
            }
        }
        // Sort by value
        if (bFoundEsc) {
            // Sort by value _except_ leave the escape symbol in slot 0
            ::sort(&pEntries[1], nEntries-1, FtorCntxValue());
            pThis->accumulateCounts();
            pThis->_iEscPosCache = 0;
        }
        else {
            pThis->sortByValue();
            pThis->_iEscPosCache = -2;
        }
    }

    // Binary search for rValue!
    Int32 l = (_iEscPosCache == 0),
          h = nEntries - 1,
          m, d;
    while (l <= h) {
        m = (l + h) >> 1;
        d = pEntries[m]._val - rValue;
        if (d == 0) {
            opCntxEntry = &pEntries[m];
            return True;
        }
        else if (d < 0)
            l = m + 1;
        else
            h = m - 1;
    }

    // If we don't find the value, then we return the position of
    // the escape symbol.
    if (_iEscPosCache >= 0)
        opCntxEntry = &pEntries[_iEscPosCache];

    return True;
}

Bool ProbContext2::lookupEntryByCumCount(Int32 iCount, const CntxEntry2 *&opCntxEntry )
const
{
    const CntxEntry2 *aEntries = _vEntries.ptr();
    const Int32       nEntries = _vEntries.length();

    const Int32       seqSearchLen = 4;
    Int32 ii=0;
    opCntxEntry = NULL;

    // For short lists, do sequential search
    if ( nEntries <= (seqSearchLen*2) ) {
        ii = 0;
        while ( (iCount >= (aEntries[ii]._cCumCount + aEntries[ii]._cCount)) &&
(ii<nEntries) ) {
            ii++;
        }

        if ( ii >= nEntries ) {
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
            Assert( 0 && "Bad probability table" );
        }
        opCntxEntry = &aEntries[ii];
    }

    // For long lists, do a short sequential searches through most likely
    // elements, then do a binary search through the rest.
    else {
        for (ii=0; ii<seqSearchLen; ii++) {
            if (iCount < (aEntries[ii]._cCumCount + aEntries[ii]._cCount)) {
                opCntxEntry = &aEntries[ii];
                return True;
            }
        }

        Int32 low=ii, high=nEntries-1, mid;
        while(1) {
            if ( high < low ) {
                break;
            }
            mid = low + ((high-low)>>1);

            if ( iCount < aEntries[mid]._cCumCount ) {
                high = mid-1;
                continue;
            }
            if ( iCount >= (aEntries[mid]._cCumCount + aEntries[mid]._cCount) ) {
                low = mid+1;
                continue;
            }

            opCntxEntry = &aEntries[mid];
            return True;
        }
        Assert( 0 && "Bad probability table" );
    }

    return True;
}

Bool ProbContext2::accumulateCounts()
{
    // Check for zero length context
    CntxEntry2 *aEntries = _vEntries.ptr();
    Int32       nEntries = _vEntries.length();
    if ( nEntries == 0 ) {
        _cTotalCount = 0;
        return True;
    }

    // Accumulate counts in _cCumCount for entries 1 and higher
    aEntries[0]._cCumCount = 0;
   Int32 ii;
    for ( ii=1 ; ii<nEntries ; ii++ ) {
        aEntries[ii]._cCumCount = aEntries[ii-1]._cCount + aEntries[ii-1]._cCumCount;
    }

    // Set the total count for the context
    _cTotalCount = aEntries[ii-1]._cCount + aEntries[ii-1]._cCumCount;

    return True;
}

struct FtorCntxValue
{
   Bool operator () (const CntxEntry2& l, const CntxEntry2& r) const
   { return  (l._val < r._val); }
};
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
Bool ProbContext2::sortByValue()
{
    // Sort the entries in order of values from smallest to largest
     ProbContext2 *pThis = (ProbContext2*) this;
     sort( _vEntries.ptr(), (size_t)(_vEntries.length()), FtorCntxValue() );

    pThis->_iEscPosCache = -1;
     pThis->accumulateCounts();

     return True;
}
```

B.1.5 CodecDriver class

```
//
// A class that deals with the conversions from SYMBOL to VALUE and
// provides end-consumer APIs for using the codecs.
//
class CodecDriver
{
public:
    // ---------- Codec Decoding Interface ----------
    // Returns the number of symbols to be read
    Int32 numSymbolsToRead();

    // Returns index of the first context entry and total number of bits
    Bool getDecodeData(Int32& iFirstContext, Int32& nTotalBits);

    // Returns the next 32 bits of CodeText
    Bool getNextCodeText(UInt32& uCodeText, Int32& nBits);

    // Adds the decoded symbol back to the driver
    Bool addOutputSymbol(Int32 iSymbol, Int32& iNextContext) ;

    // ---------- Symbol Probability Context Interface ----------
    Bool clearAllContexts();

    // Retrieves a new probability context
    Bool getNewContext(ProbabilityContext& rpCntx);

    // Returns the total number of contexts
    Int32 numContexts();

    // Returns the probability context for a given index
    Bool getContext(Int32 iSymContext, ProbabilityContext& rpCntx);

    // ---------- Predictor Type Residual Unpacking ----------

    typedef enum
    {
        PredLag1       = 0,
        PredLag2       = 1,
        PredStride1    = 2,
        PredStride2    = 3,
        PredStripIndex = 4,
        PredRamp       = 5,
        PredXor1       = 6,
        PredXor2       = 7,
        PredNULL       = 8
    } PredictorType;

    // Returns the original values from the predicted residual values.
    static Bool unpackResiduals(Vector<Int32>& rvResidual,
                                Vector<Int32>& rvVals,
                                PredictorType  ePredType);
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    static Bool unpackResiduals(Vector<Float64>& rvResidual,
                                Vector<Float64>& rvVals,
                                PredictorType  ePredType);

    // Predict values
    static Int32 predictValue(Vector<Int32>& vVal,
                              Int32 iIndex,
                              PredictorType ePredType);

    static Float64 predictValue(Vector<Float64>& vVal,
                                Int32 iIndex,
                                PredictorType ePredType);
}

Bool CodecDriver::unpackResiduals(Vector<Int32>& rvResidual,
                                  Vector<Int32>& rvVals,
                                  PredictorType  ePredType)
{
    Int32 iPredicted;

    Int32 len = rvResidual.length();
    rvVals.setLength(len);
    Int32* aVals = (Int32 *) rvVals;
    Int32* aResidual = (Int32 *) rvResidual;

    for( Int32 i = 0; i < len; i++ )
    {
        if( i < 4 )
        {
            // The first four values are just primers
            aVals[i] = aResidual[i];
        }
        else
        {
            // Get a predicted value
            iPredicted = predictValue(rvVals, i, ePredType);

            if( ePredType == PredXor1 || ePredType == PredXor2 )
            {
                // Decode the residual as the current value XOR predicted
                aVals[i] = aResidual[i] ^ iPredicted;
            }
            else
            {
                // Decode the residual as the current value plus predicted
                aVals[i] = aResidual[i] + iPredicted;
            }
        }
    }

    return True;
}

Bool CodecDriver::unpackResiduals(Vector<Float64>& rvResidual,
                                  Vector<Float64>& rvVals,
                                  PredictorType  ePredType)
{
    if( ePredType == PredXor1 || ePredType == PredXor2 )
        return False;

    if( ePredType == PredNULL )
    {
        rvVals = rvResidual;
        return True;
    }

    Float64 iPredicted;
    Int32 len = rvResidual.length();
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    rvVals.setLength(len);

    for( Int32 i = 0; i < len; i++ )
    {
        if( i < 4 )
        {
            // The first four values are just primers
            rvVals[i] = rvResidual[i];
        }
        else
        {
            // Get a predicted value
            iPredicted = predictValue(rvVals, i, ePredType);

            // Decode the value as the residual plus predicted
            rvVals[i]   = rvResidual[i] + iPredicted;
        }
    }

    return True;
}

Int32 CodecDriver::predictValue(Vector<Int32>& vVal,
                                Int32 iIndex,
                                PredictorType ePredType)
{
    Int32* aVals = (Int32 *) rvVals;
    JtInt32 iPredicted,
             v1 = aVals[iIndex-1],
             v2 = aVals[iIndex-2],
             v3 = aVals[iIndex-3],
             v4 = aVals[iIndex-4];

    switch( ePredType )
    {
        default:
        case PredLag1:
        case PredXor1:
            iPredicted = v1;
            break;

        case PredLag2:
        case PredXor2:
            iPredicted = v2;
            break;

        case PredStride1:
            iPredicted = v1 + (v1 - v2);
            break;

        case PredStride2:
            iPredicted = v2 + (v2 - v4);
            break;

        case PredStripIndex:
            if( v2 - v4 < 8 && v2 - v4 > -8 )
                iPredicted = v2 + (v2 - v4);
            else
                iPredicted = v2 + 2;
            break;

        case PredRamp:
            iPredicted = iIndex;
            break;
    }

    return iPredicted;
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
Float64 CodecDriverBase::predictValue(Vector<Float64>& vVal,
                                      Int32 iIndex,
                                      PredictorType ePredType)
{
    Float64* aVals = (Float64 *) rvVals;
    Float64 iPredicted,
             v1 = aVals[iIndex-1],
             v2 = aVals[iIndex-2],
             v3 = aVals[iIndex-3],
             v4 = aVals[iIndex-4];

    switch( ePredType )
    {
        default:
        case PredLag1:
            iPredicted = v1;
            break;

        case PredLag2:
            iPredicted = v2;
            break;

        case PredStride1:
            iPredicted = v1 + (v1 - v2);
            break;

        case PredStride2:
            iPredicted = v2 + (v2 - v4);
            break;

        case PredStripIndex:
            if( v2 - v4 < 8 && v2 - v4 > -8 )
                iPredicted = v2 + (v2 - v4);
            else
                iPredicted = v2 + 2;
            break;

        case PredRamp:
            iPredicted = iIndex;
            break;
    }

    return iPredicted;
}
```

B.1.6 CodecDriver2 class

```
//
// A class that deals with the conversions from SYMBOL to VALUE and
// provides end-consumer APIs for using the codecs.
//
class CodecDriver2
{
  public:

    // ---------- Internal Types ----------
    typedef enum {
        CodecNull       = 0,    // Null Codec
        CodecBitLength  = 1,    // Bitlength Codec
        CodecArithmetic = 3,    // Arightmetic Codec
        CodecChopper    = 4     // Chopper Pseudo-codec
    } CodecType;
    // Type of value predictor
    typedef enum {
            PredLag1       = 0, // Predicts as last values
            PredLag2       = 1, // Predicts as value before last
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        PredStride1    = 2, // Predict using stride from last two values
        PredStride2    = 3, // Predict using stride from values 2 and 4 back.
        PredStripIndex = 4, // Domain-specific tristrip index list predictor
        PredRamp       = 5, // Predict value i as i
        PredXor1       = 6, // Predict as last, but use xor instead of subtract
        PredXor2       = 7, // Predict as last, but use xor instead of subtract
        PredNULL       = 8, // No prediction.
        PredMean2      = 9, // Predict as mean of last two values
        PredMean3      = 10,// Predict as mean of last three values
        PredMean4      = 11 // Predict as mean of last four values
} PredictorType;

static Bool unpackResiduals(const Veci    &rvResidual,
                            Veci          &rvVals,
                            PredictorType  ePredType);
static Bool unpackResiduals(const Vecu    &rvResidual,
                            Vecu          &rvVals,
                            PredictorType  ePredType);

static Float64 log2( Float64 x ) { return (log(x) / 0.69314718055994531); }

//////////////////////////////
// Convenience Methods
//////////////////////////////

