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.

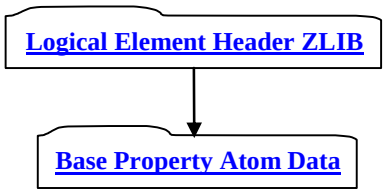
application or enterprise properties/meta-data on Nodes in JT files can be found in 9.6 Metadata Conventions section of this reference.

7.2.1.2.1 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.

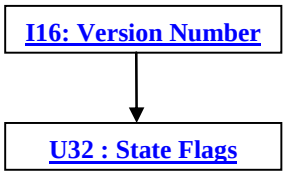
Figure 70: Base Property Atom Element data collection



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

7.2.1.2.1.1 Base Property Atom Data

Figure 71: Base Property Atom Data collection



I16: Version Number

valid value for Base Property Atom Data.

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.

7.2.1.2.2 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 72: String Property Atom Element data collection

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

Complete description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

I16: Version Number

y valid value for String Property Atom Element.

MbString : Value

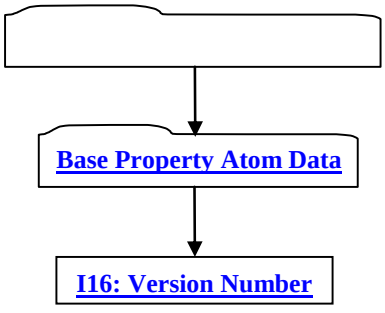
Value contains the character string value for this property atom.

7.2.1.2.3 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.

Figure 73: Integer Property Atom Element data collection



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

Complete description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

I16: Version Number

for Integer Property Atom Element.

I32 : Value

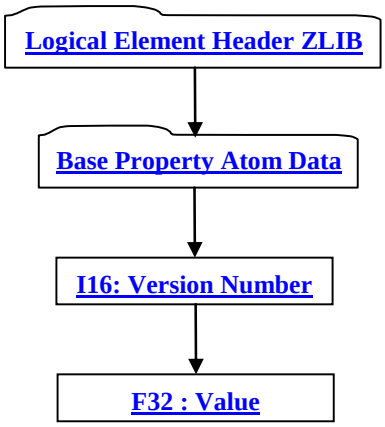
Value contains the integer value for this property atom.

7.2.1.2.4 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 74: Floating Point Property Atom Element data collection



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

Complete description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

I16: Version Number

ntly the only valid value.

F32 : Value

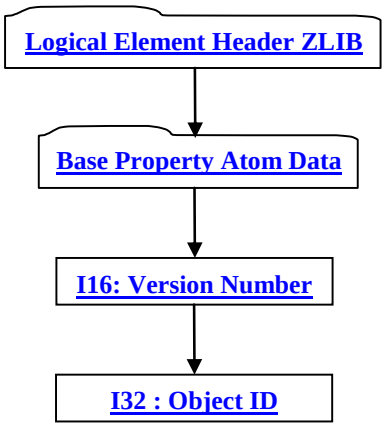
Value contains the floating point value for this property atom.

7.2.1.2.5 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 75: JT Object Reference Property Atom Element data collection



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

Complete description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

I16: Version Number

.

I32 : Object ID

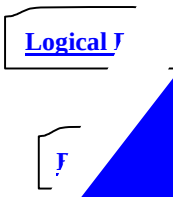
Object ID specifies the identifier within the JT file for the referenced object.

7.2.1.2.6 Date Property Atom Element

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

Date Property Atom Element

Figure 76: Date perty



escription for Logical Element Header ZLIB can be found in 7.1.3.2.3 Logical Element Header ZLIB.

description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

Version Number

Late Loaded Property Atom Element.

I6 : 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.

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.

7.2.1.2.7 Late Loaded Property Atom Element

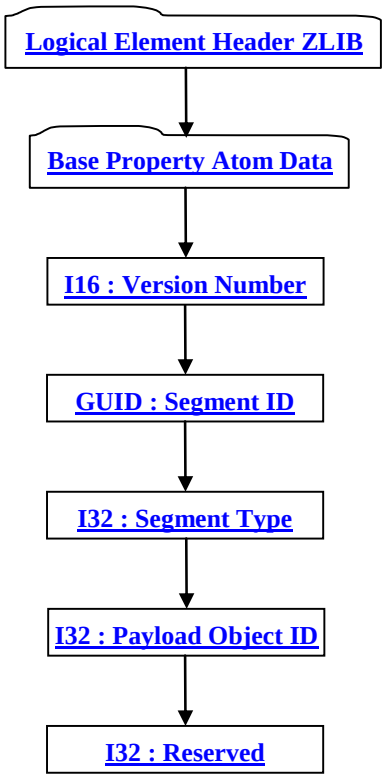
Object Type ID: 0xe0b05be5, 0xfbbd, 0x11d1, 0xa3, 0xa7, 0x00, 0xaa, 0x00, 0xd1, 0x09, 0x54

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.

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 7.2.2 Shape LOD Element and 7.2.3 JT B-Rep Segment).

Figure 77: Late Loaded Property Atom Element data collection



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

Complete description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

I16 : Version Number

for Late Loaded Property Atom Element.

GUID : Segment ID

Segment ID is the globally unique identifier for the associated data segment in the JT file. See 7.1.2 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 in Table 3: Segment Types.

I32 : Segment Type

denotes that the segment co 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

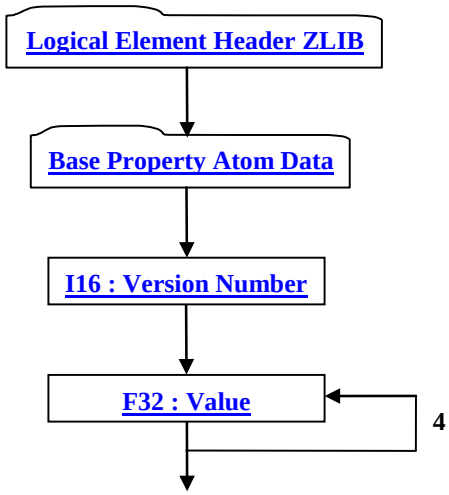
Reserved data field that is guaranteed to always be greater than or equal to 1

7.2.1.2.8 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 78: Vector4f Property Atom Element data collection



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

Complete description for Base Property Atom Data can be found in 7.2.1.2.1.1Base Property Atom Data.

I16 : Version Number

ently the only valid value for Late Loaded Property Atom Element.

F32 : Value

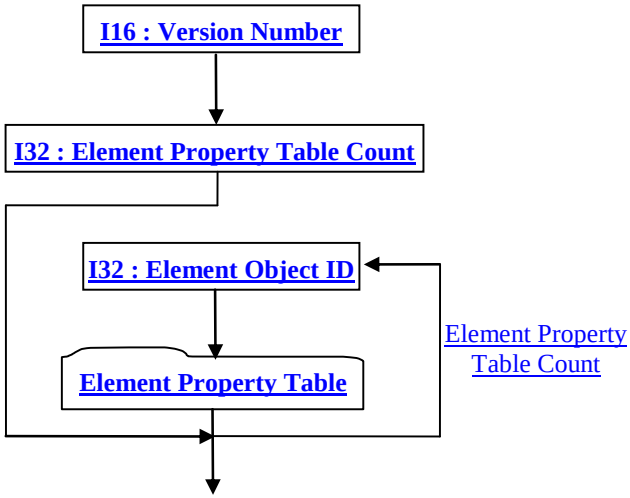
Value contains the floating point value for this property atom

7.2.1.3 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 79: Property Table data collection



I16 : Version Number

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 7.2.1.1.1Node Elements) and Attribute Elements (see 7.2.1.1.2 Attribute Elements) that have associated Property Atom Elements (see 7.2.1.2 Property Atom Elements).

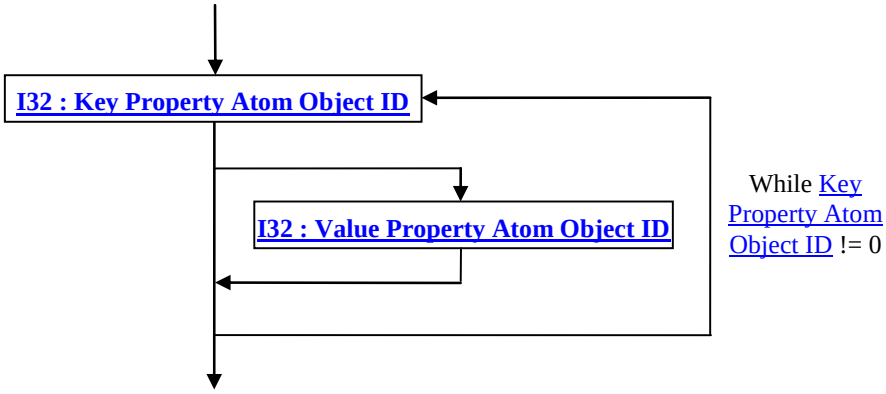
I32 : Element Object ID

Element Object ID is the identifier for the Node Element object (see 7.2.1.1.1Node Elements) or the Attribute Element object (see 7.2.1.1.2Attribute 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).

7.2.1.3.1 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 O Key Property Atom Object ID.

Figure 80: 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 7.2.1.2 Property Atom Elements)

I32 : Value Property Atom Object ID

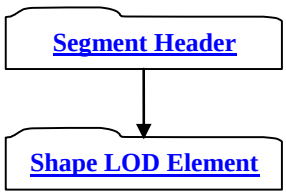
Value Property Atom Object ID is the identifier for the Property Atom Element object (see 7.2.1.2 Property Atom Elements) Key Property Atom Object ID has a

v

7.2.2 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 7.2.1.1.1.10 Shape Node Elements and 0 Late Loaded Property Atom Element respectively).

Figure 81: Shape LOD Segment data collection



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

7.2.2.1 Shape LOD Element

A Shape LOD Element is the holder/container of the geometric shape definition data (e.g. vertices, polygons, normals, etc.) ontained 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.

7.2.2.1.1 Base Shape LOD Element

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

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

Figure 82: Base Shape LOD Element data collection

Complete description for Logical Element Header can be found in 7.1.3.2.1Logical Element Header.

7.2.2.1.1.1Base Shape LOD Data

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

Figure 83: Base Shape LOD Data collection

I16 : Version Number

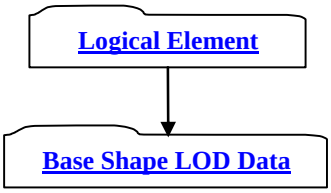
is currently the only valid value.

7.2.2.1.2 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 84: Vertex Shape LOD Element data collection



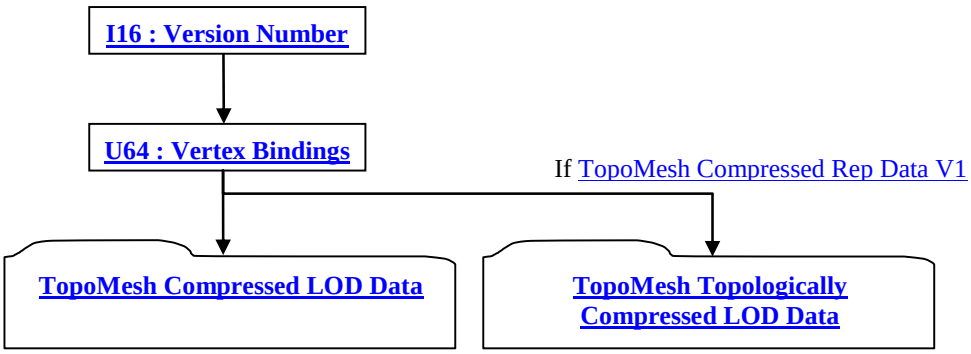
Complete description for Logical Element Header can be found in 7.1.3.2.1Logical Element Header.

7.2.2.1.2.1 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.

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 9.10 Brep Face Group Associations.

Figure 85: Vertex Shape LOD Data collection



Complete description for TopoMesh Compressed LOD Data and TopoMesh Topologically Compressed LOD Data can be found in 7.2.2.1.2.3 TopoMesh Compressed LOD Data and 7.2.2.1.2.4 TopoMesh Topologically Compressed LOD Data.