////////////////////
//
// NOTE: These methods "CANNOT" be used with the 9.x serialization of
//       Coordinates, Normals, Colours, and Tex Coordinate do the
//       way the hash value is calculated.
//
/////

void assemble( const Vecu    &vMant,    // Output vector of mantissae          *
               const Vecu    &vExp,     // Output vector of exponents          *
               UInt32         iComp,    // Active data componenet per record   *
               UInt32         nComp,    // Number of component per data record *
               UInt32         nIdx,     // Length of index list                *
               UInt32        &uHash,    // Hash value                          *
               Float32       *opData )  // Ptr to data                         *
{
    __FloatInt32Union fi;
    UInt32 *pExp  = (UInt32*)vExp.ptr(),
           *pMant = (UInt32*)vMant.ptr();
    for (UInt32 i = 0 ; i < nIdx ; i++) {
        fi.ui  = pExp[i] << 23
               | pMant[i];
        opData[i * nComp + iComp] = fi.f;
    }
    uHash = ::hash32(vMant.ptr(), nIdx, uHash);
    uHash = ::hash32(vExp.ptr(), nIdx, uHash);
}
void assemble( const Vecu    &vMant0,    // Output vector of mantissae          *
               const Vecu    &vMant1,    // Output vector of mantissae          *
               const Vecu    &vExp,      // Output vector of exponents          *
               UInt32         iComp,     // Active data componenet per record   *
               UInt32         nComp,     // Number of component per data record *
               UInt32         nIdx,      // Length of index list                *
               UInt32        &uHash,     // Hash value                          *
               Float64       *opData )   // Ptr to data                         *
{
    __FloatInt64Union fi;
    UInt32 *pExp  =  (UInt32*)vExp.ptr(),
           *pMant1 = (UInt32*)vMant1.ptr(),
           *pMant0 = (UInt32*)vMant0.ptr();
    for (UInt32 i = 0 ; i < nIdx ; i++) {
        fi.ui  = (UInt64(pExp[i]) << 52) | (UInt64(pMant1[i]) << 31) | pMant0[i];
        opData[i * nComp + iComp] = fi.f;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        }
        uHash = ::hash32(vMant0.ptr(), nIdx, uHash);
        uHash = ::hash32(vMant1.ptr(), nIdx, uHash);
        uHash = ::hash32(vExp.ptr(), nIdx, uHash);
    }


  protected:
    static Int32 _predictValue( const Int32 *vVal, Int32 iIndex,
                                PredictorType ePredType );
};

Bool
CodecDriver2::unpackResiduals( const Veci      &rvResidual,
                               Veci            &rvVals,
                               PredictorType    ePredType )
{
    const Int32 len = rvResidual.length();
    Int32 iPredicted;
    rvVals.setLength(len);
    Int32 *aVals = rvVals.ptr();
    const Int32 *aResidual   = rvResidual.ptr();
    for ( Int32 i = 0 ; i < len ; i++ ) {
        if (i < 4) {
            // The first four values are just primers
            aVals[i] = aResidual[i];
        } else {
            // Get a predicted value
            iPredicted = _predictValue(rvVals.ptr(), i, ePredType);

            if (ePredType == PredXor1 || ePredType == PredXor2) {
                // Encode the residual as the current value XOR predicted
                aVals[i] = aResidual[i] ^ iPredicted;
            } else  {
                // Encode the residual as the current value plus predicted
                aVals[i] = aResidual[i] + iPredicted;
            }
        }
    }

    return True;
}

Bool
CodecDriver2::unpackResiduals( const Vecu     &rvResidual,
                               Vecu           &rvVals,
                               PredictorType  ePredType )
{
    return unpackResiduals(*((const Veci*)&rvResidual),
                           *((Veci*) &rvVals),
                           ePredType);
}

Int32
CodecDriver2::_predictValue( const Int32*  paVals,
                             Int32         iIndex,
                             PredictorType ePredType )
{
    Int32 iPredicted;
    switch (ePredType) {

        default:
        case PredLag1:      // Predicts as last value
        case PredXor1:      // Predicts as last value
            iPredicted    = paVals[iIndex-1];
            break;
        case PredLag2:      // Predicts as value before last
        case PredXor2:      // Predicts as value before last
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
            iPredicted    = paVals[iIndex-2];
            break;
        case PredStride1: // Predict using stride from last two values
            iPredicted = paVals[iIndex-1] + (paVals[iIndex-1] - paVals[iIndex-2]);
            break;
        case PredStride2: // Predict using stride from values 2 and 4 back.
            iPredicted = paVals[iIndex-2] + (paVals[iIndex-2] - paVals[iIndex-4]);
            break;
        case PredStripIndex:// Domain-specific tristrip index list predictor
            // Predict using stride from last two values 2 and 4 back.
            if (paVals[iIndex-2] - paVals[iIndex-4] < 8 && paVals[iIndex-2] -
paVals[iIndex-4] > -8)
                iPredicted = paVals[iIndex-2] + (paVals[iIndex-2] - paVals[iIndex-4]);
            else
                iPredicted = paVals[iIndex-2] + 2;
            break;
        case PredRamp:     // Predict value i as i
            iPredicted = iIndex;
            break;
        case PredMean2:     // Predict as mean of last two values
            iPredicted = (paVals[iIndex-1] + paVals[iIndex-2]) >> 1;
            break;
        case PredMean3:     // Predict as mean of last three values
            iPredicted = (paVals[iIndex-1] + paVals[iIndex-2] + paVals[iIndex-3]) / 3;
            break;
        case PredMean4:     // Predict as mean of last four values
            iPredicted = ( paVals[iIndex-1] + paVals[iIndex-2] +
                           paVals[iIndex-3] + paVals[iIndex-4] ) >> 2;
            break;
    }

    return iPredicted;
}
```

B.2 Bitlength decoding classes

The following sub-sections contain a sample implementation of the decoding portion of the Bitlength CODEC algorithm. A summary technical explanation of the Bitlength CODEC can be found in 13.2.2 Bitlength CODEC.

B.2.1 BitLengthCodec class

```
class BitLengthCodec
{
  public:
       // This method decodes a given stream of symbols into their values.
       // The stream is described by the codec driver
       Bool decode(CodecDriver* pDriver);
       Bool encode(CodecDriver* pDriver);

       Int32 nBitsInSymbol(Int32 iSymbol) const;
       Int32 gStepBits = 2;
};

Bool BitLengthCodec::encode( CodecDriverBase *pDriver )
{
    JtInt32  iSymbol,               // Symbol value to encode
             iNextSymbol;           // Lookahead symbol value
    JtInt32  cNumCurBits = 0,       // Current field width
             cNumSymBits = 0,       // Number of bits in iSymbol
             cNumSymBitsNext = 0,   // Number of bits in iNextSymbol
             nSymbols;              // Number of symbols

    // Reset driver's symbol iterator and get number of symbols.
    pDriver->resetInputSymbolIter();
    nSymbols = pDriver->numInputSymbols();
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    // Short circuit for null array of symbols
    if (nSymbols <= 0)
        return JtTrue;

    // Get list of input symbols/counts/contexts from driver
    const Int32 *aiSymbol, *acSymCount, *acCumCount, *acTotalCount, *aiSymContext;
    pDriver->getInputSymbols(aiSymbol, acSymCount, acCumCount, acTotalCount,
aiSymContext);

    // Prime the symbol pump
    iNextSymbol = aiSymbol[0];
    cNumSymBitsNext = _nBitsInSymbol(iNextSymbol);

    // Iterate over the remaining symbols
    UInt32 uCodeText = 0;
    Int32  nBits     = 0;
    for (Int32 i = 0; i < nSymbols; i++ ) {

        // Get the next symbol
        iSymbol = iNextSymbol;
        cNumSymBits = cNumSymBitsNext;
        if (i < nSymbols - 1)
            iNextSymbol = aiSymbol[i + 1];

        // Find out how many bits it takes to represent the symbol
        cNumSymBitsNext = _nBitsInSymbol(iNextSymbol);

        // Look ahead
        if (cNumCurBits > cNumSymBits && cNumSymBitsNext > cNumSymBits) {
            // Case 0: cur > next
            if (cNumCurBits > cNumSymBitsNext) {
                // Number of bits it will cost to adjust down
                // and then back up to NextSymBits from here.
                Int32 cCost = ((cNumCurBits     - cNumSymBits + cStepBits - 1) /
cStepBits
                            +  (cNumSymBitsNext - cNumSymBits + cStepBits - 1) /
cStepBits)
                            * 2;
                // Number of bits that will be saved by stepping
                // down the number of encoding bits.
                Int32 cSave = ((cNumCurBits - cNumSymBits) / cStepBits)
                            * cStepBits;
                if (cCost > cSave)
                    cNumSymBits = cNumSymBitsNext;
            }
            // Case 1: next > cur
            else {
                // Number of bits it will cost to adjust down
                // and then back up to here from here.
                Int32 cCost = (cNumCurBits - cNumSymBits + cStepBits - 1)
                            / cStepBits * 4;
                // Number of bits that will be saved by stepping
                // down the number of encoding bits.
                Int32 cSave = (cNumCurBits - cNumSymBits) / cStepBits
                            * cStepBits;
                if (cCost > cSave)
                    cNumSymBits = cNumCurBits;
            }
        }

        // Emit bits to adjust the code length to match
        // Note: We could look ahead and see if it's worth
        // bothering to adjust...
        UInt32 uPrefix = 0;
        Int32  nPrefixBits = 0;

        // Stay at the same number of bits
        if (cNumCurBits >= cNumSymBits &&
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        cNumCurBits <= cNumSymBits + cStepBits - 1)
    {
        // A 0 bit indicates that the bit length stays the same
        uPrefix = 0;
        nPrefixBits = 1;
    }
    // Go up or down some multiple of cStepBits
    else {
        // Start with a 1 bit to indicate that we are
        // changing the number of bits.
        uPrefix = 1;
        nPrefixBits = 1;
        // Accumulate a 1 for every step to increase
        while (cNumCurBits < cNumSymBits) {
            uPrefix <<= 1;
            uPrefix |= 0x1;
            nPrefixBits++;
            cNumCurBits += cStepBits;
        }
        // Accumulate a 0 for every step to decrease
        // Note that we never both increas and decrease!
        while (cNumCurBits > cNumSymBits + cStepBits - 1) {
            uPrefix <<= 1;
            nPrefixBits++;
            cNumCurBits -= cStepBits;
        }
        // Terminator bit is opposite of last bit
        uPrefix = (uPrefix << 1) | ((uPrefix & 0x1) ^ 0x1);
        nPrefixBits++;;
    }

    // Turn the symbol value itself into bitcode
    // Equivalent: pDriver->addCodeText(uPrefix, nPrefixBits);
    // Equivalent: pDriver->addCodeText(iSymbol, cNumCurBits);

    // Output prefix code
    if (nBits + nPrefixBits <= 32) {
        uCodeText <<= nPrefixBits;
        uCodeText |=  uPrefix;
        nBits     +=  nPrefixBits;
    }
    else {
        // Flush the local bit cache and start a new one
        pDriver->addCodeText(uCodeText, nBits);
        uCodeText =  uPrefix;
        nBits     =  nPrefixBits;
    }
    // Output symbol
    if (nBits + cNumCurBits <= 32) {
        if (cNumCurBits == 32) {
            uCodeText  =  iSymbol;
        } else {
            uCodeText <<= cNumCurBits;
            uCodeText |=  iSymbol & ((1<<cNumCurBits)-1);
        }
        nBits +=  cNumCurBits;
    }
    else {
        // Flush the local bit cache and start a new one
        pDriver->addCodeText(uCodeText, nBits);

        //uCodeText = the lowest order cNumCurBits of iSymbol
        uCodeText = iSymbol & ((((cNumCurBits>>5)^1)<<cNumCurBits)-1);
        nBits     = cNumCurBits;
    }

}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    // Flush the local bit cache
    pDriver->addCodeText(uCodeText, nBits);

    return True;
}

Int32 BitLengthCodec::_nBitsInSymbol(Int32 iSymbol) const
{
    // Zero is the only number that can be encoded in a
    // single bit with this scheme, so this method returns
    // 0 bits for a symbol value of zero!  A side-effect of
    // this is that this method never says there are any
    // 1-bit symbols (See below).
    if (iSymbol == 0)
        return 0;

    // Note: This calculation assumes that iSymbol can be
    // either positive or negative!  Therefore, the number
    // of bits, say for 5 is given as 4 instead of 3 so
    // that -5 can also be encoded.
    Int32 cMaxCodeSpan = ::jtabs(iSymbol);
    Int32 i, nBits;
    for (i = 1, nBits = 0 ; i <= cMaxCodeSpan && nBits < 31 ; i += i, nBits++);
    return nBits + 1;
}

Bool BitLengthcodec::decode(CodecDriver* pDriver)
{
       Int32 iBit;                 // Codetext bit number
       Int32 nBits = 0;            // Number of codetext bits decoded so far
       Int32 nTotalBits = 0;       // Total number of codetext bits expected
       Int32 nValBits = 0;         // Number of accumulated value bits
       Int32 iContext = 0;         // Probability context number
       Int32 iSymbol;              // Decoded symbol value
       UInt32 uVal = 0;            // Current chunk of codetext bits
       UInt32 uAccVal = 0;         // Number of valid bits returned from
                                   // getNextCodeText
       UInt32 uLastIncBit = 0;     // Used to calculate whether terminator bit
                                   // is 0 or 1
       Int32 cNumCurBits = 0;      // Current field width in bits
       Int32 nAccBits = 0;         // Number of bits accum'ed in uAccVal
       Int32 iDecodeState = 0;     // State of decoder; see below