I16 : Version Number

Version
valid value.

U64 : Vertex Bindings

Binding Attributes is a collection of normal, texture coordinate, and color binding information encoded within a single U64 using the following bit allocation. All undocumented bits are reserved.

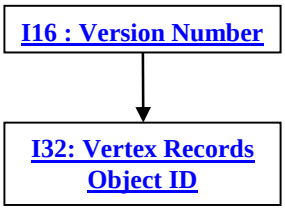
Bits 1-3	Vertex Coordinate Binding. The Vertex Coordinate Binding denotes per vertex coordinate field data is present when one of the bits is set. Bit 1 - 2 Component Vertex Coordinates Bit 2 - 3 Component Vertex Coordinates Bit 3 - 4 Component Vertex Coordinates
Bit 4	Normal Binding. The Normal Binding 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	Color Binding. The Color Binding denotes per vertex color field data is present when one of the bits is set. Bit 5 - 3 Component Colors Bit 6 - 4 Component Color
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
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.

7.2.2.1.2.2 TopoMesh LOD Data

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

Figure 86: TopoMesh LOD Data collection



I16 : Version Number

are

currently the only valid values.

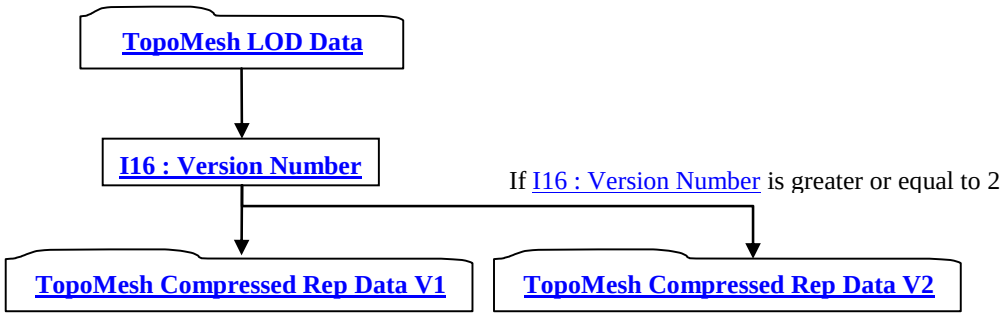
I32: Vertex Records Object ID

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.

7.2.2.1.2.3 TopoMesh Compressed LOD Data

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

Figure 87: 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 7.2.2.1.2.2 TopoMesh LOD Data, 7.2.2.1.2.7 TopoMesh Compressed Rep Data V1, and 7.2.2.1.2.8 TopoMesh Compressed Rep Data V2.

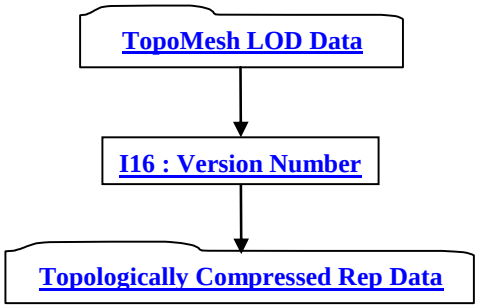
I16 : Version Number

Version Number is the version i
currently the only valid values.

7.2.2.1.2.4 TopoMesh Topologically Compressed LOD Data

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

Figure 88: TopoMesh Topologically Compressed LOD Data collection



Complete description for TopoMesh LOD Data and Topologically Compressed Rep Data can be found in 7.2.2.1.2.2 TopoMesh LOD Data and 7.2.2.1.2.5 Topologically Compressed Rep Data.

I16 : Version Number

Version Number is the version identifier for this TopoMesh Topologically Compressed LOD Data. Version number currently the only valid values.

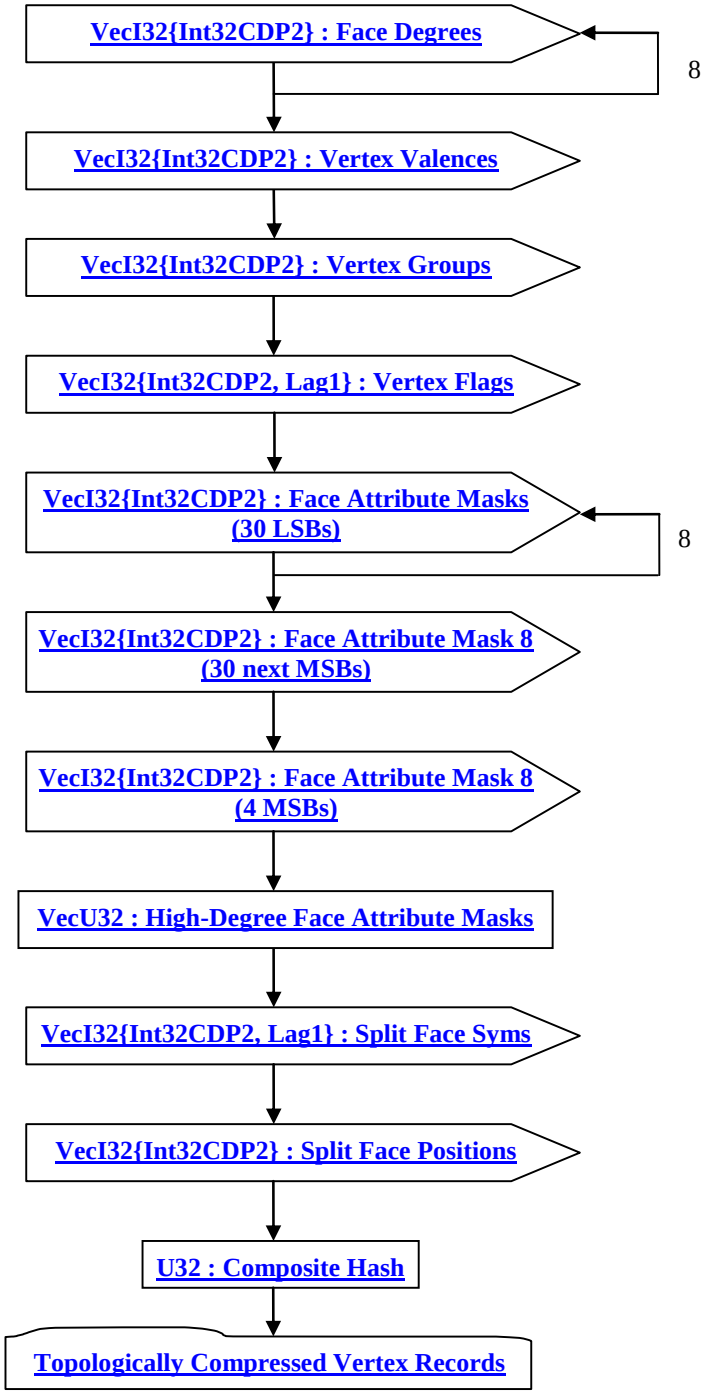
7.2.2.1.2.5 Topologically Compressed Rep Data

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 89. 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 Appendix E: Polygon Mesh Topology Coder on this data to reconstruct the topologically connected representation of the triangle mesh in a so-called " -weight data structure can then be exported to a lighter-weight form, and the dual VFMesh discarded if desired.

Figure 89: Topologically Compressed Rep Data Collection



VecI32{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.

Vecl32{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.

Vecl32{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.

Vecl32{Int32CDP2} : Face Attribute Masks (30 LSBs)

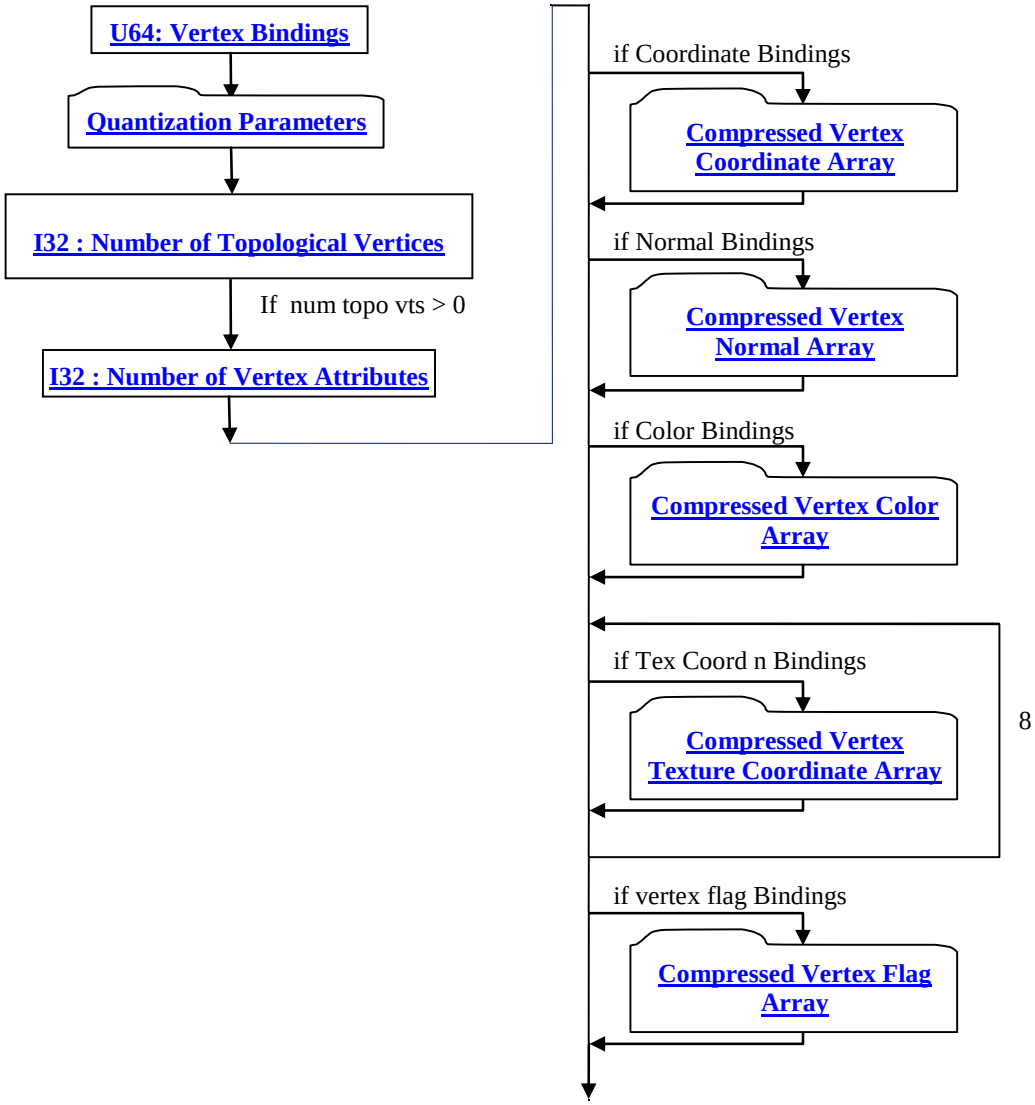
This

```
  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 );
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 );
```

7.2.2.1.2.6 Topologically Compressed Vertex Records

Documented here is the format of the vertex data written by the topological encoder from Appendix E: . Some additional explanation is necessary, however, becaus

Figure 90: Topologically Compressed Vertex Records data collection



U64: Vertex Bindings

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

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.

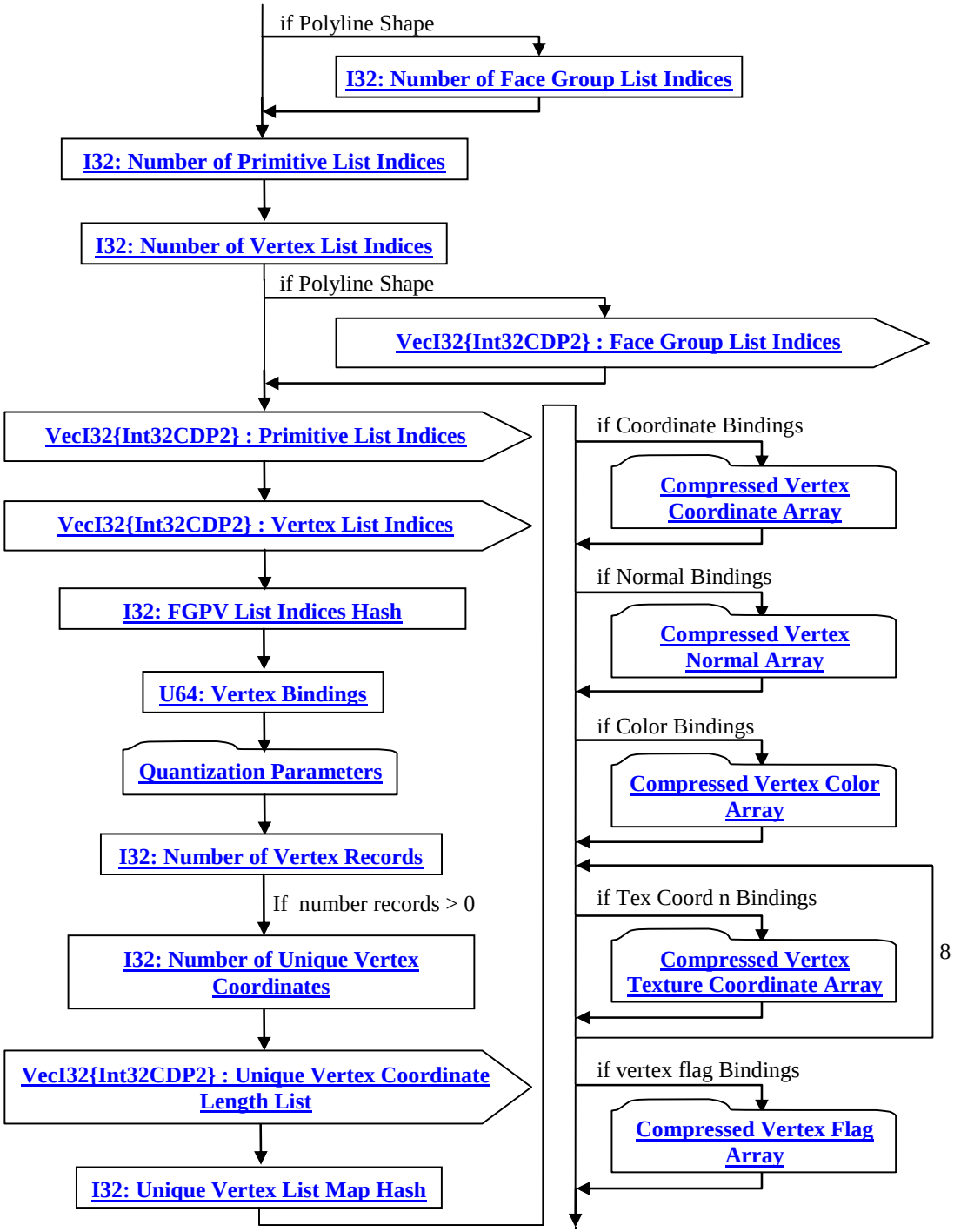
I32 : Number of Vertex Attributes

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 must 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.

7.2.2.1.2.7 TopoMesh Compressed Rep Data V1

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

Figure 91: TopoMesh Compressed Rep Data V1 data collection



Complete description for Quantization Parameters can be found in 7.2.1.1.1.10.2.1.1 Quantization Parameters.

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 i

I32: Unique Vertex List Map Hash

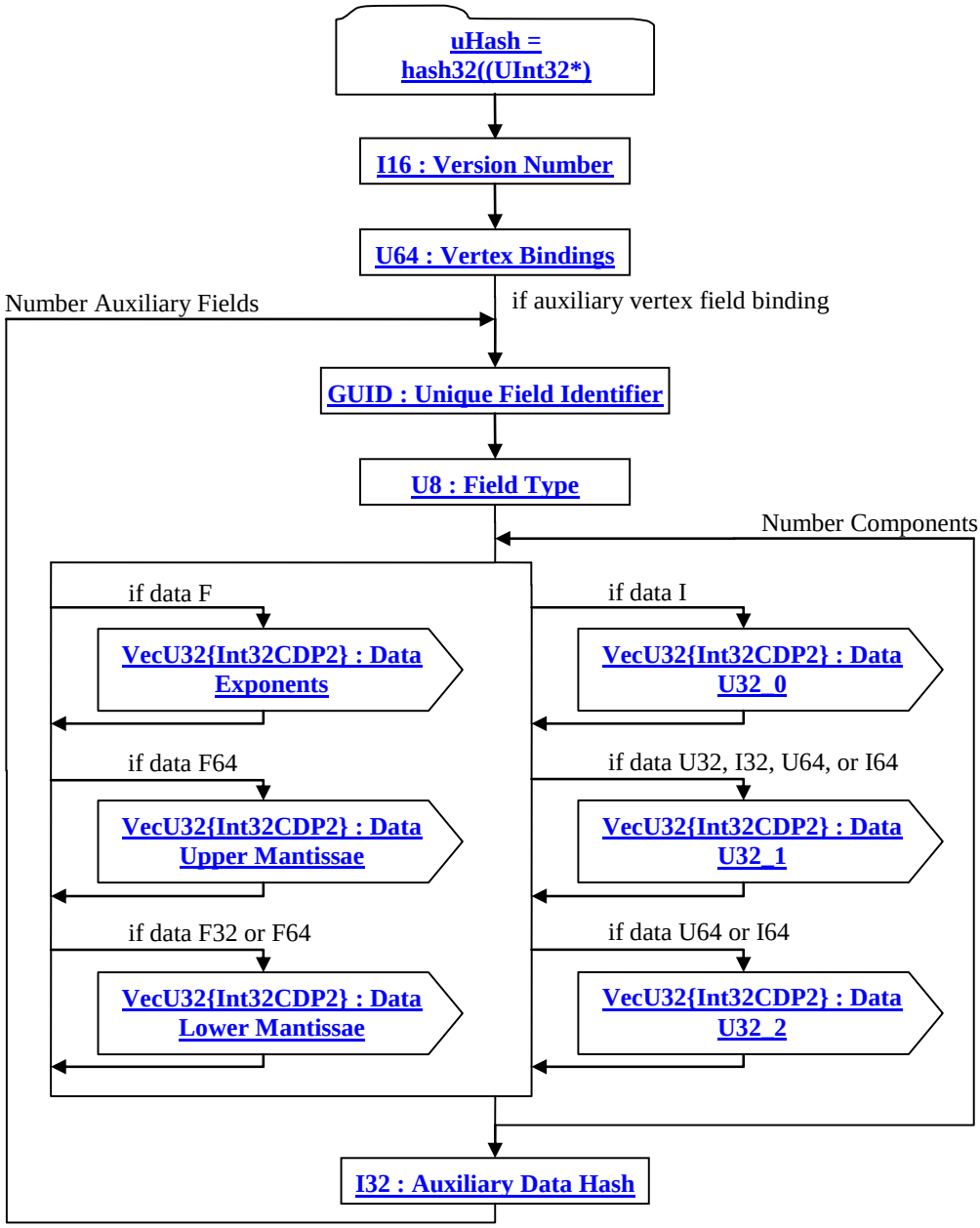
The Unique Vertex List Map Hash is the hash value of Unique Vertex Coordinate Length List. Refer to section 9.5 for a more detailed description on hashing.

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

7.2.2.1.2.8 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. Auxiliary fields are parallel to the existing vertex record information and contain additional information pertaining to each vertex.

Figure 92: 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

currently the only valid value.

U64 : Vertex Bindings

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

GUID : Unique Field Identifier

Each Auxiliary Vertex Field is associated with Unique Field Identifier to denote the usage of the contained data. All Unique Field Identifiers are currently reserved. These identifiers are intended to be unique across all application domains, therefore any JT file producer wishing to "lock down" a Unique Field Identifier so that others can rely on its semantic identity should contact the JTOpen industry liaison to obtain them.

U8 : Field Type

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

Type	Data	Components
1	U8	1
2	U8	2
3	U8	3
4	U8	4
5	I8	1
6	I8	2
7	I8	3
8	I8	4
9	U16	1
10	U16	2
11	U16	3
12	U16	4
13	I16	1
14	I16	2
15	I16	3
16	I16	4
17	U32	1
18	U32	2
19	U32	3
20	U32	4
21	I32	1
22	I32	2
23	I32	3

Type	Data	Components
24	I32	4
25	U64	1
26	U64	2
27	U64	3
28	U64	4
29	I64	1
30	I64	2
31	I64	3
32	I64	4
33	F32	1
34	F32	2
35	F32	3
36	F32	4
37	F32	2x2
38	F32	3x3
39	F32	4x4
40	F64	1
41	F64	2
42	F64	3
43	F64	4
44	F64	2x2
45	F64	3x3
46	F64	4x4

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.