       // Get codetext from the driver and loop over it until it's gone!
       pDriver->getDecodeData(iContext, nTotalBits);

       while( nBits < nTotalBits ) {
          // Get the next 32 bits from the input stream
          pDriver->getNextCodeText(uVal, nValBits);
          // Scan through each bit either walking the Huffman code
          // tree or accumulating escaped bit values.
          Int32 n = min(32, min(nValBits, nTotalBits - nBits));
          for( iBit = 0; iBit < n ; iBit++ ) {
              // Code-accumulation mode is handled is this block
              // as many bits at a time as possible.
              if( iDecodeState == 2 ) {
                 // Slice off as many bits as we can all at once.
                 Int32 m = min(n - iBit, cNumCurBits - nAccBits);
                    if( m < 32 ) {
                       uAccVal <<= m;
                       uAccVal |= ((uVal >> (32 - m)) & ((1 << m) - 1));
                       nAccBits += m;
                       iBit += m - 1;
                       // Advance the bit-marching counters
                       uVal <<= m;
                       nBits += m;
                       nValBits -= m;
              } else {
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
                uAccVal = uVal;
                nAccBits += m;
                iBit += m - 1;

                // Advance the bit-marching counters
                uVal = 0;
                nBits += m;
                nValBits -= m;
            }
            if( nAccBits >= cNumCurBits ) {
                // Convert and sign-extend the symbol
                iSymbol = Int32(uAccVal);
                iSymbol <<= (32 - cNumCurBits);
                iSymbol >>= (32 - cNumCurBits);
                // Output the symbol and restart
                pDriver->addOutputSymbol(iSymbol, iContext);
                iDecodeState = 0;
                uAccVal  = 0;
                nAccBits = 0;
            }
            } else {
              // All other decode states are handled one bit at a time
              // inside this block.
              // Get the next bit
              uAccVal = (uVal >> 31);

              switch( iDecodeState ) {
                // DecodeState = 0: Recognize prefix bit (0=Same size
                // code, 1=Different size code).
                case 0:
                    // Recognize "same length" prefix code
                    if( uAccVal == 0 ) {
                      iDecodeState = 2;
                    } else {
                      // Recognize "different length" prefix code
                      iDecodeState = 1;
                      uLastIncBit = 2;
                    }
                    uAccVal  = 0;
                    break;

                    case 1: // Length adjustment mode
                      // Recognize the terminator bit
                      if( uLastIncBit != 2 && (uAccVal ^ uLastIncBit) ) {
                        iDecodeState = 2;
                        uLastIncBit = 2;
                      } else {
                        // Recognize the "increment" prefix code
                        if( uAccVal == 1 ) {
                          cNumCurBits += cStepBits;
                        } else {
                          // Recognize the "decrement" prefix code
                          cNumCurBits -= cStepBits;
                        }
                        uLastIncBit = uAccVal;
                      }
                        uAccVal  = 0;
                        break;
                }
                  // Advance the bit-marching counters that keep track of the
                  // "current codetext bit", and how many bits are left.
                  uVal <<= 1;
                  nBits++;
                 nValBits--;
              }
        }
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        // If the last symbol was zero and the current bit length
        // is also zero, then the above loop terminated before
        // actually decoding the last zero-valued symbol.  Test
        // for that condition here and decode it if necessary.
        if( iDecodeState == 2 && cNumCurBits == 0 ) {
                // Output the symbol and restart
                iSymbol = Int32(0);
                pDriver->addOutputSymbol(iSymbol, iContext);
        }

        return True;
}
```

B.2.2 BitLengthCodec2 class

```
class BitLengthCodec2 : public Codec2
{
  public:

    Bool encode( const Veci   &vValues,
                 Veci         &ovOOBValues,
                 Vecu         &ovCodeText,
                 Int32        &onBitsCodeText,
                 ProbContext2 *pProbCntx       );
    Bool decode( Int32         nValues,
                 const Vecu   &vCodeText,
                 Int32         nBitsCodeText,
                 Veci         &ovValues );

  protected:
   Int32 _nBitsInSymbol(Int32 iSymbol) const;
    Bool getNextCodeText (UInt32 &uCodeText, Int32 &nBits);

    Vecu     *_pvCodeText;
    Int32    *_pcCodeTextLen;
    Int32     _iCurCodeText;
    Vecus     _vnValBits;
};

#define GetSignedBits(iOut, n)                      \
    GetUnsignedBits(iOut,n)                         \
    iOut <<= (32 - n);                              \
    iOut >>= (32 - n);

#define GetUnsignedBits(uOut, n)                    \
    if (n == 0) uOut = 0;                           \
    else if (nValBits >= n) {                       \
        uOut = uVal >> (32 - n);                    \
        uVal    <<= n;                              \
        uVal &= (n==32)-1;                          \
        nValBits -= n;                              \
        nBits    += n;                              \
    }                                               \
    else {                                          \
        Int32 _nLBits = nValBits;                   \
        uOut = uVal >> (32 - n);                    \
        nBits    += _nLBits;                        \
        getNextCodeText (uVal, nValBits);           \
        Int32 _nRBits = (n - _nLBits);              \
        uOut |= uVal >> (32 - _nRBits);             \
        uVal    <<= _nRBits;                        \
        uVal &= (_nRBits==32)-1;                    \
        nValBits -= _nRBits;                        \
        nBits    += _nRBits;                        \
    }

Bool
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
BitLengthCodec2::encode(const Veci   &vValues,
                        Veci         &ovOOBValues,
                        Vecu         &ovCodeText,
                        Int32        &onBitsCodeText,
                        ProbContext2 *pProbCntx      )
{
    Int32  i, j, k;
    Int32  iSymbol;               // Symbol value to encode
    Int32  cSymBits = 0,          // Number of bits in iSymbol
           nValues;               // Number of values to encode
    // Initialize output state
    ovOOBValues.setLength(0);
    ovCodeText.setLength(0);
    onBitsCodeText = 0;
    _pvCodeText = &ovCodeText;
    _pcCodeTextLen = &onBitsCodeText;

    // Short circuit for null array of values
    nValues = vValues.length();
    if (nValues <= 0)
        return True;

    _vnValBits.setLength(nValues);
    UInt16 *paiSymBits = _vnValBits.ptr();
    const Int32 *paiValues = vValues.ptr();
    // Find the minimum value and compute how many bits each value takes
    Int32 iMinSymbol = 0x7fffffff;
    Int32 iMaxSymbol = 0x80000000;
    Float32 fMean = 0.0f;
    for (i = 0 ; i < nValues ; i++) {
        iMinSymbol = ::min(iMinSymbol, paiValues[i]);
        iMaxSymbol = ::max(iMaxSymbol, paiValues[i]);
        fMean += Float32(paiValues[i]);
    }
    fMean /= nValues;
    Int32 iMean = Int32(fMean);
    for (i = 0 ; i < nValues ; i++) {
        paiSymBits[i] = _nBitsInSymbol(::abs(paiValues[i] - iMean)) + 1;
    }
    // A "block" is: 3 bits of number of bits (repeats while value is either 3 or -4 for
larger width changes)
    //               4 bits of block length
    Int32 cBlkLenBits = 4; // Number of bits used to express a block length. 0 means 0.
    Int32 cBlkValBits = 4; // *Delta* number of bits to express the current field width
    Int32 cBlkHdrBits = cBlkLenBits + cBlkValBits;
    Bool bMerged;
    // Block-forming: Merge Down/Up blocks
    do {
        bMerged = False;
        Int32 iPrevRunPos = 0,
              iCurRunPos  = 0,
              iNextRunPos = 0;
        while (iCurRunPos < nValues - 1) {
            // Advance the next run
            iNextRunPos++;
            while (iNextRunPos < nValues - 1 &&
                   paiSymBits[iNextRunPos] == paiSymBits[iNextRunPos-1])
            {
                iNextRunPos++;
            }
            if (iNextRunPos >= nValues)
                break;
            Int32  wab = (iCurRunPos - iPrevRunPos),
                   wbc = (iNextRunPos - iCurRunPos);
            if (wab == 0) {
                iCurRunPos = iNextRunPos;
                continue;
            }
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        else if (wbc == 0) {
            continue;
        }
        // If we've bitten off more than one block's worth of data
        // we must start a new block.  Length 0 is allowed because
        // we may need to insert multiple consecutive block headers
        // in order to change the field width more bits than can be
        // represented in cBlkValBits.
        if (wab > (1 << cBlkLenBits)) {
            iPrevRunPos += (1 << cBlkLenBits);
        }
        else if (wab == (1 << cBlkLenBits)) {
            iPrevRunPos = iCurRunPos;
            iCurRunPos  = iNextRunPos;
            continue;
        }
        UInt16 &ua = paiSymBits[iPrevRunPos],
               &ub = paiSymBits[iCurRunPos],
               &uc = paiSymBits[iNextRunPos];
        // If the runs go "down-up"
        if (ua > ub && ub < uc) {
            // Test if we should increase ub to the lesser of ua and uc
            if (ua < uc) {
                // Test if we should increase ub to ua
                if (wbc * (ua - ub) <= cBlkHdrBits) {
                    for (j = iCurRunPos ; j < iNextRunPos ; j++)
                        paiSymBits[j] = ua;
                    iCurRunPos = iNextRunPos;
                    continue;
                }
            }
            else if (ua > uc) {
                // Test if we should increase ub to uc
                if (wbc * (uc - ub) <= cBlkHdrBits) {
                    for (j = iCurRunPos ; j < iNextRunPos ; j++)
                        paiSymBits[j] = uc;
                    iNextRunPos = iCurRunPos;
                    bMerged = True;
                    continue;
                }
            }
            else {  // ua == uc
                // Test if we should increase ub to ua/uc
                if (wbc * (ua - ub) <= 2 * cBlkHdrBits) {
                    for (j = iCurRunPos ; j < iNextRunPos ; j++)
                        paiSymBits[j] = ua;
                    iCurRunPos  = iPrevRunPos;
                    iNextRunPos = iPrevRunPos;
                    bMerged = True;
                    continue;
                }
            }
        }

        // Shift down the runs
        iPrevRunPos = iCurRunPos;
        iCurRunPos  = iNextRunPos;
    }
} while (bMerged);
// Block forming: Merge down/down and up/up runs
do {
    bMerged = JtFalse;
    Int32 iPrevRunPos = 0,
          iCurRunPos  = 0,
          iNextRunPos = 0;
    while (iCurRunPos < nValues - 1) {
        // Advance the next run
        iNextRunPos++;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
while (iNextRunPos < nValues - 1 &&
       paiSymBits[iNextRunPos] == paiSymBits[iNextRunPos-1])
{
    iNextRunPos++;
}
if (iNextRunPos >= nValues)
    break;
Int32  wab = (iCurRunPos - iPrevRunPos),
       wbc = (iNextRunPos - iCurRunPos);
if (wab == 0) {
    iCurRunPos = iNextRunPos;
    continue;
}
else if (wbc == 0) {
    continue;
}
// If we've bitten off more than one block's worth of data
// we must start a new block.  Length 0 is allowed because
// we may need to insert multiple consecutive block headers
// in order to change the field width more bits than can be
// represented in cBlkValBits.
if (wab > (1 << cBlkLenBits)) {
    iPrevRunPos += (1 << cBlkLenBits);
}
else if (wab == (1 << cBlkLenBits)) {
    iPrevRunPos = iCurRunPos;
    iCurRunPos  = iNextRunPos;
    continue;
}
UInt16 &ua = paiSymBits[iPrevRunPos],
       &ub = paiSymBits[iCurRunPos],
       &uc = paiSymBits[iNextRunPos];
// If the runs go "up-down"
if (ua < ub && ub > uc) {
    /* Empty */
}
// If the runs go "up-up"
else if (ua < ub && ub < uc) {
    // Test if we should increase ua to ub
    if (wab * (ub - ua) < cBlkHdrBits) {
        for (j = iPrevRunPos ; j < iCurRunPos ; j++)
            paiSymBits[j] = ub;
        iCurRunPos = iNextRunPos;
        bMerged = True;
        continue;
    }
    // Test if we should increase ub to uc
    else if (wbc * (uc - ub) < cBlkHdrBits) {
        for (j = iCurRunPos ; j < iNextRunPos ; j++)
            paiSymBits[j] = uc;
        iNextRunPos = iCurRunPos;
        bMerged = True;
        continue;
    }
}
// If the runs go "down-down"
else if (ua > ub && ub > uc) {
    // Test if we should increase ub to ua
    if (wbc * (ua - ub) < cBlkHdrBits) {
        for (j = iCurRunPos ; j < iNextRunPos ; j++)
            paiSymBits[j] = ua;
        iCurRunPos = iNextRunPos;
        continue;
    }
}