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 9.5 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 );
  }
}
```

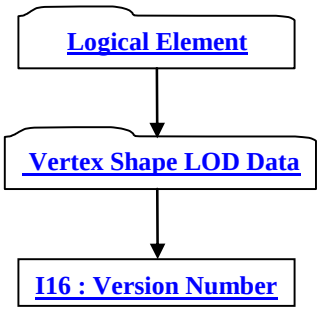
7.2.2.1.3 Tri-Strip Set Shape LOD Element

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

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 [4].

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 7.2.1.1.1.10.3 Tri-Strip Set Shape Node Element and 0 Late Loaded Property Atom ElementLate Loaded Property Atom Element respectively).

Figure 93: Tri-Strip Set Shape LOD Element data collection



Complete description for Logical Element Header can be found in 7.1.3.2.1Logical Element Header.

Complete description for Vertex Shape LOD Data can be found in 7.2.2.1.2.1 Vertex Shape LOD Data.

I16 : Version Number

Version Number is the version identifier for this Tri-
valid value.

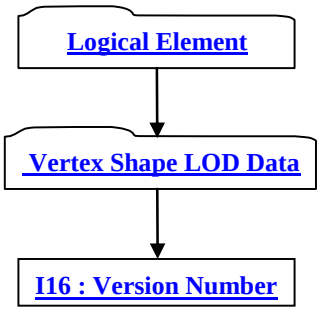
7.2.2.1.4 Polyline Set Shape LOD Element

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 7.2.1.1.1.10.5 Polyline Set Shape Node Element and 0 Late Loaded Property Atom Element respectively).

Figure 94: Polyline Set Shape LOD Element data collection



Complete description for Logical Element Header can be found in 7.1.3.2.1Logical Element Header.

Complete description for Vertex Shape LOD Data can be found in 7.2.2.1.2.1 Vertex Shape LOD Data.

I16 : Version Number

Version Number is the version identifier
valid value.

7.2.2.1.5 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 o

I16 : Version Number

Version Number is the version identifier for this Null Shape LOD Element
valid value.

BBoxF32 : Untransformed BBox

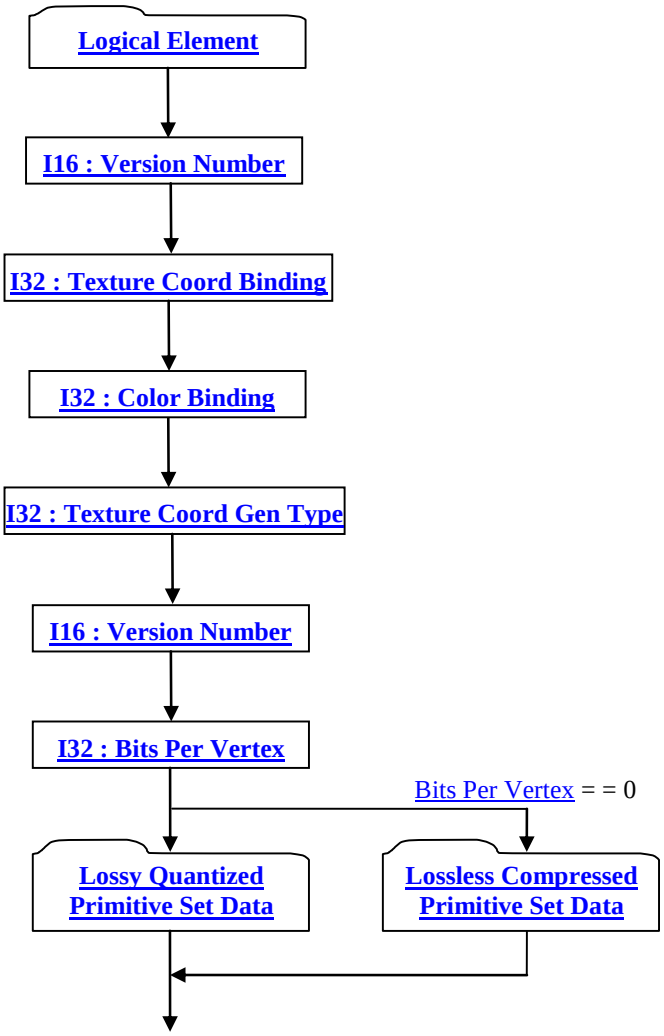
The Untransformed BBox is an axis-aligned LCS bounding box and represents the untransformed extents for this Null Shape LOD Element.

7.2.2.2 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 (
vertices, polygons, normals, etc) for LODs is not directly stored; instead some basic shape information is stored (e.g. sphere center and radius) from which LODs can be generated.

Figure 97: Primitive Set Shape Element data collection



Complete description for Logical Element Header can be found in 7.1.3.2.1Logical Element Header.

I16 : Version Number

Version Number is the version identifier for this element. Only version number 0x0001 is valid for now

I32 : Texture Coord Binding

Texture Coord Binding specifies
values are as follows:

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

I32 : Color Binding

C
as documented for Texture Coord Binding data field.

I16 : Version Number

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

= 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 must be within range [0:32] inclusive.

I32 : Texture Coord Gen Type

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

= 0	image will be applied to significant primitive features (i.e. cube face, cylinder wall, end cap) no matter how eccentrically shaped.
= 1	surfaces such that a mapped texel stays approximately square.

7.2.2.2.1 Lossless Compressed Primitive Set Data

The Lossless Compressed Primitive Set Data collection contains all the per-primitive information
compression format for all primitives in the Primitive Set. The Lossless Compressed Primitive Set Data collection is only present when the Bits Per Vertex 7.2.2.2 Primitive Set Shape Element for complete description).

Figure 98: Lossless Compressed Primitive Set Data collection

I32 : Uncompressed Data Size

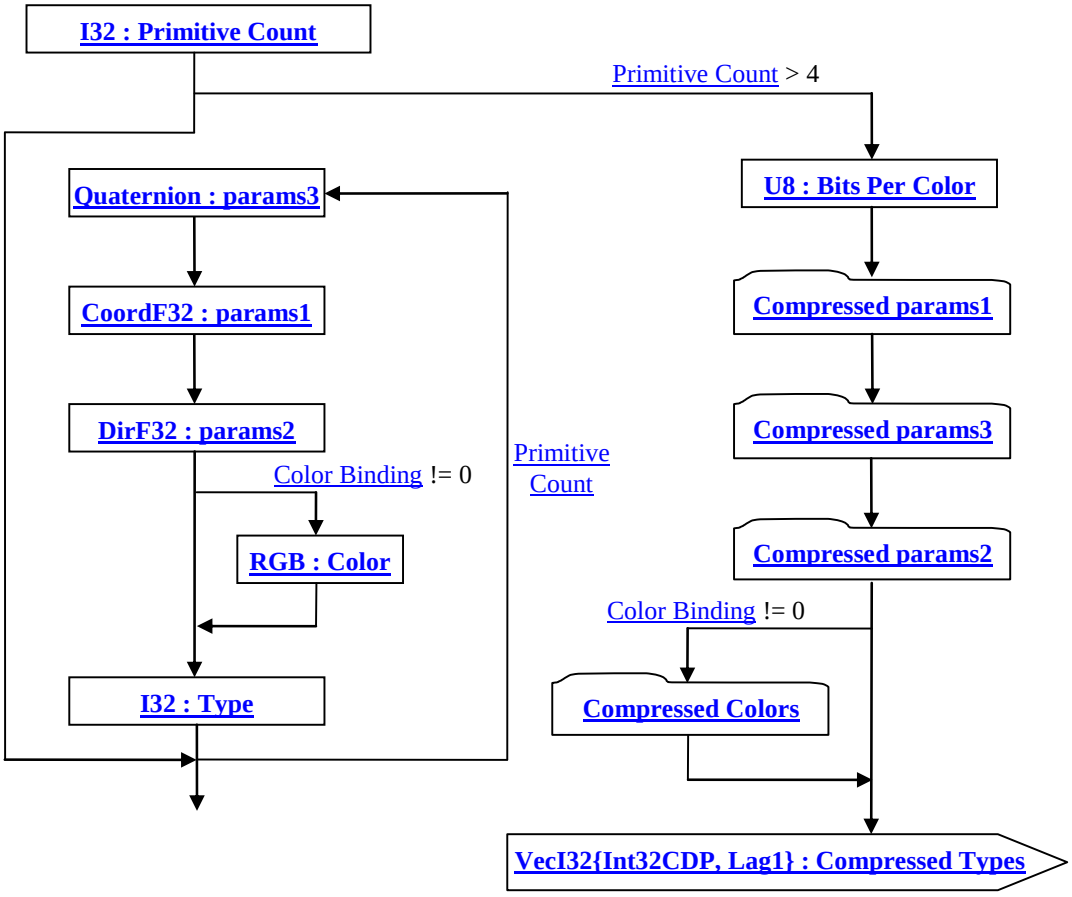
Uncompressed Data size specifies the uncompressed size of Primitive Data or Compressed Primitive Data in bytes.

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

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 center X	base center Y	base center Z	spine X	spine Y	spine Z	radi5 0 T			

Figure 99: 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 6: Primitive Set Interpretation in 7.2.2.2.1 Lossless Compressed Primitive Set Data for per-primitive type description of the params3 data fields.

CoordF32 : params1

Interpretation of params1 data field is primitive Type dependent. See Table 6: Primitive Set Interpretation in 7.2.2.2.1 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 6: Primitive Set Interpretation in 7.2.2.2.1 Lossless Compressed Primitive Set Data for per-primitive type description of the params1 data fields.

RGB : Color

Color specifies the Red, Green Blue color components for the primitive. This data field is only present if previously read Color Binding (see 7.2.2.2 Primitive Set Shape Element

I32 : Type

Type specifies the primitive type. See Table 5: Primitive Set Primitive Data Elements in 7.2.2.2.1 Lossless Compressed Primitive Set Data for valid primitive Type values.

U8 : Bits Per Color

Bits Per Color specifies the number of quantization bits per color component. Value must be within range [0:32] inclusive.

VecI32{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 decode data. In an uncompressed form the valid primitive Type vales are as documented in

7.2.2.2.2.2 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 6: Primitive Set Data Fields Interpretation in 7.2.2.2.1 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*

Quantizer is a scalar quantizer/encoder whose range is divided into levels of equal spacing). See 8 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*

params3 data.

The storage format of Compressed params3 is exactly the same as that documented in Figure 100: Compressed params1 data collection.

7.2.2.2.2.3 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 6: Primitive Set Data Fields Interpretation in 7.2.2.2.1 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 values. Since *params2*

m Quantizers (where a Uniform Quantizer is a scalar quantizer/encoder whose range is divided into levels of equal spacing). See 8 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*

params2 data.

The storage format of Compressed params2 is exactly the same as that documented in Figure 100: Compressed params1 data collection.

7.2.2.2.2.4 Compressed Colors

Compressed Colors is the compressed representation of the *color* data for all the primitives in the Primitive Set. This data collection is only present if previously read Color Binding (see 7.2.2.2 Primitive Set Shape Element

The *color* data for all primitives in the Primitive Set is compressed/encoded on a per ordinate basis using a separate Uniform Quantizer (with Bits Per Color number of quantization bits) for each collection of ordinate values. Since *color* is of type

scalar quantizer/encoder whose range is divided into levels of equal spacing). See 8 Data Compression and Encoding for more complete description of Uniform Quantizer.

The JT Format packs all the

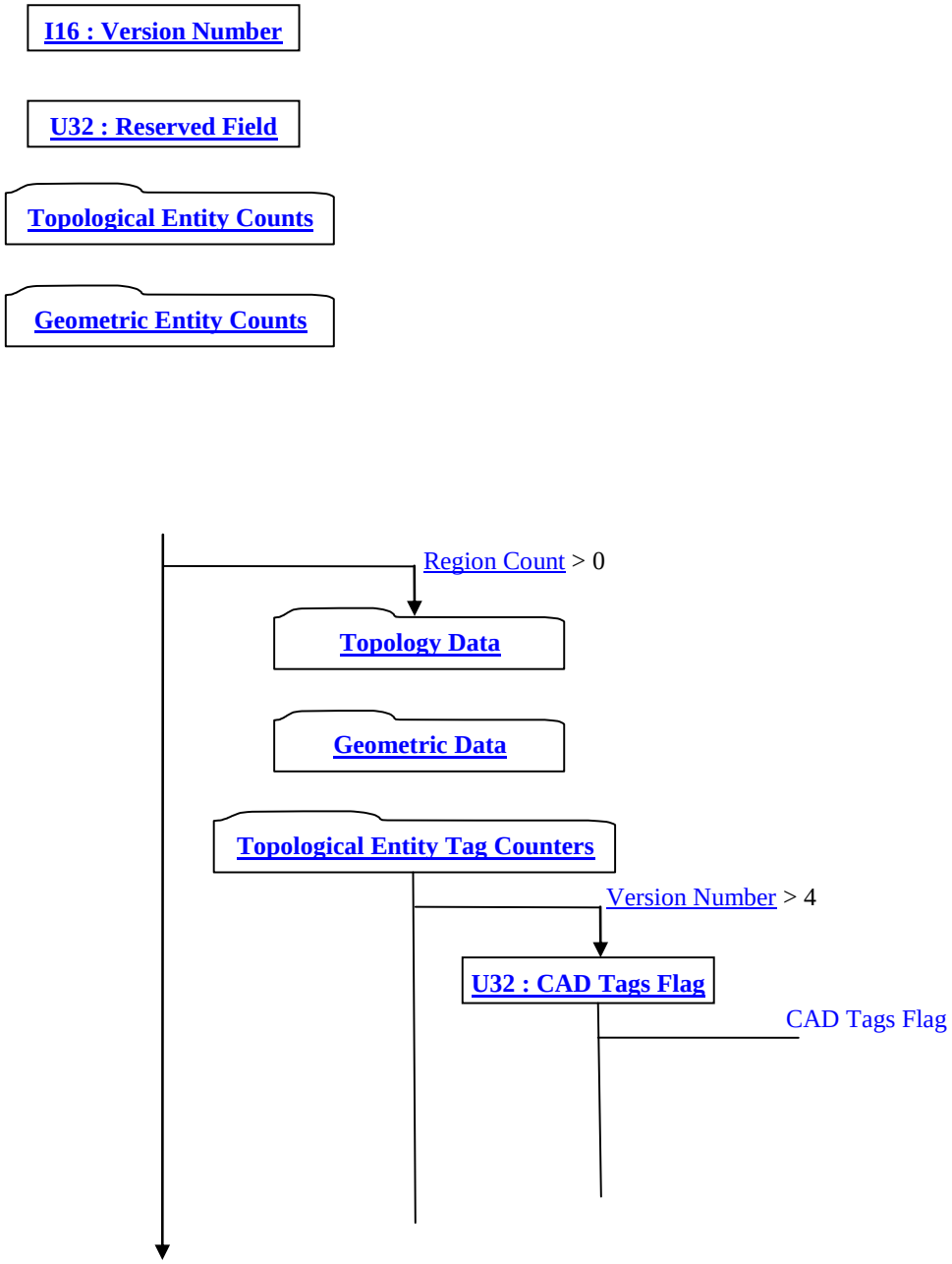
two faces using it.

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 [10], [11] and [12] listed in 3 References and Additional Information section of this document.

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.

Figure 102: JT B-Rep Element data collection



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

I16 : Version Number

Version Number is the version identifier for this JT B-Rep Element. Only version number 0x0001 is currently defined.

U32 : Reserved Field

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

CoordF64 : Reserved Field

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

F64 : Reserved Field

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

U32

I32 : Loop Count

Loop Count indicates the number of topological loop entities in the B-Rep

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

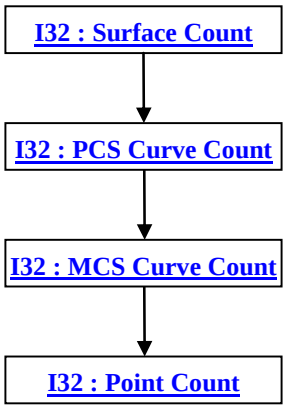
I32 : Vertex Count

Vertex Count indicates the number of topological vertex entities in the B-Rep

7.2.3.1.2 Geometric Entity Counts

Geometric Entity Counts data collection defines the counts for each of the various geometric entities within a B-Rep.

Figure 104: 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

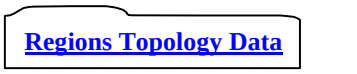
MCS Curve Count indicates the number of distinct geometric (Model Coordinate Space) curves (i.e. XYZ curve) entities in the B-Rep.

I32 : Point Count

Point Count indicates the number of distinct geometric point entities in the B-Rep.

7.2.3.1.3 Topology Data

Figure 105: Topology Data collection

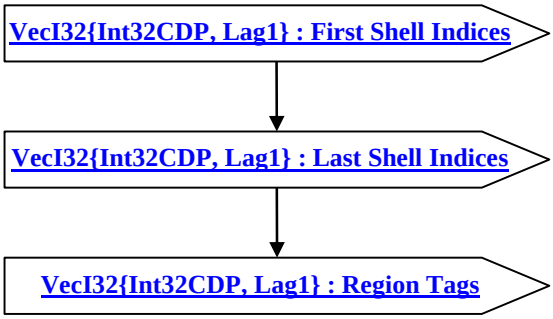


7.2.3.1.3.1 Regions Topology Data

Regions Topology Data defines the set of non-overlapping Shells comprising each Region. The volume of a Region is that volume - -
Region. A Region is analogous to a dimensionally elevated face where Region corresponds to Face and Shell corresponds to Trim Loop.

Each 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).

Figure 106: Regions Topology Data collection



VecI32{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.

VecI32{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.

VecI32{Int32CDP, Lag1} : Region Tags

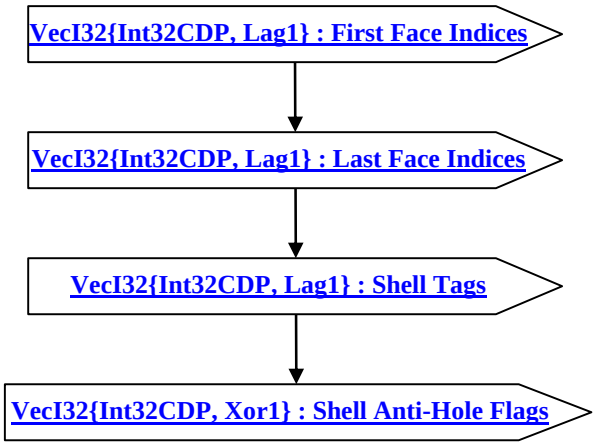
Each Region has an identifier tag. Region Tags is a vector of identifier tags for a set of Regions. Region Tags uses the Int32 version of the CODEC to compress and encode data.

7.2.3.1.3.2 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 in -

Each 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 107: Shells Topology Data collection



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:

= 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.

7.2.3.1.3.3 Faces Topology Data

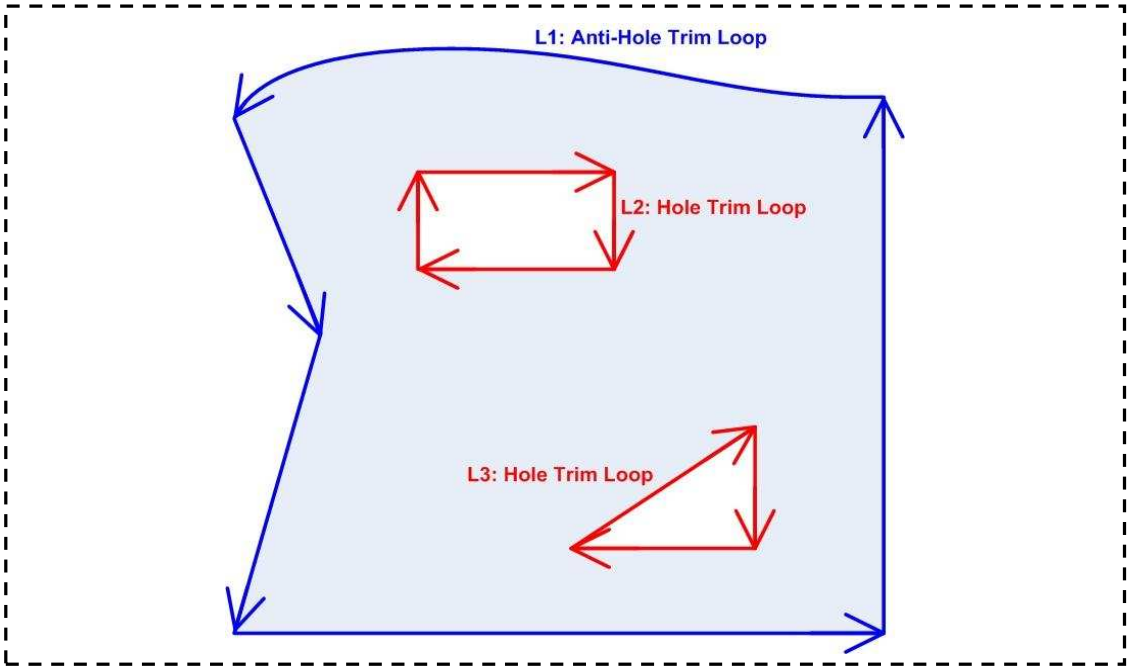
A Face is a two-dimensional topological building block defined as the active (that portion to be used in the model) regions/areas of a Geometric Surface; where active regions/areas of a Geometric Surface are indicated using oriented Trim Loops. Faces Topology Data specifies the underlying Geometric Surface and Trim Loops making up each Face along with a

.

A Face must be trimmed with at least one anti-hole Trim Loop and zero or more hole Trim Loops. Thus the area of the - nd outside each No i- or be tangent at any point. - with a counter-clockwise orientation in the underlying surface's parameter space where with a clockwise orientation. With this Trim Loop orientation definition, as one traverses a Trim Loop of a Face, the material Figure 108 gives an example in parameter space of proper trim loop definition and orientation -