// Shift down the runs
iPrevRunPos = iCurRunPos;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
            iCurRunPos  = iNextRunPos;
        }
    } while (bMerged);

    // Compute the total bits
    Int32 cTotalBits = paiSymBits[0] + cBlkHdrBits;
    for (i = 1 ; i < nValues ; i++) {
        cTotalBits += paiSymBits[i];
        if (paiSymBits[i] != paiSymBits[i-1])
            cTotalBits += cBlkHdrBits;
    }
    Int32 cFixedWidBits = nValues * _nBitsInSymbol(iMaxSymbol - iMinSymbol);

    /////
    // If the fixed-width total bits are better, then write out the values
    // in a single fixed-width format.
    /////
    if (cFixedWidBits < cTotalBits) {
        // Write the fixed-width tag
        addCodeText(0, 1);

        // Find the minimum symbol, and encode all further symbols as
        // positive number excess of this minimum value.  This will save
        // us at least half a bit on average because _nBitsInSymbol()
        // doesn't have to add a bit to account for negative numbers.
        Int32 iMinSymbol = 0x7fffffff;
        Int32 iMaxSymbol = 0x80000000;
        for (Int32 i = 0 ; i < nValues ; i++) {
            iMinSymbol = ::min(iMinSymbol, paiValues[i]);
            iMaxSymbol = ::max(iMaxSymbol, paiValues[i]);
        }
        Int32 cBitsInMinSymbol;
        Int32 cBitsInMaxSymbol;
        // If some symbols are negative, then we must include a sign bit
        cBitsInMinSymbol = _nBitsInSymbol(iMinSymbol) + 1;
        cBitsInMaxSymbol = _nBitsInSymbol(iMaxSymbol) + 1;
        // Write the min and max symbols into the stream
        addCodeText(cBitsInMinSymbol, 6);
        addCodeText(cBitsInMaxSymbol, 6);
        addCodeText((UInt32) iMinSymbol, cBitsInMinSymbol);
        addCodeText((UInt32) iMaxSymbol, cBitsInMaxSymbol);

        // Use a fixed number of bits for all symbols
        cSymBits = _nBitsInSymbol(iMaxSymbol - iMinSymbol);

        // Iterate over the remaining symbols
        UInt32 uCodeText = 0;
        for (Int32 i = 0; i < nValues; i++ ) {

            // Get the next symbol
            iSymbol = paiValues[i];

            // Write it
            uCodeText = iSymbol - iMinSymbol;
            addCodeText(uCodeText, cSymBits);
        }
    }
    /////
    // Otherwise, encode with variable-length fields
    /////
    else {
        // Write the variable-width tag
        addCodeText(1, 1);

        // Write out the mean value
        addCodeText(iMean, 32);
        addCodeText(cBlkValBits, 3);
        addCodeText(cBlkLenBits, 3);
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        // Set the initial field-width
        Int32 cMaxFieldDecr = -(1 << (cBlkValBits - 1)),       // -ve number
              cMaxFieldIncr =  (1 << (cBlkValBits - 1)) - 1;   // +ve number
        Int32 cCurFieldWidth = 0;
        Int32 cTargFieldWidth;
        for (Int32 ii = 0 ; ii < nValues ;) {
            // Adjust the current field width to the target field width
            cTargFieldWidth = paiSymBits[ii];
            {
                if (cCurFieldWidth <= cTargFieldWidth) {
                    while (cTargFieldWidth - cCurFieldWidth >= cMaxFieldIncr) {
                        addCodeText(cMaxFieldIncr, cBlkValBits);
                        cCurFieldWidth += cMaxFieldIncr;
                    }
                }
                else {
                    while (cTargFieldWidth - cCurFieldWidth <= cMaxFieldDecr) {
                        addCodeText(cMaxFieldDecr, cBlkValBits);
                        cCurFieldWidth += cMaxFieldDecr;
                    }
                }
                addCodeText(cTargFieldWidth - cCurFieldWidth, cBlkValBits);
                cCurFieldWidth = cTargFieldWidth;
            }

            // Write out the run length
            for (j = ii+1 ; j < ii + (1 << cBlkLenBits) - 1 && j < nValues ; j++)
                if (paiSymBits[ii] != paiSymBits[j])
                    break;
            addCodeText(j - ii, cBlkLenBits);

            // Write out the data bits for the run
            for (k = ii ; k < j ; k++)
                addCodeText(paiValues[k] - iMean, cCurFieldWidth);

            // Advance to the end of the run
            ii = j;
        }
    }

    return True;
}

Bool
BitLengthCodec2::decode( Int32         nValues,
                         const Vecu   &vCodeText,
                         Int32         nBitsCodeText,
                         Veci         &ovValues )
{
    Int32  nBits = 0,          // Number of codetext bits decoded so far
           nTotalBits = 0,     // Total number of codetext bits expected
           nValBits = 0,       // Number of accumulated value bits
           iSymbol;            // Decoded symbol value
    UInt32 uVal = 0;           // Current chunk of codetext bits
    Int32  cNumCurBits = 0;    // Current field width in bits
    Int32  cBitsInMinSymbol;   // Number of bits in the minimum symbol
    Int32  cBitsInMaxSymbol;   // Number of bits in the maximum symbol
    Int32  iMinSymbol = 0;     // The minimum symbol value.  Used as bias.
    Int32  iMaxSymbol = 0;     // The maximum symbol value.  Used as bias.
    Int32  nSyms = 0;          // Number of symbols read so far
    Int32 *paiValues;          // Pointer into ovValues where we write decoded values

    // Get codetext from the driver and loop over it until it's gone!
    ovValues.setLength(nValues);
    paiValues = ovValues.ptr();

    _iCurCodeText = 0;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    _pvCodeText = (Vecu*) &vCodeText;
    _pcCodeTextLen = &nBitsCodeText;

    /////
    // If the fixed-width total bits are better, then write out the values
    // in a single fixed-width format.
    /////
    // Read the variable-width tag
    Int32 iTmp;
    GetUnsignedBits(iTmp, 1); // 0 = Fixed-width, 1 = Variable width
    if (iTmp == 0) {
        // Read the min and max symbols from the stream
        GetUnsignedBits(cBitsInMinSymbol, 6);
        GetUnsignedBits(cBitsInMaxSymbol, 6);
        GetSignedBits  (iMinSymbol,       cBitsInMinSymbol);
        GetSignedBits  (iMaxSymbol,       cBitsInMaxSymbol);
        cNumCurBits = _nBitsInSymbol(iMaxSymbol - iMinSymbol);

        // Read each fixed-width field and output the value
        while (nBits < nTotalBits || nSyms < nValues) {
            GetUnsignedBits(iSymbol, cNumCurBits);
            iSymbol += iMinSymbol;
            *paiValues++ = iSymbol;
            nSyms++;
        }
    }
    /////
    // Otherwise, encode with variable-length fields
    /////
    else {
        // Write out the mean value
        Int32 iMean, cBlkValBits, cBlkLenBits;
        GetSignedBits(iMean, 32);
        GetUnsignedBits(cBlkValBits, 3);
        GetUnsignedBits(cBlkLenBits, 3);

        // Set the initial field-width
        Int32 cMaxFieldDecr = -(1 << (cBlkValBits - 1)),       // -ve number
              cMaxFieldIncr =  (1 << (cBlkValBits - 1)) - 1;   // +ve number
        Int32 cCurFieldWidth = 0, cDeltaFieldWidth, cRunLen, k;
        for (Int32 ii = 0 ; ii < nValues ;) {
            // Adjust the current field width to the target field width
            do {
                GetSignedBits(cDeltaFieldWidth, cBlkValBits);
                cCurFieldWidth += cDeltaFieldWidth;
            } while (cDeltaFieldWidth == cMaxFieldDecr || cDeltaFieldWidth ==
cMaxFieldIncr);

            // Read in the run length
            GetUnsignedBits(cRunLen, cBlkLenBits);

            // Read in the data bits for the run
            for (k = ii ; k < ii + cRunLen ; k++) {
                GetSignedBits(iTmp, cCurFieldWidth);
                *paiValues++ = iTmp + iMean;
            }

            // Advance to the end of the run
            ii += cRunLen;
        }
    }

    // Assert that we have consumed exactly all of the bits
    Assert(nValBits == 0);
    Assert(uVal == 0);

    return True;
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
Int32
BitLengthCodec2::_nBitsInSymbol(Int32 iSymbol) const
{
    // Zero is the only number that can be encoded in a
    // single bit with this scheme, so this method returns
    // 0 bits for a symbol value of zero!
    if (iSymbol == 0)
        return 0;

    // Note: This calculation assumes that iSymbol is
    // positive!
    Int32 cMaxCodeSpan = ::abs(iSymbol);
    Int32 i, nBits;
    for (i = 1, nBits = 0 ; i <= cMaxCodeSpan && nBits < 31 ; i += i, nBits++);
    return nBits;
}

Bool
BitLengthCodec2::getNextCodeText (UInt32 &uCodeText, Int32 &nBits)
{
    uCodeText = _pvCodeText->value(_iCurCodeText);
    nBits = ::min(32, *_pcCodeTextLen - 32 * _iCurCodeText);
    _iCurCodeText++;
    return True;
}
```

B.3 Arithmetic decoding classes

The following sub-sections contain a sample implementation of the decoding portion of the Arithmetic CODEC algorithm. A summary technical explanation of the Arithmetic CODEC can be found in 13.2.3 Arithmetic CODEC.

B.3.1 ArithmeticProbabilityRange class

```
class ArithmeticProbabilityRange
{
public:
   UInt16 low_count;
   UInt16 high_count;
   UInt16 scale;
};
```

B.3.2 ArithmeticCodec class

```
class ArithmeticCodec
{
  public:
   ArithmeticCodec() :
       code = 0x0000,
       low = 0x0000,
       high = 0xffff,
       nUnderflowBits = 0,
       bitBuffer =0x00000000,
       nBits = 0 {}

   // Encodes a list of symbols.
   Bool encode( CodecDriver* pDriver );

   // Decodes a list of symbols. The codecDriver provides the range
   // info for the symbols to decode.  It also stores the symbols as
   // they are decoded.
   Bool decode( CodecDriver* pDriver );

  private:
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    void encodeSymbol(ArithmeticProbabilityRange &sym );
   // Remove the most recently decoded symbol from the front of the
   // list of encoded symbols.
    Bool removeSymbolFromStream( ArithmeticProbabilityRange& sym,
                                 CodecDriver* pDriver);
    void flushEncoder();
    void writeBit(UInt16 bit );

   //State variables used in decoding.
   UInt16 code;      // Present input code value, for decoding only
   UInt16 low;       // Start of the current code range
   UInt16 high;      // End of the current code range

   UInt32 bitBuffer; // Temporary i/o buffer
   Int32  nBits;     // Number of bits in _bitBuffer
};

Bool ArithmeticCodec::encode( CodecDriver *pDriver )
{
    _pDriver = pDriver;

    // Initialize the encoder state
    _iCurContext = 0;
    _iCurSymbol  = 0;
    _low  = 0x0000;
    _high = 0xffff;
    _nUnderflowBits = 0;

    _bitBuffer = 0x00000000;
    _nBits = 0;
    const Int32 *aiSymbol;
   const Int32 *acSymCount;
   const Int32 *acCumCount;
   const Int32 *acTotalCount;
   const Int32 *aiSymContext;

    pDriver->getInputSymbols(aiSymbol, acSymCount, acCumCount, acTotalCount,
aiSymContext);

    ArithmeticProbabilityRange newSymbolRange;
    Int32 nSymbols = pDriver->numInputSymbols();
    for (Int32 i = 0; i < nSymbols; i++ ) {
#ifdef DEBUG
        Assert( acCumCount[i]                <= 0xffff );
        Assert( acSymCount[i] + acCumCount[i] <= 0xffff );
        Assert( acTotalCount[i]              <= 0xffff );
#endif
        newSymbolRange.low_count  = acCumCount[i];
        newSymbolRange.high_count = acSymCount[i] + acCumCount[i];
        newSymbolRange.scale      = acTotalCount[i];
       Assert (acSymCount[i] > 0);

        encodeSymbol(newSymbolRange);
    }

    flushEncoder();