each hole is always represented by a separate

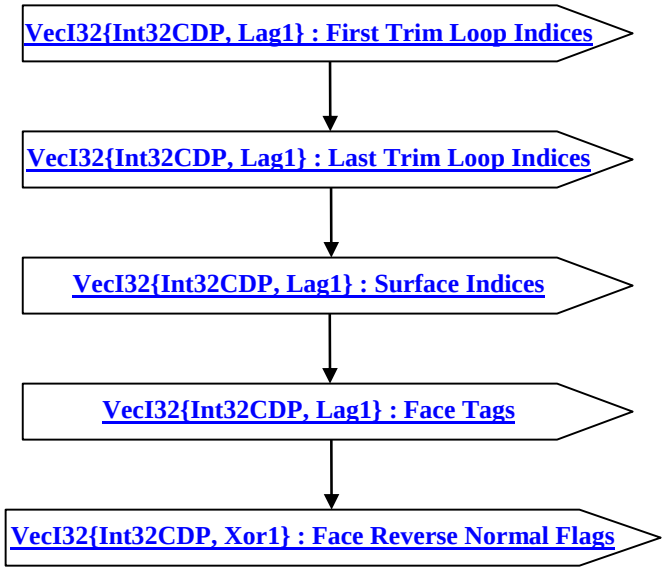
Figure 108: Trim Loop example in parameter Space - One Face with 2 Holes



by an index into a list of Geometric Surfaces. Each Face

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 109: 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.

Vecl32{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.

Vecl32{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.

Vecl32{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.

Vecl32{Int32CDP, Xor1} : Face Reverse Normal Flags

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:

= 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.

7.2.3.1.3.4 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 must be closed and non-self-intersecting.

in each Loop (i.e. all CoEdges inclusive between the specified first and last CoEdge list index define the particular Loop).

Figure 110: Loops Topology Data collection

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

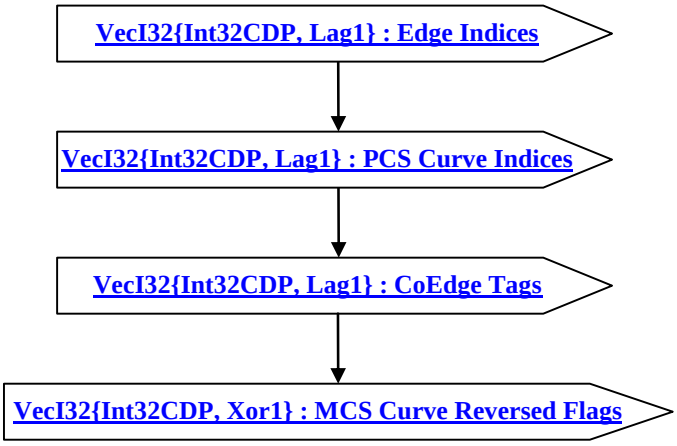
In an uncompressed/decoded form the flag values have the following meaning:

= 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.

7.2.3.1.3.5 CoEdges Topologically Connected to Edges

Figure 111: CoEdges Topology Data collection



VecI32{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.

VecI32{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.

VecI32{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.

VecI32{Int32CDP, Xor1} : MCS Curve Reversed Flags

Each CoEdge has a flag indicating whether the directional
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:

= 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.

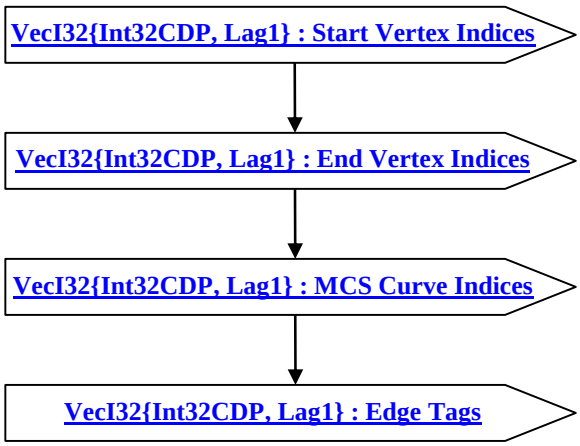
MCS Curve Reversed Flags uses the Int32 version of the CODEC to compress and encode data.

7.2.3.1.3.6 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 112: Edges Topology Data collection



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.

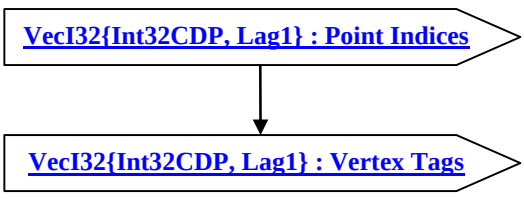
7.2.3.1.3.7 Vertices Topology Data

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 113: Vertices Topology Data collection



VecI32{Int32CDP, Lag1} : Point Indices

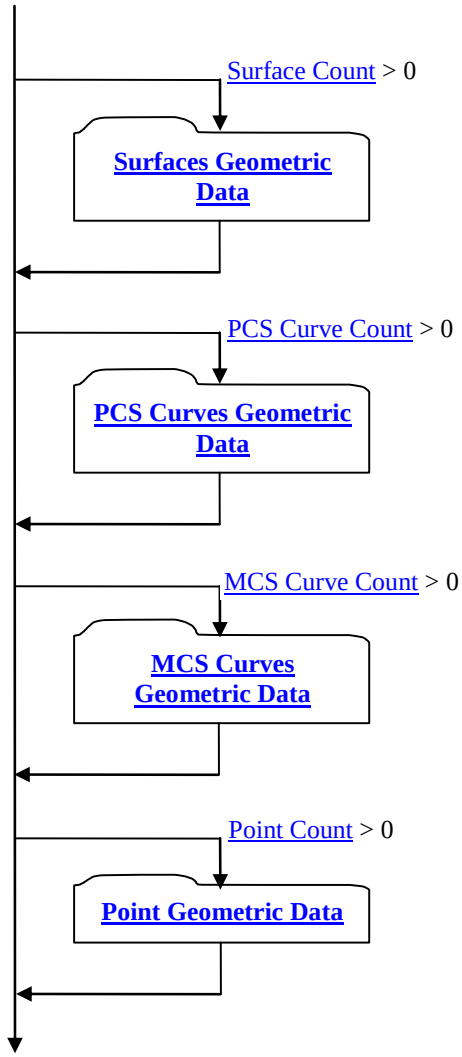
Point Indices is a vector of indices representing the index of the geometric point for each Vertex. Point Indices uses the Int32 version of the CODEC to compress and encode data.

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.

7.2.3.1.4 Geometric Data

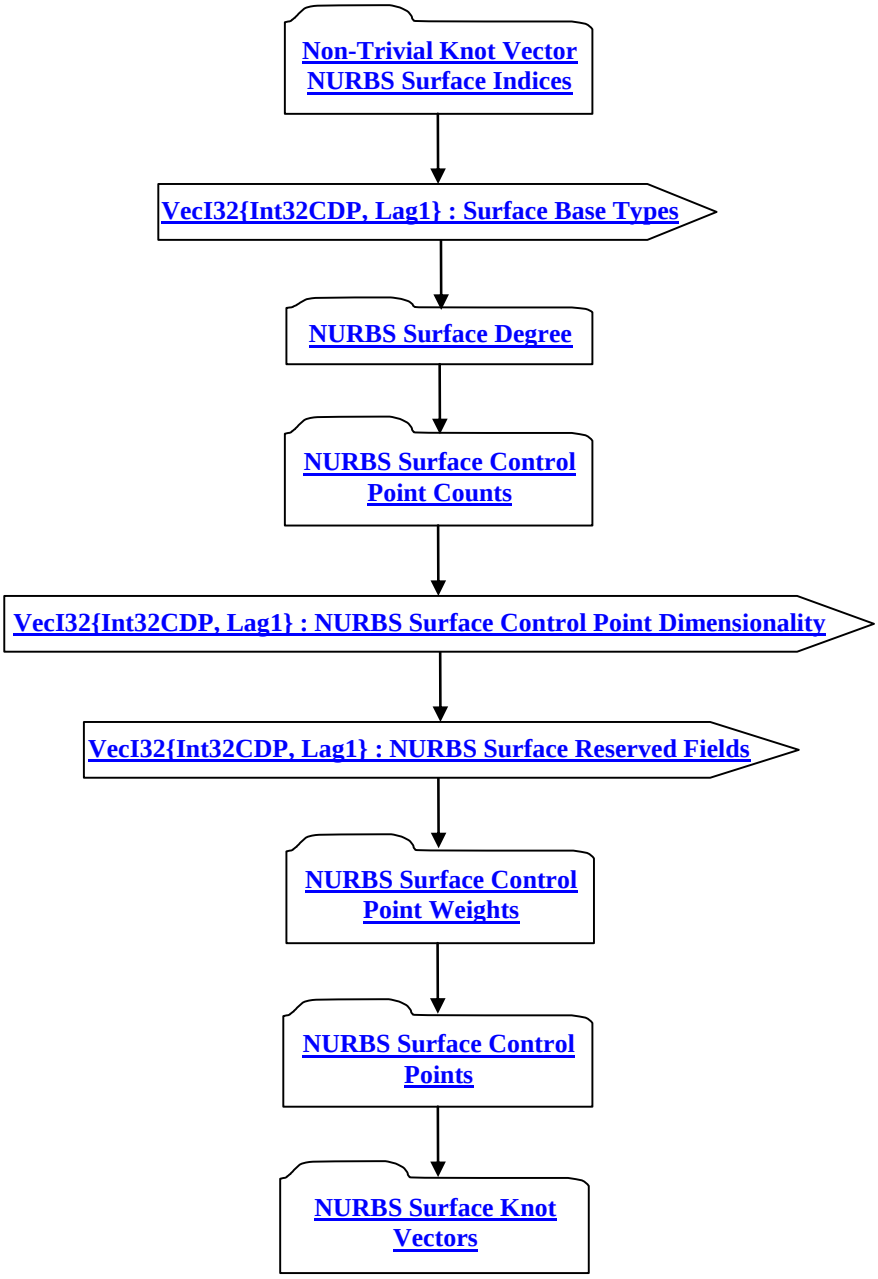
Figure 114: Geometric Data collection



7.2.3.1.4.1 Surfaces Geometric Data

Surfaces Geometric Data collection contains the JT B-Rep 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 7.2.3.1.2 Geometric Entity Counts.

Figure 115: Surfaces Geometric Data collection



VecI32{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:

= 1	Surface is a NURBS surface

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:

= 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 Reserved Fields

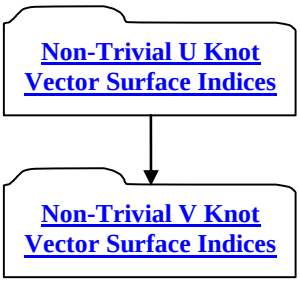
NURBS Surface Reserved Fields is a vector of data reserved for future expansion of the JT format. Each NURBS Surface in a list of Surfaces has one reserved data field entry in this NURBS Surface Reserved Fields vector. NURBS Surface Reserved Fields uses the Int32 version of the CODEC to compress and encode data

7.2.3.1.4.1.1 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 - 8.1.13 Compressed Entity List for Non-Trivial Knot Vector.

This Surface index data is stored in a compressed format.

Figure 116: Non-Trivial Knot Vector NURBS Surface Indices data collection

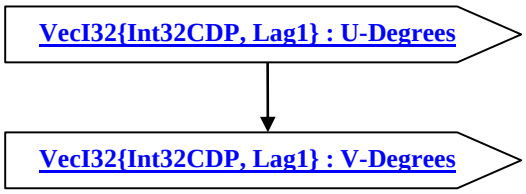


Both Non-Trivial U Knot Vector Surface Indices and Non-Trivial V Knot Vector Surface Indices have the same data format as that documented for data collection 8.1.13 Compressed Entity List for Non-Trivial Knot Vector.

7.2.3.1.4.1.2 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.

Figure 117: NURBS Surface Degree data collection



VecI32{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.

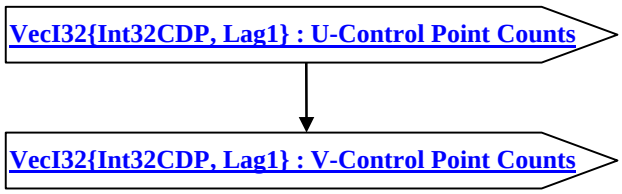
VecI32{Int32CDP, Lag1} : V-Degrees

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.

7.2.3.1.4.1.3 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 118: NURBS Surface Control Point Counts data collection



VecI32{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.

VecI32{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.

7.2.3.1.4.1.4 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 Contr - is stored for the

The NURBS Surface Control Point Weights data is stored in a compressed format.

Figure 119: NURBS Surface Control Point Weights data collection

Complete description for Compressed Control Point Weights Data can be found in 8.1.14 Compressed Control Point Weights Data.

7.2.3.1.4.1.5 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 120: NURBS Surface Control Points data collection

VecF64{Float64CDP, NULL} : Control Points

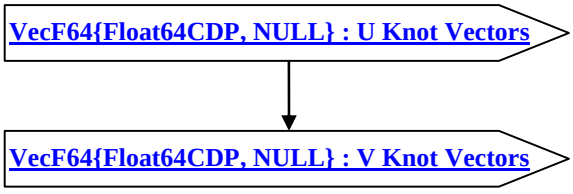