    _pDriver = 0;
    return True;
}

void ArithmeticCodec::encodeSymbol(ArithmeticProbabilityRange &sym )
{
    // These three lines rescale _high and _low for the new symbol.
    UInt32 range = UInt32(_high - _low) + 1;
    _high  = _low + (range * sym.high_count) / sym.scale - 1;
    _low   = _low + (range * sym.low_count ) / sym.scale;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    // This loop turns out new bits until _high and _low are far enough
    // apart to have stabilized.
    for (;;)
    {
        // If this test passes, it means that the most signif digits match,
        // and can be sent to the output stream.
        if ( (_high & 0x8000) == (_low & 0x8000) )
        {
            UInt16 bitToWrite = _high>>15;
            writeBit(bitToWrite);
            bitToWrite = ~bitToWrite;
            while ( _nUnderflowBits > 0 ) {
                writeBit(bitToWrite);
                _nUnderflowBits--;
            }
        }

        // If this test passes, the numbers are in danger of underflow, because
        // the most sigif digits don't match, and the 2nd digits are just one apart.
        //
        // _low =  01... and _high = 10...
        else if ( ( _low & 0x4000 ) && !( _high & 0x4000 ))
        {
            _nUnderflowBits += 1;
            _low  &= 0x3fff;
            _high |= 0x4000;
        }
        else
            break;

        // Shift all bits left.  Move 0 into _low and 1 into _high.
        _low  <<= 1;
        _high <<= 1;
        _high |=  1;
    }
}

void ArithmeticCodec::flushEncoder()
{
    // Write out some underflow bits and misc.
    writeBit( (_low & 0x4000)>>14 );

    _nUnderflowBits++;

    while ( _nUnderflowBits-- > 0 ) {
        writeBit( (~_low & 0x4000)>>14);
    }

    //Need 16 zeros at the end, for this decoding algorithm
    UInt32 zeroBit = 0x0000;
    for (int ii=0; ii<16; ii++) {
        writeBit(zeroBit);
    }

    //Flush out the local buffer
    if (_nBits > 0) {
        _pDriver->addCodeText( _bitBuffer<<(32-_nBits), 32 );
    }
}

void ArithmeticCodec::writeBit(UInt16 bit )
{
    _bitBuffer <<= 1;
    _bitBuffer  |= ((UInt32)bit & 0x1);
    _nBits++;

    if (_nBits==32) {
        _pDriver->addCodeText(_bitBuffer, 32);
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        _nBits = 0;
    }
}

Bool ArithmeticCodec::decode( CodecDriver* pDriver )
{
   ArithmeticProbabilityRange newSymbolRange;
   Int32 iCurrContext, nDummyTotalBits, cSymbolsCurrCtx, iCurrEntry;

   Int32 nSymbols = pDriver->numSymbolsToRead();

   ProbabilityContext* pCurrContext = NULL;
   CntxEntry* pCntxEntry = NULL;

   // Initialize decoding process
   Int32 nBitsRead = -1;
   pDriver->getNextCodeText(bitBuffer, nBitsRead);

   low  = 0;
   high = 0xffff;
   code = (bitBuffer >> 16);

   bitBuffer <<= 16;
   nBits = 16;

   // Begin decoding
   pDriver->getDecodeData(iCurrContext, nDummyTotalBits);
   for( Int32 ii = 0; ii < nSymbols; ii++ ) {
        pDriver->getContext(iCurrContext, pCurrContext);

        cSymbolsCurrCtx = pCurrContext->totalCount();
        UInt16 rescaledCode = ((((UInt32)(code - low) + 1) * (UInt32) cSymbolsCurrCtx -
1) / ((UInt32)(high - low) + 1));

        pCurrContext->lookupEntryByCumCount((Int32)rescaledCode,iCurrEntry);

        pCurrContext->getEntry(iCurrEntry, pCntxEntry);

        newSymbolRange.high_count = pCntxEntry->cCumCount + pCntxEntry.cCount;
        newSymbolRange.low_count  = pCntxEntry->cCumCount;
        newSymbolRange.scale      = cSymbolsCurrCtx;

        removeSymbolFromStream(newSymbolRange, pDriver);

        pDriver->addOutputSymbol(pCntxEntry);

        iCurrContext = pCntxEntry->iNextCntx;
   }

   return True;
}

Bool ArithmeticCodec::removeSymbolFromStream( ArithmeticProbabilityRange &sym,
                                              CodecDriver               *pDriver )
{
   // First, the range is expanded to account for the symbol removal.
   UInt32 range = UInt32(high - low)  + 1;
   high = low + (UInt32)((range * sym.high_count) / sym.scale - 1);
   low  = low + (UInt32)((range * sym.low_count ) / sym.scale);

   //Next, any possible bits are shipped out.
   for (;;) {
       // If the most signif digits match, the bits will be shifted out.
       if( (~(high^low)) & 0x8000 ) {
           // Do nothing
       } else if( (low & 0x4000) && !(high & 0x4000) ) {
          // Underflow is threatening, shift out 2nd most signif digit.
          code ^= 0x4000;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
            low  &= 0x3fff;
            high |= 0x4000;
        } else {
            // Nothing can be shifted out, so return.
            return True;
        }

        low  <<= 1;
        high <<= 1;
        high |=  1;
        code <<= 1;

        if( nBits == 0 ) {
            // The returned nBits here will always be 32
            pDriver->getNextCodeText(bitBuffer, nBits);
        }

        code |= (UInt16)(bitBuffer >> 31);
        bitBuffer <<= 1;
        nBits--;
    }
}
```

B.3.3 ArithmeticCodec2 class

```
class ArithmeticCodec2: public Codec2
{
  public:
    Bool encode( const Veci   &vValues,
                 Veci         &ovOOBValues,
                 Vecu         &ovCodeText,

                 Int32        &onBitsCodeText,

                 ProbContext2 *pProbCntx      );
    Bool decode( Int32         nValues,
                 const Veci   &vOOBValues,
                 const Vecu   &vCodeText,
                 Int32         nBitsCodeText,
                 Veci         &ovValues,
                 ProbContext2 *pProbCntx      );

  protected:
    Bool _encodeSymbol(UInt16 uLowCt, UInt16 uHighCt, UInt16 uScale );
    Bool _flushEncoder();
    Bool _removeSymbolFromStream( UInt16 uLowCt, UInt16 uHighCt, UInt16 uScale );
    Bool _flushDecoder();

    Bool getNextCodeText (UInt32 &uCodeText, Int32 &nBits);

    UInt16  _code;              // Present input code value, for decoding only
    UInt16  _low;               // Start of the current code range
    UInt16  _high;              // End of the current code range
    Int32   _nUnderflowBits;    // Number of underflow bits pending

    Vecu   *_pvCodeText;
    Int32  *_pcCodeTextLen;
    Int32   _iCurCodeText;

    UInt32  _uBitBuff;
    Int32   _nBitBuff;
};

// Reads a bit and places it into ouBit
#define ReadBit(ouBit)                          \
    if (_nBitBuff==0) {                         \
        getNextCodeText(_uBitBuff, _nBitBuff); \
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    }                                                   \
    ouBit = (_uBitBuff >> 31);                          \
    _uBitBuff <<= 1;                                    \
    _nBitBuff--;

// Reads a bit and ORs it into bit 0 of ouBit
#define ReadBit0(ouBit)                                 \
    if (_nBitBuff==0) {                                 \
        getNextCodeText(_uBitBuff, _nBitBuff);          \
    }                                                   \
    ouBit |= (_uBitBuff >> 31);                         \
    _uBitBuff <<= 1;                                    \
    _nBitBuff--;
// Writes bit 0 of uBit
#define WriteBit(uBit)                                  \
    if (_nBitBuff==32) {                                \
        addCodeText(_uBitBuff, 32);                     \
        _uBitBuff = _nBitBuff = 0;                      \
    }                                                   \
    _uBitBuff <<= 1;                                    \
    _uBitBuff  |= (UInt32(uBit) & 0x1);                 \
    _nBitBuff++;

Bool ArithmeticCodec2::encode(const Veci   &vValues,
                              Veci         &ovOOBValues,
                              Vecu         &ovCodeText,
                              Int32        &onBitsCodeText,
                              ProbContext2 *pProbCntx     )
{
    // Initialize output state
    ovOOBValues.setLength(0);
    ovCodeText.setLength(0);
    onBitsCodeText = 0;
    _pvCodeText = &ovCodeText;
    _pcCodeTextLen = &onBitsCodeText;

    // Initialize the encoder state
    _low  = 0x0000;
    _high = 0xffff;
    _nUnderflowBits = 0;

    // Prime the bit buffer
    _uBitBuff = 0;
    _nBitBuff = 0;

    const JtInt32 *paiValues = vValues.ptr();
    const JtCntxEntry2 *pEntry;
    Int32 nValues = vValues.length();
    Int32 cTotalCount = pProbCntx->totalCount();
    for (Int32 i = 0; i < nValues; i++ ) {
        // Look up the value in the prob context
        pProbCntx->lookupValue(paiValues[i], pEntry );

        // If this is not the null context, then we emit an escape symbol,
        // move the context it specifies, and restart the tranlation of
        // the same value.  Thus, a value may emit more than one symbol.
        if (pEntry->_iSym == CntxEntryBase2::CEBEscape) {
            ovOOBValues.append(paiValues[i]);
        }

        _encodeSymbol(pEntry->_cCumCount,
                      pEntry->_cCumCount + pEntry->_cCount,
                      cTotalCount));
    }

    _flushEncoder();

    return True;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
}

Bool ArithmeticCodec2::_encodeSymbol(UInt16 uLowCt, UInt16 uHighCt, UInt16 uScale )
{
    // These three lines rescale _high and _low for the new symbol.
    UInt32 uRange = UInt32(_high - _low) + 1;
    _high  = _low + (uRange * uHighCt) / uScale - 1;
    _low   = _low + (uRange * uLowCt ) / uScale;

    // This loop turns out new bits until _high and _low are far enough
    // apart to have stabilized.
    for (;;) {
        // If this test passes, it means that the most signif digits match,
        // and can be sent to the output stream.
        if ( (_high & 0x8000) == (_low & 0x8000) )
        {
            // Flush the bit buff if the MSB and underflow bits
            // won't all fit in what's left
            if (1+_nUnderflowBits > 32 - _nBitBuff) {
                addCodeText(_uBitBuff, _nBitBuff);
                _uBitBuff = 0;
                _nBitBuff = 0;
            }
            // Write the MSB and all uflow bits at once
            if (1+_nUnderflowBits <= 32 - _nBitBuff) {
                _uBitBuff <<= (1 + _nUnderflowBits);
                _uBitBuff |= (1 << _nUnderflowBits)
                          + (Int32(JtInt16(~_high)) >> 15);
                _nBitBuff += 1 + _nUnderflowBits;
                _nUnderflowBits = 0;
            }
            else {
                // We're writing more than 32 bits.
                _uBitBuff = (1 << 31)
                          + (Int32(JtInt16(~_high)) >> 15);
                addCodeText(_uBitBuff, 32);
                _nBitBuff = 0;
                _nUnderflowBits -= 31;
                // Emit the rest of the underflow bits
                _uBitBuff = (_uBitBuff << 1) | (_uBitBuff & 1);
                while (_nUnderflowBits >= 32) {
                    addCodeText(_uBitBuff, 32);
                    _nUnderflowBits -= 32;
                }
                addCodeText(_uBitBuff, _nUnderflowBits);
                _nUnderflowBits = 0;
                _uBitBuff = 0;
            }
        }

        // If this test passes, the numbers are in danger of underflow, because
        // the most sigif digits don't match, and the 2nd digits are just one apart.
        //
        // _low =  01... and _high = 10...
        else if ( ( _low & 0x4000 ) && !( _high & 0x4000 ))
        {
            _nUnderflowBits++;
            _low  &= 0x3fff;
            _high |= 0x4000;
        }
        else
            break;

        //Shift all bits left.  Move 0 into _low and 1 into _high.
        _low  <<= 1;
        _high <<= 1;
        _high |=  1;
    }
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    return True;
}

Bool ArithmeticCodec2::_flushEncoder()
{
    // Write out some underflow bits and misc.
    WriteBit((_low & 0x4000)>>14)
    _nUnderflowBits++;
    while (_nUnderflowBits-- > 0)
        WriteBit((~_low & 0x4000)>>14)

    //Need 16 zeros at the end, for this decoding algorithm
    UInt32 zeroBit = 0x0000;
    for (Int32 ii=0; ii<16; ii++) {
        WriteBit(zeroBit)
    }