Control Points is a vector of Control Point coordinates for all the NURBS Surfaces in a list of Surfaces. All the NURBS Surfaces Control Point coordinates are cumulated into this single vector in the same order as the Surface appears in the Surface list (i.e. Surface-1 U Control Points, Surface-1 V Control 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 .

7.2.3.1.4.1.6 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 7.2.3.1.4.1.1 Non-Trivial Knot Vector NURBS Surface Indices.

The knot vector data for the list of Surfaces is stored in a compressed format.

Figure 121: 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.

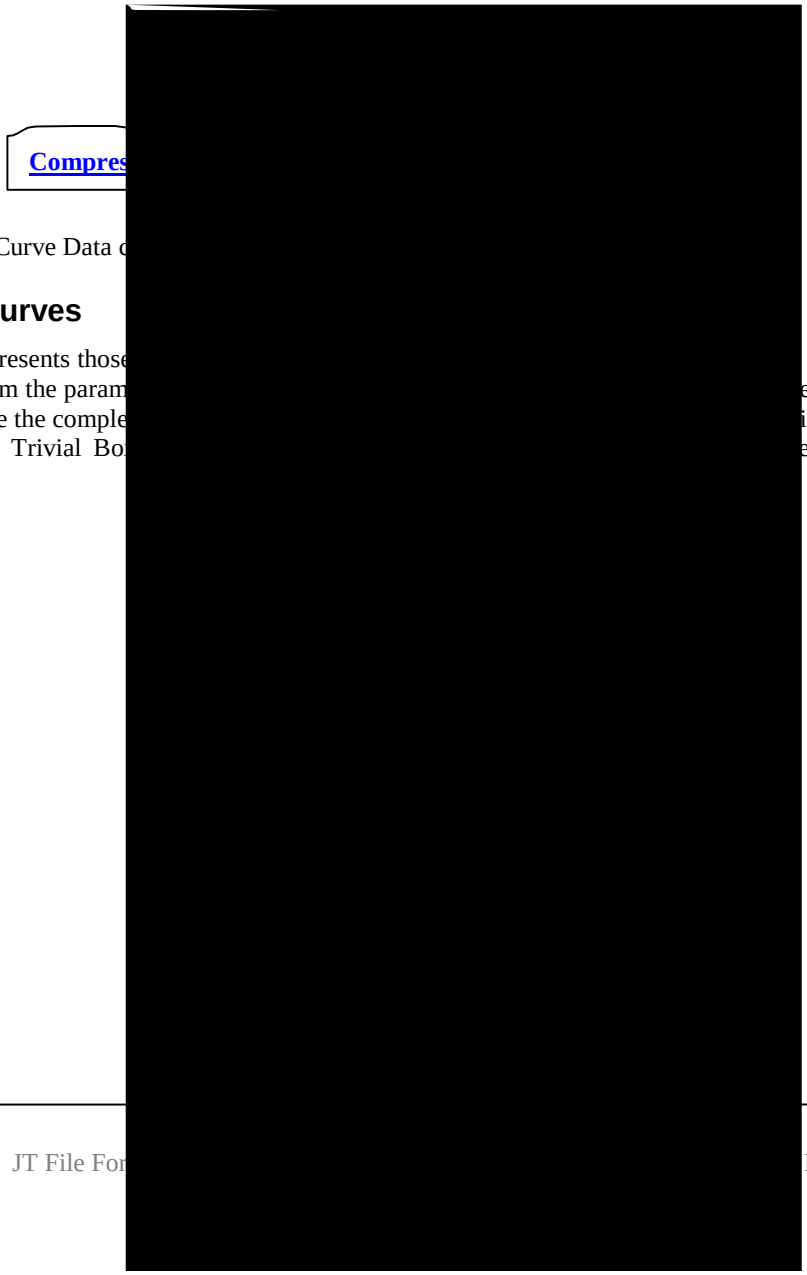
7.2.3.1.4.2 PCS Curves Geometric Data

PCS Curves Geometric Data collection contains the JT B-Rep
Curve data). This geometric PCS Curve data is divided up into two collection types; one data collection for what are
nd one data collection for compressed/encoded PCS NURBS Curve data.

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 7.2.3.1.2 Geometric Entity Counts.

Figure 122: PCS Curves Geometric Data collection

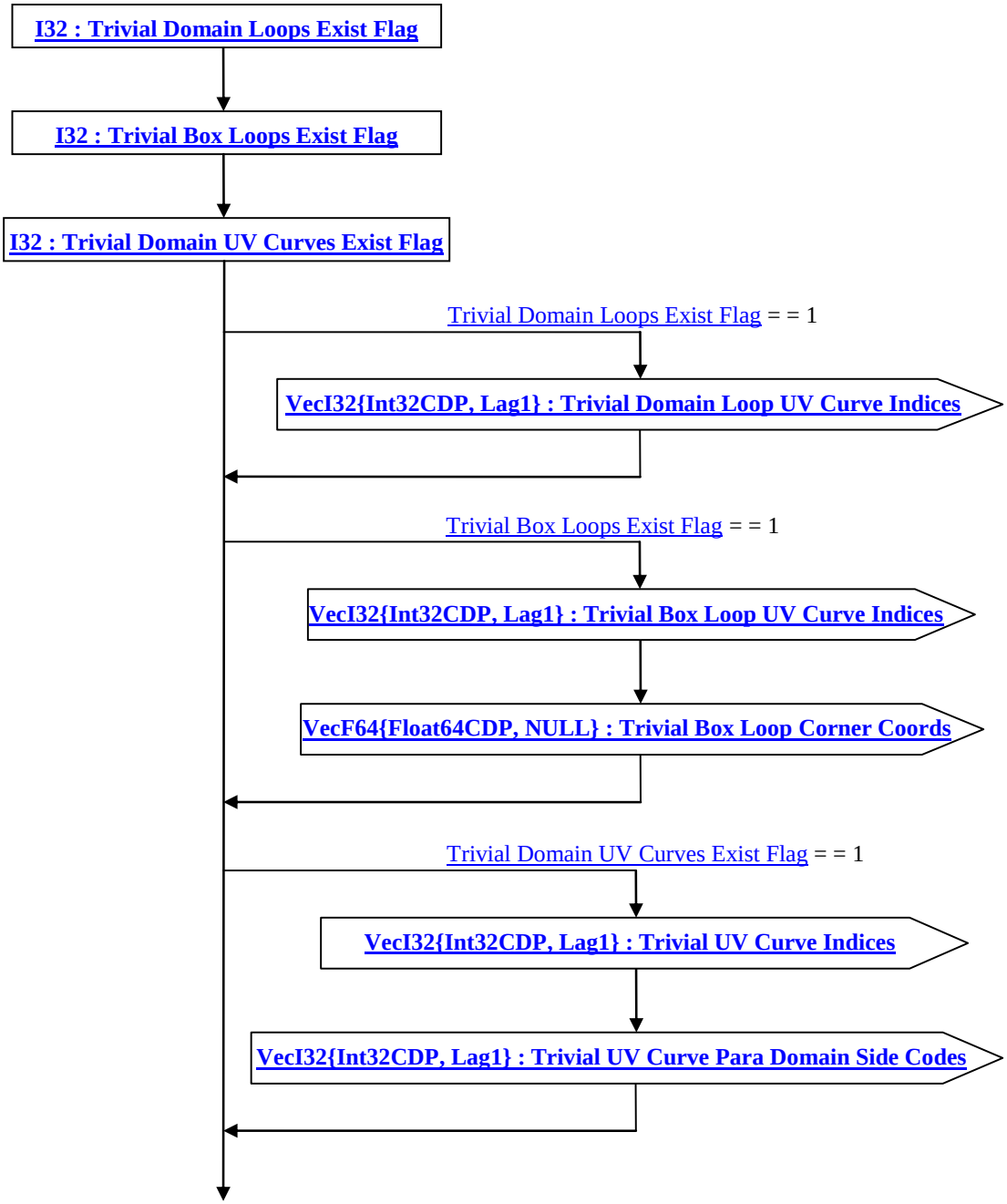


Complete description for Compressed Curve Data c

7.2.3.1.4.2.1 Trivial PCS Curves

Trivial PCS Curves data collection represents those
UV curve definition can be derived from the param e data for
each UV curve (i.e. do not have to store the comple into three
classifications (Trivial Domain Loop, Trivial Bo ed in the
following sub-sections.

Figure 123: Trivial PCS Curves data collection



I32 : Trivial Domain Loops Exist Flag

Trivial Domain Loops Exi
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 must always be made up of four Trivial Domain UV curves.

= 0	Trivial Domain Loops do not exist.

= 1	Trivial Domain Loops exist.

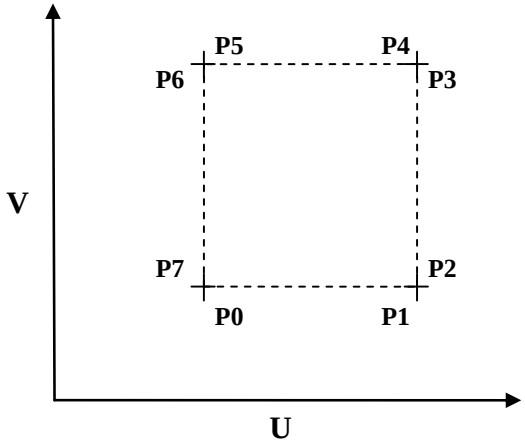
I32 : Trivial Box Loops Exist Flag

t

forms a rectangle (i.e. corresponding curve end coordinates of opposite sides of the box are equal). Given this criteria a Trivial Box Loop must always be made up of four UV curves

= 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 as follows:



P0[0] ± P5[0] = 0
P1[0] ± P4[0] = 0
P2[1] ± P7[1] = 0
P3[1] ± P6[1] = 0

I32 : Trivial Domain UV Curves Exist Flag

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).

= 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

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.

VecI32{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.

VecI32{Int32CDP, Lag1} : Trivial UV Curve Para Domain Side Codes

Trivial UV Curve Para Domain Side Codes is a vector containing a side code
parametric domain side the UV Curve lies on.

In an uncompressed/decoded form the parametric domain side values have the following meaning:

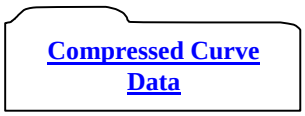
= 0	Bottom side of parametric domain
= 1	Right side of parametric domain
= 2	Top side of parametric domain
= 3	Left side of parametric domain

Trivial UV Curve Para Domain Side Codes uses the Int32 version of the CODEC to compress and encode data.

7.2.3.1.4.3 MCS Curves Geometric Data

MCS Curves Geometric Data collection contains the JT B-Rep
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 7.2.3.1.2 Geometric Entity Counts.

Figure 124: MCS Curves Geometric Data collection

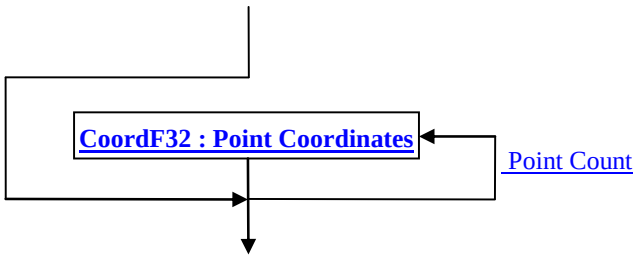


Complete description for Compressed Curve Data can be found in 8.1.15 Compressed Curve Data.

7.2.3.1.4.4 Point Geometric Data

Point Geometric Data collection contains the JT B-Rep Point data. Each Point is simply represented by a The count/number of Points within a JT B-Rep is indicated by data field Point Count documented in 7.2.3.1.2 Geometric Entity Counts.

Figure 125: Point Geometric Data collection



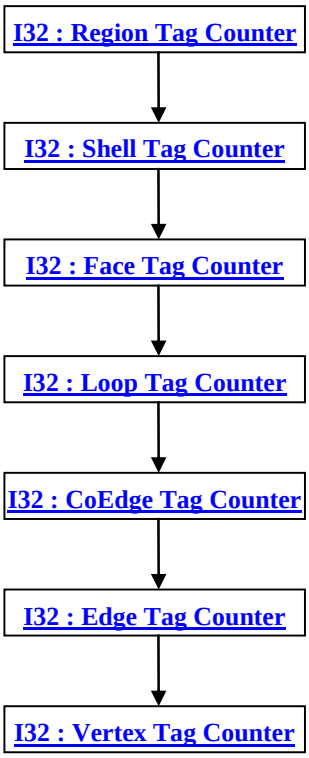
CoordF32 : Point Coordinates

Point Coordinates specifies the XYZ coordinate components for a Point.

7.2.3.1.5 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.

Figure 126: Topological Entity Tag Counters data collection



I32 : Region Tag Counter

I32 : Shell Tag Counter

Shell Tag Counter specifies the

I32 : Face Tag Counter

I32 : Loop Tag Counter

ntity.

I32 : CoEdge Tag Counter

I32 : Edge Tag Counter

I32 : Vertex Tag Counter

Vertex T

7.2.3.1.6 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 7.2.3.1 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 Face Count + Edge Count 7.2.3.1.1 Topological Entity Counts.

Figure 127: B-Rep CAD Tag Data collection

Complete description for Compressed CAD Tag Data can be found in 8.1.16 Compressed CAD Tag Data.

7.2.4 XT B-Rep Segment

XT B-Rep Segment contains an Element that defines the precise geometric Boundary Representation data for a particular Part in Parasolid boundary representation (XT) 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 7.2.3 JT B-Rep Segment.

XT B-Rep Segments are typically referenced by Part Node Elements (see 7.2.1.1.1.5Part Node Element) using Late Loaded Property Atom Elements (see 0Second specifies the date Second value. Valid values are [0, 59] inclusive.

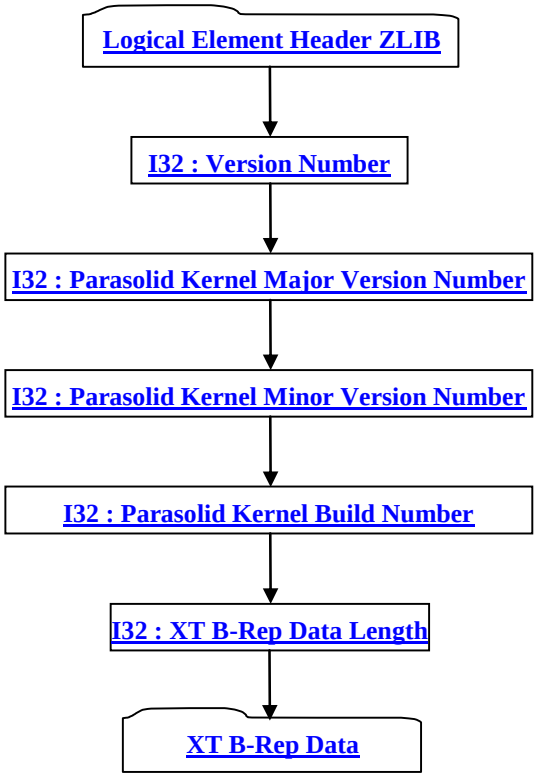