    // Flush out the local buffer
    addCodeText(_uBitBuff, _nBitBuff);

    return True;
}

Bool ArithmeticCodec2::decode( Int32         nValues,
                               const Veci    &vOOBValues,
                               const Vecu    &vCodeText,
                               Int32         nBitsCodeText,
                               Veci          &ovValues,
                               ProbContext2 *pProbCntx     )
{
    ovValues.setLength(0);
    const Int32 *paiOOBValues = vOOBValues.ptr();
    ovValues.setLength(nValues);
    Int32 *paiValues = ovValues.ptr();
    Int32 cSymbolsCurrCtx = pProbCntx->totalCount();
    const CntxEntry2   *pCntxEntry = 0;

    // Initialize the arithmetic decoder state
    _iCurCodeText = 0;
    _pvCodeText = (Vecu*) &vCodeText;
    _pcCodeTextLen = &nBitsCodeText;
    getNextCodeText(_uBitBuff, _nBitBuff);
    _low  = 0;
    _high = 0xffff;
    _code = (_uBitBuff >> 16);
    _uBitBuff <<= 16;
    _nBitBuff -= 16;

    // Decode each symbol
    for (Int32 i = 0 ; i < nValues ; i++) {
        // Scale the current "code" into the range of counts presented by
        // the probcontext so we can look up the code.
        UInt16 rescaledCode = (((UInt32)(_code - _low) + 1) * (UInt32)cSymbolsCurrCtx -
1)
             / ((UInt32)(_high - _low) + 1);
        pProbCntx->lookupEntryByCumCount( (Int32)rescaledCode, pCntxEntry );

        // Emit the value corresponding to the symbol we just decoded
        if (pCntxEntry->_iSym != CntxEntryBase2::CEBEscape)
            *paiValues++ = pCntxEntry->_val;
        else
            *paiValues++ = *paiOOBValues++;

        // Set up the symbol's range and adjust the decoder state
        // to "remove" it.
        _removeSymbolFromStream( pCntxEntry->_cCumCount,
                                 pCntxEntry->_cCumCount + pCntxEntry->_cCount,
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
                                    cSymbolsCurrCtx );
    }

    _flushDecoder();

    return True;
}

Bool ArithmeticCodec2::_flushDecoder()
{
    UInt32 dummyBit;
    ReadBit(dummyBit)
    ReadBit(dummyBit)

    Assert( _uBitBuff == 0 );
    _nBitBuff = 0;

    return True;
}

Bool ArithmeticCodec2::_removeSymbolFromStream( UInt16 uLowCt, UInt16 uHighCt, UInt16
uScale )
{
    // First, the range is expanded to account for the symbol removal.
    UInt32 uRange = UInt32(_high - _low)  + 1;
    _high  = _low + (UInt32)((uRange * uHighCt) / uScale - 1);
    _low   = _low + (UInt32)((uRange * uLowCt ) / uScale);

    //Next, any possible bits are shipped out.
    for (;;) {
        // If the most signif digits match, the bits will be shifted out.
        if (UInt16(~(_high ^ _low)) >> 15){
        }
        // Else, if underflow is threatening, shift out the 2nd most signif digit.
        //else if ((_low & 0x4000) && !(_high & 0x4000))
        // If high=10xx and low=01xx
        else if (((_low >> 14) == 1) & ((_high >> 14) == 2)) {
            _code ^= 0x4000;
            _low  &= 0x3fff;
            _high |= 0x4000;
        }
        // Otherwise, nothing can be shifted out, so return.
        else {
            return True;
        }

        _low  <<= 1;
        _high <<= 1;
        _high |=  1;
        _code <<= 1;

        ReadBit0( _code )
    }
}
```

B.4 Deering Normal decoding classes

The following sub-sections contain a sample implementation of the decoding portion of the Deering Normal CODEC algorithm. A summary technical explanation of the Deering Normal CODEC can be found in 13.2.4 Deering Normal CODEC.

B.4.1 DeeringNormalLookupTable class

The DeeringNormalLookupTable class represents a lookup table used by the DeeringNormalCodec class for faster conversion from the compressed normal representation to the standard 3-float representation. The tables hold precomputed results of the trig functions called during conversion.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
class DeeringNormalLookupTable
{
public:
    DeeringNormalLookupTable();

    // Lookup and return the result of converting iTheta and iPsi to
    // real angles and taking the sine and cosine of both.  This gives
    // a slight speedup for normal decoding.
    Bool lookupThetaPsi(Int32 iTheta,
                        Int32 iPsi,
                        UInt32 numberBits,
                        Float32 outCosTheta,
                        Float32 outSinTheta,
                        Float32 outCosPsi,
                        Float32 outSinPsi );

    UInt32 numBitsPerAngle() {return nBits;}

private:
    UInt32 nBits;
    Vector vCosTheta;
    Vector vSinTheta;
    Vector vCosPsi;
    Vector vSinPsi;
};

DeeringNormalLookupTable::DeeringNormalLookupTable()
{
    UInt32 numberbits = 8;
    nBits = min(numberbits, (UInt32)31);

    Int32 tableSize = (1 << nBits);

    vCosTheta.setLength(tableSize+1);
    vSinTheta.setLength(tableSize+1);
    vCosPsi.setLength(tableSize+1);
    vSinPsi.setLength(tableSize+1);

    Float32 fPsiMax = 0.615479709;
    Float32 fTableSize = (Float32)tableSize;

    for( Int32 ii = 0; ii <= tableSize; ii++ )
    {
    Float32 fTheta =
        asin(tan(fPsiMax * Float32(tableSize - ii) / fTableSize));

        Float32 fPsi   = fPsiMax * (((Float32)ii) / fTableSize);
        vCosTheta[ii] = cos(fTheta);
        vSinTheta[ii] = sin(fTheta);
        vCosPsi[ii]   = cos(fPsi);
        vSinPsi[ii]   = sin(fPsi);
    }
}

Bool DeeringNormalLookupTable::lookupThetaPsi(Int32 iTheta,
                                              Int32 iPsi,
                                              UInt32 numberBits,
     Float32 outCosTheta,
     Float32 outSinTheta,
     Float32 outCosPsi,
     Float32 outSinPsi)
{
    Int32 offset = nBits - numberBits;

    outCosTheta = vCosTheta[iTheta << offset];
    outSinTheta = vSinTheta[iTheta << offset];
    outCosPsi   = vCosPsi[iPsi << offset];
    outSinPsi   = vSinPsi[iPsi << offset];
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    return True;
}
```

B.4.2 DeeringNormalCodec class

The DeeringNormalCodec class converts a normal vector to and from the standard 3-float representation and a lower-precision representation. The precision can be adjusted using the nbits parameter.

```
class DeeringNormalCodec
{
public:
    DeeringNormalCodec(Int32 numberbits = 6)
    {
        numBits = numberbits;
    }

    // Converts a compressed normal into a vector.
    Bool convertCodeToVec(UInt32 code, Vector& outVec);

    // Converts a compressed normal into a vector.
    Bool convertCodeToVec(UInt32 iSextant,
 UInt32 iOctant,
 UInt32 iTheta,
 UInt32 iPsi,
 Vector& outVec);

    // Separates an encoded normal into its 4 pieces
    Bool unpackCode(UInt32  code,
 UInt32& outSextant,
 UInt32& outOctant,
 UInt32& outTheta,
 UInt32& outPsi );

    private:
        Int32 numBits;
}

Bool DeeringNormalCodec::convertCodeToVec(UInt32 code, Vector& outVec)
{
    UInt32 s=0, o=0, t=0, p=0;
    unpackCode(code, s, o, t, p);

    convertCodeToVec(s, o, t, p, outVec);

    return True;
}

Bool DeeringNormalCode::convertCodeToVec(UInt32 iSextant,
                                         UInt32 iOctant,
                                         UInt32 iTheta,
                                         UInt32 iPsi,
                                         Vector& outVec)
{
    // Size of code = 6+2*numBits, and max code size is 32 bits,
    // so numBits shall be <= 13.

    // Code layout: [sextant:3][octant:3][theta:numBits][psi:numBits]

    outVec.setValues(0,0,0);
    Float32 fPsiMax = 0.615479709;

    UInt32  iBitRange = 1<<numBits;
    Float32 fBitRange = Float32(iBitRange);

    // For sextants 1, 3, and 5, iTheta needs to be incremented
    iTheta += (iSextant & 1);
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
Float32 fCosTheta, fSinTheta, fCosPsi, fSinPsi;

DeeringNormalLookupTable LookupTable;

if( (LookupTable.numBitsPerAngle() < (UInt32)numBits) ||
     !LookupTable.lookupThetaPsi(iTheta, iPsi, numBits,
                                 fCosTheta, fSinTheta,
                                 fCosPsi, fSinPsi) )
{
    Float32 fTheta = asin(tan(fPsiMax * Float32(iBitRange - iTheta) /
                              fBitRange));

    Float32 fPsi = fPsiMax * (iPsi / fBitRange);
    fCosTheta = cos(fTheta);
    fSinTheta = sin(fTheta);
    fCosPsi   = cos(fPsi);
    fSinPsi   = sin(fPsi);
}

Float32 x,y,z;
Float32 xx = x = fCosTheta * fCosPsi;
Float32 yy = y = fSinPsi;
Float32 zz = z = fSinTheta * fCosPsi;

//Change coordinates based on the sextant
switch( iSextant )
{
    case 0:     // No op
        break;

    case 1:     // Mirror about x=z plane
        z = xx;
        x = zz;
        break;

    case 2:     // Rotate CW
        z = xx;
        x = yy;
        y = zz;
        break;

    case 3:     // Mirror about x=y plane
        y = xx;
        x = yy;
        break;

    case 4:     // Rotate CCW
        y = xx;
        z = yy;
        x = zz;
        break;

    case 5:     // Mirror about y=z plane
        z = yy;
        y = zz;
        break;
};

//Change some more based on the octant

//if first bit is 0, negate x component
if( !(iOctant & 0x4) )
    x = -x;

//if second bit is 0, negate y component
if( !(iOctant & 0x2) )
    y = -y;
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
    //if third bit is 0, negate z component
    if( !(iOctant & 0x1) )
        z = -z;

    outVec.setValues(x,y,z);

    return True;
}

Bool DeeringNormalCodec::unpackCode(UInt32 code,
                                    UInt32& outSextant,
                                    UInt32& outOctant,
                                    UInt32& outTheta,
                                    UInt32& outPsi)
{
    UInt32 mask = (1<<numBits)-1;

    outSextant = (code >> (numBits+numBits+3)) & 0x7;
    outOctant  = (code >> (numBits+numBits))   & 0x7;
    outTheta   = (code >> (numBits))           & mask;
    outPsi     = (code)                        & mask;

    return True;
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Annex C
(informative)

Hashing – An Implementation

This Appendix provides a sample C++ implementation for the creation of hash values (as detailed in 13.2 Encoding Algorithms) used in the JT format.

```
unsigned int hash32( const unsigned int *pWords,
                     int nWords,
                     unsigned int uSeedHashValue )
{ return hash2(pWords, nWords, uSeedHashValue); }

unsigned int jthash16(const unsigned short *pBytes,
                      int nShort,
                      unsigned int uSeedHashValue)
{ return hash3(pBytes, nShort, uSeedHashValue); }

//--------------------------------------------------------------------
//  mix -- mix 3 32-bit values reversibly.
// For every delta with one or two bit set, and the deltas of all three
//   high bits or all three low bits, whether the original value of a,b,c
//   is almost all zero or is uniformly distributed,
// * If mix() is run forward or backward, at least 32 bits in a,b,c
//   have at least 1/4 probability of changing.
// * If mix() is run forward, every bit of c will change between 1/3 and
//   2/3 of the time.  (Well, 22/100 and 78/100 for some 2-bit deltas.)
// mix() was built out of 36 single-cycle latency instructions in a
//   structure that could supported 2x parallelism, like so:
//       a -= b;
//       a -= c; x = (c>>13);
//       b -= c; a ^= x;
//       b -= a; x = (a<<8);
//       c -= a; b ^= x;
//       c -= b; x = (b>>13);
//       ...
//   Unfortunately, superscalar Pentiums and Sparcs can't take advantage
//   of that parallelism.  They've also turned some of those single-cycle
//   latency instructions into multi-cycle latency instructions.  Still,
//   this is the fastest good hash I could find.  There were about 2^^68
//   to choose from.  I only looked at a billion or so.
--------------------------------------------------------------------

#define mix(a,b,c) \
{ \
  a -= b; a -= c; a ^= (c>>13); \
  b -= c; b -= a; b ^= (a<<8); \
  c -= a; c -= b; c ^= (b>>13); \
  a -= b; a -= c; a ^= (c>>12);  \
  b -= c; b -= a; b ^= (a<<16); \
  c -= a; c -= b; c ^= (b>>5); \
  a -= b; a -= c; a ^= (c>>3);  \
  b -= c; b -= a; b ^= (a<<10); \
  c -= a; c -= b; c ^= (b>>15); \
}

--------------------------------------------------------------------
// hash() -- hash a variable-length key into a 32-bit value
//   k     : the key (the unaligned variable-length array of bytes)
//   len   : the length of the key, counting by bytes
//   level : can be any 4-byte value
// Returns a 32-bit value.  Every bit of the key affects every bit of
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
// the return value.  Every 1-bit and 2-bit delta achieves avalanche.
// About 36+6len instructions.

// The best hash table sizes are powers of 2.  There is no need to do
// mod a prime (mod is sooo slow!).  If you need less than 32 bits,
// use a bitmask.  For example, if you need only 10 bits, do
//   h = (h & hashmask(10));
// In which case, the hash table should have hashsize(10) elements.
//
// If you are hashing n strings (JtUInt8 **)k, do it like this:
//   for (i=0, h=0; i<n; ++i) h = hash( k[i], len[i], h);
//
// By Bob Jenkins, 1996.  bob_jenkins@burtleburtle.net.  You may use this
// code any way you wish, private, educational, or commercial.  It's free.
//
// See http://burtleburtle.net/bob/                  // 2010/02/12
// See http://burtleburtle.net/bob/hash/doobs.html   // 2010/02/12
//
// Use for hash table lookup, or anything where one collision in 2^32 is
// acceptable.  Do NOT use for cryptographic purposes.
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// This works on all machines.  hash2() is identical to hash() on
// little-endian machines, except that the length has to be measured
// in ub4s instead of bytes.  It is much faster than hash().  It
// requires
// -- that the key be an array of UInt32's, and
// -- that all your machines have the same endianness, and
// -- that the length be the number of UInt32's in the key
// --------------------------------------------------------------------
unsigned int hash(const usigned char *k,        // key
                  unsigned int        length,   // length of the key
                  unsigned int        initval)  // prev hash, or an arbitrary value
{
   register unsigned int a,b,c,len;

   /* Set up the internal state */
   len = length;
   a = b = 0x9e3779b9;  /* the golden ratio; an arbitrary value */
   c = initval;           /* the previous hash value */
   /*---------------------------------------- handle most of the key */
   while (len >= 12) {
      a += (k[0] +((UInt32)k[1]<<8) +((UInt32)k[2]<<16) +((UInt32)k[3]<<24));
      b += (k[4] +((UInt32)k[5]<<8) +((UInt32)k[6]<<16) +((UInt32)k[7]<<24));
      c += (k[8] +((UInt32)k[9]<<8) +((UInt32)k[10]<<16)+((UInt32)k[11]<<24));
      mix(a,b,c);
      k += 12; len -= 12;
   }
   /*------------------------------------ handle the last 11 bytes */
   c += length;
   switch(len) {            /* all the case statements fall through */
     case 11: c+=((UInt32)k[10]<<24);
     case 10: c+=((UInt32)k[9]<<16);
     case 9 : c+=((UInt32)k[8]<<8);
      /* the first byte of c is reserved for the length */
     case 8 : b+=((UInt32)k[7]<<24);
     case 7 : b+=((UInt32)k[6]<<16);
     case 6 : b+=((UInt32)k[5]<<8);
     case 5 : b+=k[4];
     case 4 : a+=((UInt32)k[3]<<24);
     case 3 : a+=((UInt32)k[2]<<16);
     case 2 : a+=((UInt32)k[1]<<8);
     case 1 : a+=k[0];
     /* case 0: nothing left to add */
   }
   mix(a,b,c);
   /*-------------------------------------------- report the result */
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
   return c;
}

unsigned int hash3(const unsigned short *k,       /* the key */
                   unsigned int          length,  /* the length of the key */
                   unsigned int          initval) /* the previous hash, or an arbitrary
value */
{
   unsigned int a,b,c,len;

   /* Set up the internal state */
   len = length;
   a = b = 0x9e3779b9;    /* the golden ratio; an arbitrary value */
   c = initval;           /* the previous hash value */

   /*---------------------------------------- handle most of the key */
   while (len >= 6)
   {
      a += (k[0] + (UInt32(k[1]) << 16));
      b += (k[2] + (UInt32(k[3]) << 16));
      c += (k[4] + (UInt32(k[5]) << 16));
      mix(a,b,c);
      k += 6; len -= 6;
   }

   /*------------------------------------- handle the last 2 uint32s */
   c += length;
   switch(len)              /* all the case statements fall through */
   {
       case 5 : c+=(UInt32(k[4]) << 16);
       /* c is reserved for the length */
       case 4 : b+=(UInt32(k[3]) << 16);
       case 3 : b+=k[2];
       case 2 : a+=(UInt32(k[1]) << 16);
       case 1 : a+=k[0];
       /* case 0: nothing left to add */
   }
   mix(a,b,c);
   /*-------------------------------------------- report the result */
   return c;
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

Annex D
(informative)
Polygon Mesh Topology Coder

The topology coding algorithm described here is used to code the *dual* of the desired mesh. Thus, for example, the reader will need to take the dual of the decoded mesh in order to obtain the original primal mesh. Presented below are classes suitable for representing the dual of a polygon mesh and the dual topology decoding algorithm.

At a high level, the topology coder works by traversing the dual mesh to be encoded one vertex and one face at a time. The coder maintains a queue of faces to be processed; the initial queue is created using the valence of an arbitrary vertex of the mesh followed by the degrees of the faces adjacent to that vertex, and adds the adjacent faces to the face queue. Each time it visits a face, it encodes the *degree* of that face and emits any incident vertices that have not yet been visited. Each time the coder visits a vertex, it encodes the *valence* of the vertex (usually 3 in the current case), and emits any incident faces that have not yet been visited. It works its way through the mesh in this fashion until all vertices and faces have been encoded. Thus, the primary output from the topology coder is a list of vertex valences and face degrees. These two fields plus two more encoding so-called *split faces*, coupled with the exact coder implementation completely encode the mesh topology in a very compact manner[1].

In addition to these two basic fields are added a number of other fields that organize the dual vertices into *vertex groups*, and also encode the *vertex attributes* (e.g. normals, colours, and texture coordinates) around each dual face's *degree ring*.

The topological coder can only encode *closed, manifold* meshes. It cannot encode *boundaries*; it can only encode edges with exactly two incident faces. But, as we know, real-world data is chock full of meshes with boundaries. In order to encode these types of meshes, it is necessary to add *cover faces* incident to all boundary loops whose sole job is to turn the mesh into a *closed* mesh. It is the dual of this closed, manifold mesh that is actually encoded. Thus, most meshes encoded in JT files contain a few cover faces. These faces may be of arbitrarily high degree, and they represent the only exceptions to the general rule that the numbers in the dual vertex valence array are usually three. It is necessary to flag all such artificially introduced cover faces so that they can be removed by the loader. These flags are encoded below in the Face Flags array. Primal faces are flagged with zero, while cover faces are flagged with one.

Now, let us make the connection between topological vertices and how vertex attributes relate to them. Several faces may be incident on the same topological mesh vertex. While this topological vertex has only a single 3D coordinate, it may have a different set of *vertex attributes* for each incident face. Vertex attributes include colour, normal, and texture coordinates. An important observation in real-world data is that adjacent faces tend to share the same vertex attributes. Thus, a natural way to encode which vertex attributes map to which faces within a given valence ring (the counter-clockwise ordered set of faces incident on a given vertex) is by way of a bit vector. The bit vector begins at the first face the coder encounters that is incident to the vertex, and proceeds counter clockwise around the vertex, allocating one bit per incident face. A value of 0 is assigned to the bit if all vertex attributes for the face are the same as the face immediately clockwise. A value of 1 is assigned if the vertex attributes for the face are different. Recall that these bits from the original primal mesh are encoded as face attributes in the dual mesh.

Thus, at the end of the coding process, there will be one such bit vector per topological vertex in the mesh. These bit vectors will be of disparate lengths because all vertex valences are not the same. Though there is no theoretical limit to the valence of any given vertex, in practice, the vertex valences seldom rise above six, and only rarely rise into the dozens. As a matter of practicality, then, we break this list of bit vectors into those of length 64 and smaller into one group, and all others into a list of so-called “high-valence” bit vectors. The

[1] Similar methods of topology coding are described in [7] and US patent # 7,098,916. The topology coding algorithm described herein differs from such methods in that while they utilize a queue of active *vertices*, the instant algorithm utilizes a queue of active *faces*. Other differences include the tracking of face group numbers and per-vertex attributes such as normals, colours, and texture coordinates.

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

low-valence bit vectors are encoded into three fields of 30, 30, and 4 bits respectively. The high-valence bit vectors are adjoined end-to-end into a single long bit vector, and encoded as a single array of integers. As an additional optimization, the low-valence bit vectors are grouped into 8 “context groups” depending on the valence of the vertex being coded. This is done in order to improve compression performance because the valence bit vectors in each of the most common groups typically share similar statistics. Context group number 8 is the only one that encodes valence rings up to valence 64. Again, recall that these attribute bits from the original primal mesh are encoded as face attribute bits in the dual mesh.

D.1 DualVFMesh

The DualVFMesh (Dual Vertex-Facet Mesh) is a support class paired with the topology decoder itself, and represents a closed two-manifold polygon mesh. The topology decoder reconstructs the encoded dual mesh into a DualVFMesh, building it one vertex and one facet at a time. When the decoder is finished, it will have visited each vertex and each face of the dual mesh exactly once. DualVFMesh is not intended as a work horse in-memory storage container because its way of encoding the topological connections between faces and vertices is memory-intensive.

```
class DualVFMesh
{
  public:
    // ========== Housekeeping Interface ==========
    DualVFMesh();
    DualVFMesh (const DualVFMesh &rhs);
    DualVFMesh &operator=(const DualVFMesh &rhs);

    // ========== Topology Interface ==========

    // Vtx creation
    bool          isValidVtx (Int32  iVtx) const;
    bool          newVtx     (Int32  iVtx,
                              Int32  iValence,
                              UInt16 uFlags = 0);
    bool          setVtxFlags(Int32  iVtx,
                              UInt16 uFlags);
    bool          setVtxGrp  (Int32  iVtx,
                              Int32  iVGrp);
    UInt16        vtxFlags   (Int32  iVtx) const;
    Int32         vtxGrp     (Int32  iVtx) const;

    // Face creation
    bool          isValidFace  (Int32  iFace) const;
    bool          newFace      (Int32  iFace,
                                Int32  cDegree,
                                Int32  cFaceAttrs = 0,
                                UInt64 uFaceAttrMask = 0,
                                UInt16 uFlags = 0);
    bool          newFace      (Int32  iFace,
                                Int32  cDegree,
                                Int32  cFaceAttrs,
                                const BitVec *pvbFaceAttrMask,
                                UInt16 uFlags);
    bool          setFaceFlags (Int32  iFace,
                                UInt16 uFlags);
    UInt16        faceFlags    (Int32  iVtx) const;
    bool          setFaceAttr  (Int32  iFace,
                                Int32  iAttrSlot,
                                Int32  iFaceAttr);
    Int32         faceAttr     (Int32  iFace,
                                Int32  iAttrSlot) const;

    // Topology connection
    bool          setVtxFace(Int32  iVtx,
                             Int32  iFaceSlot,
                             Int32  iFace);
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
  bool          setFaceVtx(Int32  iFace,
                           Int32  iVtxSlot,
                           Int32  iVtx);

  // Queries
  Int32         valence    (Int32  iVtx) const
      { return _vVtxEnts[iVtx].cVal; }
  Int32         degree   (Int32  iFace ) const
      { return _vFaceEnts[iFace].cDeg; }
  Int32         face     (Int32  iVtx,
                          Int32  iFaceSlot) const
      { return _viVtxFaceIndices[(_vVtxEnts[iVtx]).iVFI + iFaceSlot]; }
  Int32         vtx      (Int32  iFace,
                          Int32  iVtxSlot) const
      { return _viFaceVtxIndices[_vFaceEnts[iFace].iFVI + iVtxSlot]; }
  Int32         numVts       () const
      { return _vVtxEnts.length(); }
  Int32         numFaces     () const
      { return _vFaceEnts.length(); }
  Int32         numAttrs     () const
      { return _viFaceAttrIndices.length(); }
  Int32         numAttrs       (Int32 iFace) const
      { return _vFaceEnts[iFace].cFaceAttrs; }
  UInt64        attrMask       (Int32 iFace) const
      { return _vFaceEnts[iFace].u.uAttrMask; }
  const BitVec *attrMaskV      (Int32 iFace) const
      { return _vFaceEnts[iFace].u.pvbAttrMask; }
  Int32         findVtxSlot   (Int32 iFace,
                               Int32 iTargVtx) const;
  Int32         findFaceSlot    (Int32 iVtx,
                                 Int32 iTargFace) const;
  Int32         emptyFaceSlots  (Int32 iFace) const
      { return _vFaceEnts[iFace].cEmptyDeg; }

  // ========== VFMesh Data Members ==========
public:
  class VtxEnt {
    public:
      VtxEnt() : cVal(0), uFlags(0), iVGrp(-1), iVFI(-1) {}
      UInt16    cVal;   // Vtx valence
      UInt16    uFlags; // User flags
      Int32     iVGrp;  // Vtx group
      Int32     iVFI;   // Idx into _viVtxFaceIndices of cVal incident faces
  };

  // Number of optimized mask bits.
  static const Int32 cMBits = 64;

  class FaceEnt {
    public:
      FaceEnt() : cDeg(0), uFlags(0), cEmptyDeg(0),
                  cFaceAttrs(0), iFVI(-1), iFAI(-1) { u.uAttrMask = 0; }
      FaceEnt(const FaceEnt &rhs) : cDeg(rhs.cDeg), cEmptyDeg(rhs.cEmptyDeg),
                                    cFaceAttrs(rhs.cFaceAttrs), iFVI(rhs.iFVI),
                                    iFAI(rhs.iFAI)
      {
          if (cDeg <= cMBits)
              u.uAttrMask = rhs.u.uAttrMask;
          else
              JtWrapNew(u.pvbAttrMask, new BitVec(*rhs.u.pvbAttrMask));
      }
      ~FaceEnt() { if (cDeg > cMBits && u.pvbAttrMask) delete u.pvbAttrMask; }
      UInt16    cDeg;       // Face degree
      UInt16    cEmptyDeg;  // Empty degrees (opt for emptyFaceSlots())
      UInt16    cFaceAttrs;  // Number of face attributes
      UInt16    uFlags;     // User flags
      union {
          UInt64  uAttrMask;    // Degree-ring attr mask as a UInt64
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
            BitVec *pvbAttrMask;  // Degree-ring attr mask as a BitVec
        } u;
        Int32     iFVI; // Idx into _viFaceVtxIndices of cDeg incident vts
        Int32     iFAI; // Idx into _viFaceAttrIndices of cAttr attributes
    };

  protected:
    // Subscripted by atom number, the entry contains the vtx valence and
    // points to the location in _viVtxFaceIndices of valence consecutive
    // integers that in turn contain the indices of the incident faces
    // in _vFaceRecs to the vtx.
    JtVec<VtxEnt>  _vVtxEnts;

    // Subscripted by unique vertex record number, the entry contains the
    // face degree and points to the location in _viFaceVtxIndices of
    // cDeg consecutive integers that in turn contain the indices of the
    // vertices indicent upon the face, in CCW order, in _vVtxRecs.
    JtVec<FaceEnt>   _vFaceEnts;

    // Combined storage for all vtxs.
    JtVeci          _viVtxFaceIndices;

    // Combined storage for all faces.
    JtVeci          _viFaceVtxIndices;

    // Combined storage for all face attribute record identifiers
    JtVeci          _viFaceAttrIndices;
};

bool
DualVFMesh::isValidVtx(Int32 iVtx) const
{
    bool bRet = JtFalse;
    if (iVtx >= 0 && iVtx < _vVtxEnts.length()) {
        const VtxEnt &rFE = _vVtxEnts[iVtx];
        bRet = (rFE.cVal != 0);
    }
    return bRet;
}

bool
DualVFMesh::newVtx(Int32 iVtx,
                   Int32 iValence,
                   UInt16 uFlags)
{
    VtxEnt &rFE = _vVtxEnts[iVtx];
    if (rFE.cVal != iValence) {
        rFE.cVal   = iValence;
        rFE.uFlags = uFlags;
        rFE.iVFI   = _viVtxFaceIndices.length();
        _viVtxFaceIndices.verify(rFE.iVFI + iValence - 1);
        for (Int32 i = rFE.iVFI ; i < rFE.iVFI + iValence ; i++)
            _viVtxFaceIndices[i] = -1;
    }
    return true;
}

bool
DualVFMesh::setVtxGrp(Int32  iVtx,
                      Int32  iVGrp)
{
    VtxEnt &rFE = _vVtxEnts[iVtx];
    rFE.iVGrp = iVGrp;
    return true;
}

bool
DualVFMesh::setVtxFlags(Int32  iVtx,
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
                        UInt16 uFlags)
{
    VtxEnt &rFE = _vVtxEnts[iVtx];
    rFE.uFlags = uFlags;
    return true;
}

Int32
DualVFMesh::vtxGrp   (Int32  iVtx) const
{
    Int32 u = -1;
    if (iVtx >= 0 && iVtx < _vVtxEnts.length()) {
        const VtxEnt &rFE = _vVtxEnts[iVtx];
        u = rFE.iVGrp;
    }
    return u;
}

UInt16
DualVFMesh::vtxFlags   (Int32  iVtx) const
{
    UInt16 u = 0;
    if (iVtx >= 0 && iVtx < _vVtxEnts.length()) {
        const VtxEnt &rFE = _vVtxEnts[iVtx];
        u = rFE.uFlags;
    }
    return u;
}


bool
DualVFMesh::isValidFace(Int32 iFace) const
{
    bool bRet = JtFalse;
    if (iFace >= 0 && iFace < _vFaceEnts.length()) {
        const FaceEnt &rVE = _vFaceEnts[iFace];
        bRet = (rVE.cDeg != 0);
    }
    return bRet;
}

bool
DualVFMesh::newFace(Int32  iFace,
                    Int32  cDegree,
                    Int32  cFaceAttrs,
                    UInt64 uFaceAttrMask,
                    UInt16 uFlags)
{
    FaceEnt &rVE = _vFaceEnts[iFace];
    if (rVE.cDeg != cDegree) {
        rVE.cDeg         = cDegree;
        rVE.cEmptyDeg    = cDegree;
        rVE.cFaceAttrs    = cFaceAttrs;
        rVE.uFlags       = uFlags;
        rVE.u.uAttrMask = uFaceAttrMask;
        rVE.iFVI         = _viFaceVtxIndices.length();
        rVE.iFAI         = _viFaceAttrIndices.length();
        _viFaceVtxIndices.verify(rVE.iFVI + cDegree  - 1);
        if (cFaceAttrs > 0)
            _viFaceAttrIndices.verify(rVE.iFAI + cFaceAttrs - 1);
        for (Int32 i = rVE.iFVI ; i < rVE.iFVI + cDegree ; i++)
            _viFaceVtxIndices[i] = -1;
        for (Int32 i = rVE.iFAI ; i < rVE.iFAI + cFaceAttrs ; i++)
            _viFaceAttrIndices[i] = -1;
    }
    return true;
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
bool
DualVFMesh::newFace(Int32  iFace,
                    Int32  cDegree,
                    Int32  cFaceAttrs,
                    const BitVec *pvbFaceAttrMask,
                    UInt16 uFlags)
{
    FaceEnt &rVE = _vFaceEnts[iFace];
    if (rVE.cDeg != cDegree) {
        rVE.cDeg        = cDegree;
        rVE.cEmptyDeg   = cDegree;
        rVE.cFaceAttrs   = cFaceAttrs;
        rVE.uFlags      = uFlags;
        rVE.u.pvbAttrMask = new BitVec(*pvbFaceAttrMask);
        rVE.iFVI        = _viFaceVtxIndices.length();
        rVE.iFAI        = _viFaceAttrIndices.length();
        _viFaceVtxIndices.verify(rVE.iFVI + cDegree  - 1);
        if (cFaceAttrs > 0)
            _viFaceAttrIndices.verify(rVE.iFAI + cFaceAttrs - 1);
        for (Int32 i = rVE.iFVI ; i < rVE.iFVI + cDegree ; i++)
            _viFaceVtxIndices[i] = -1;
        for (Int32 i = rVE.iFAI ; i < rVE.iFAI + cFaceAttrs ; i++)
            _viFaceAttrIndices[i] = -1;
    }
    return true;
}

bool
DualVFMesh::setFaceFlags(Int32  iFace,
                         UInt16 uFlags)
{
    FaceEnt &rVE = _vFaceEnts[iFace];
    rVE.uFlags = uFlags;
    return true;
}

UInt16
DualVFMesh::faceFlags   (Int32  iFace) const
{
    UInt16 u = 0;
    if (iFace >= 0 && iFace < _vFaceEnts.length()) {
        const FaceEnt &rVE = _vFaceEnts[iFace];
        u = rVE.uFlags;
    }
    return u;
}

bool
DualVFMesh::setFaceAttr(Int32  iFace,
                        Int32  iAttrSlot,
                        Int32  iFaceAttr)
{
    FaceEnt &rVE = _vFaceEnts[iFace];
    Int32 *paiFAI = _viFaceAttrIndices.ptr();
    paiFAI[rVE.iFAI + iAttrSlot] = iFaceAttr;
    return true;
}

Int32
DualVFMesh::faceAttr(Int32  iFace,
                     Int32  iAttrSlot) const
{
    Int32 u = 0;
    if (iFace >= 0 && iFace < _vFaceEnts.length()) {
        const FaceEnt &rVE = _vFaceEnts[iFace];
        if (iAttrSlot >= 0 && iAttrSlot < rVE.cDeg) {
            const Int32 *paiFAI = _viFaceAttrIndices.ptr();
            u = paiFAI[rVE.iFAI + iAttrSlot];
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved

```
        }
    }
    return u;
}

// Attaches VF face iFace to VF vertex iVtx in the vertex's
// face slot iFaceSlot
bool
DualVFMesh::setVtxFace(Int32 iVtx,
                       Int32 iFaceSlot,
                       Int32 iFace)
{
    VtxEnt &rFE = _vVtxEnts[iVtx];
    _viVtxFaceIndices[rFE.iVFI + iFaceSlot] = iFace;
    return true;
}

// Attaches VF vertex iVtx to VF face iFace in the face's
// vertex slot iVtxSlot
bool
DualVFMesh::setFaceVtx(Int32 iFace,
                       Int32 iVtxSlot,
                       Int32 iVtx)
{
    FaceEnt &rVE = _vFaceEnts[iFace];
    Int32 *paiFVI = _viFaceVtxIndices.ptr();
    rVE.cEmptyDeg -= (paiFVI[rVE.iFVI + iVtxSlot] != iVtx);
    paiFVI[rVE.iFVI + iVtxSlot] = iVtx;
    return true;
}

// Searches the list of incident vts to face iFace for
// iTargVtx and returns the vtx slot at which it is found
// or -1 if iTargVtx is not found.
Int32
DualVFMesh::findVtxSlot(Int32 iFace,
                        Int32 iTargVtx) const
{
    const FaceEnt &rVE = _vFaceEnts[iFace];
    const Int32 *const pFaceVtxIndices = _viFaceVtxIndices.ptr() + rVE.iFVI;
    Int32 cDeg = rVE.cDeg;
    Int32 iSlot = -1;
    for (Int32 iVtxSlot = 0 ; iVtxSlot < cDeg ; iVtxSlot++) {
        if (pFaceVtxIndices[iVtxSlot] == iTargVtx) {
            iSlot = iVtxSlot;
            break;
        }
    }
    return iSlot;
}

// Searches the list of incident faces to vertex iVtx for
// iTargFace and returns the face slot at which it is found
// or -1 if iTargFace is not found.
Int32
DualVFMesh::findFaceSlot (Int32 iVtx,
                          Int32 iTargFace) const
{
    const VtxEnt &rFE = _vVtxEnts[iVtx];
    const Int32 *const pVtxFaceIndices = _viVtxFaceIndices.ptr() + rFE.iVFI;
    for (Int32 iFaceSlot = 0 ; iFaceSlot < rFE.cVal ; iFaceSlot++) {
        if (pVtxFaceIndices[iFaceSlot] == iTargFace) {
            return iFaceSlot;
        }
    }
    return -1;
}
```

STANDARDSISO.COM : Click to view the full PDF of ISO 14306:2012

© ISO 2012 – All rights reserved