Late Loaded Property Atom Element). The XT B-Rep Segment type supports ZLIB compression on all element data, so all elements in XT B-Rep Segment use the Logical Element Header ZLIB form of element header data.

7.2.4.1 XT B-Rep Element

Object Type ID: 0x873a70e0, 0x2ac9, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

XT B-Rep Element

Figure 128: XT B-Rep Element data collection



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

I32 : Version Number

Version Number is the version identifier for this XT B-Rep 2
for v9 JT files.

I32 : Parasolid Kernel Major Version Number

Parasolid Kernel Major Version Number specifies the major version number for the revision of Parasolid that wrote the XT B-Rep data into JT File.

I32 : Parasolid Kernel Minor Version Number

Parasolid Kernel Minor Version Number specifies the minor version number for the revision of Parasolid that wrote the XT B-Rep data into JT File.

I32 : Parasolid Kernel Build Number

Parasolid Kernel Build Number specifies the build number for the revision of Parasolid that wrote the XT B-Rep data into JT File.

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.

7.2.4.1.1 XT B-Rep Data

The XT B-Rep Data collection specifies the raw stream of bytes that Parasolid B-Rep Body(s) in an external file. The XT B-Rep Data

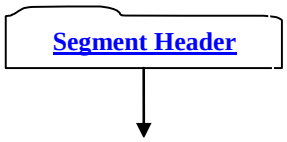
found in Appendix F: Parasolid XT Format Reference.

7.2.5 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 7.2.1.1.1.5 Part Node Element) using a Second specifies the date Second value. Valid values are [0, 59] inclusive.

Late Loaded Property Atom Element (see 0 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 129: Wireframe Segment data collection



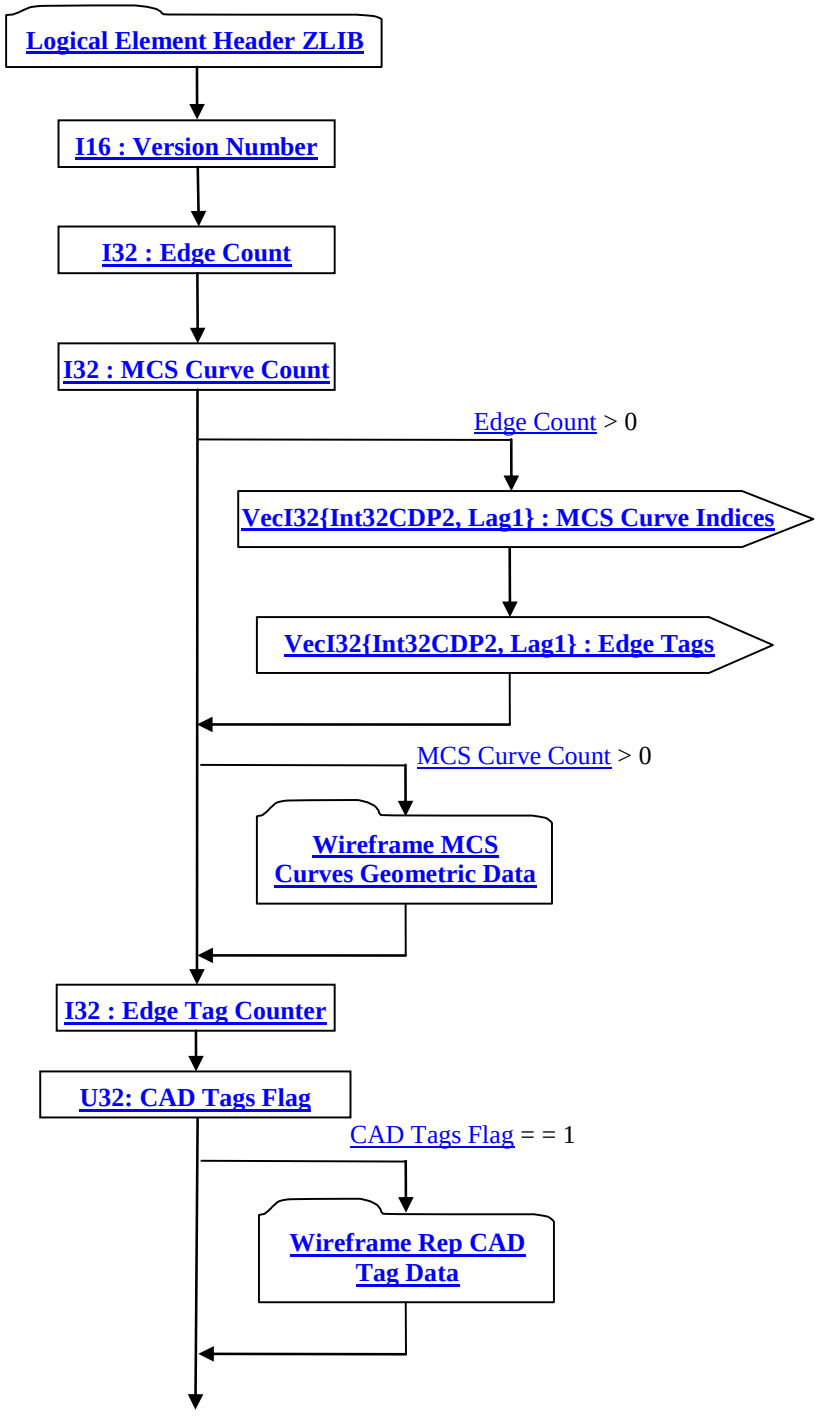
Complete description for Segment Header can be found in 7.1.3.1Segment Header.

7.2.5.1 Wireframe Rep Element

WireframeEle **Object Type ID:** 0x873a70d0, 0x2ac8, 0x11d1, 0x9b, 0x6b, 0x00, 0x80, 0xc7, 0xbb, 0x59, 0x97

A Wireframe Rep Element represents a particular s precise 3D wireframe data (e.g. reference curves, section curves). 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.

Figure 130: Wireframe Rep Element data collection



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

I16 : Version Number

I32 : Edge Count

Edge Count indicates the number of topological Edge entities in the Wireframe

Complete description for Compressed CAD Tag Data can be found in 8.1.16 Compressed CAD Tag Data.

7.2.6 Meta Data Segment

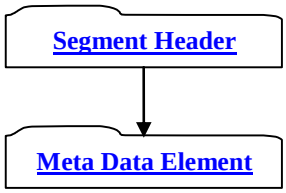
Meta Data Segments are used to store large collections of meta-data in separate addressable segments of the JT File. Storing meta-data in a separate addressable segment allows references (from within the JT file) to these segments to be constructed such that the meta-data can be late-loaded (i.e. JT file reader can be structured t
loading/reading of the referenced meta-data segment until it is actually needed).

Meta Data Segments are typically referenced by Part Node Elements (see 7.2.1.1.1.5Part Node Element) using Late Loaded Property Atom Elements (see 0 Late Loaded Property Atom ElementSecond specifies the date Second value. Valid values are [0, 59] inclusive.

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 133: Meta Data Segment data collection



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

The following sub-sections document the various I32 : Texture Coord Channel types.

7.2.6.1 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 l meta-data properties associated with a particular Meta Data Node Element (see 7.2.1.1.1.6 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.

Figure 134: Property Proxy Meta Data Element data collection

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

I16: Version Number

Version Number is the version identi

MbString : Property Key

Property Key specifies the *key* string for the property.

U8 : Property Value Type

Property Value Type specifies the data type for the Property Value.
Valid types include the following:

= 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.

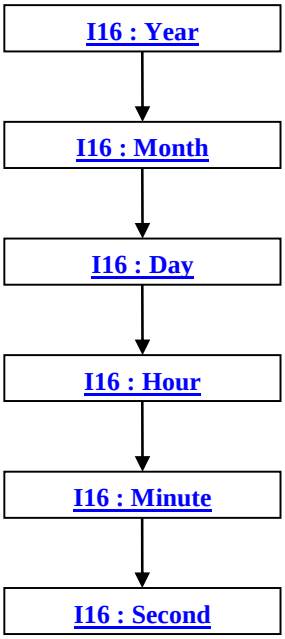
F32 : Float Property Value

Float Property Value represents the property value when Property Value Type = = 3.

7.2.6.1.1 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 135: Date Property Value data collection



I16 : Year

Year specifies the date year value.

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

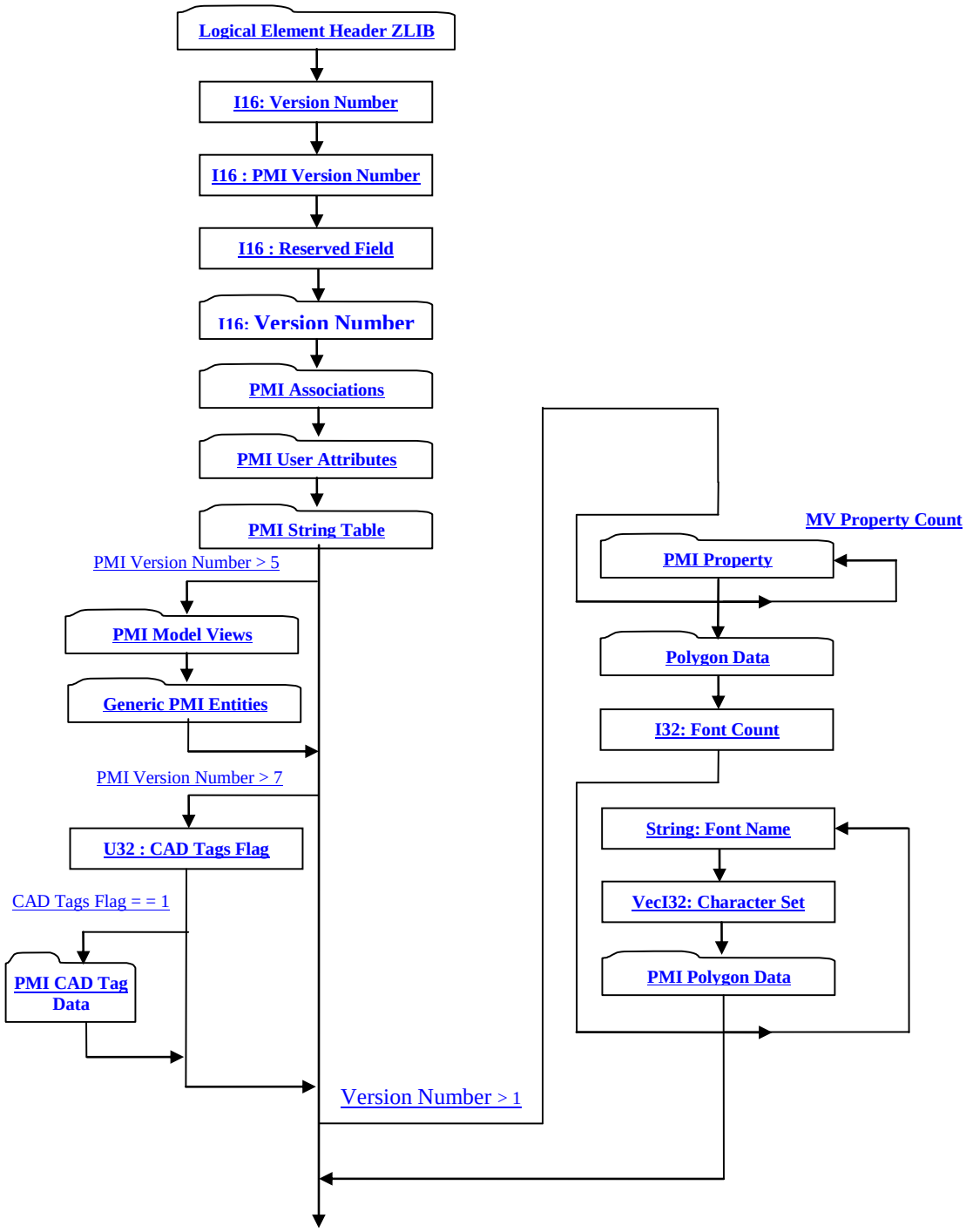
Second specifies the date Second value.

7.2.6.2 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.

Figure 136: PMI Manager Meta Data Element data collection



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

I16: Version Number

Version Number is the version identifier for this PMI Manager Element. Version numbers 0x0001 and 0x0002 are currently supported.

I16 : PMI Version Number

Version Number is the version identifier for the PMI. There are several PMI versions that must be supported for JT File format 8.1. This is because if an older JT File format containing PMI is read and then re-exported to JT File Format 8.1, the exported PMI data must be maintained in the version format originally read from the initial JT file (i.e. PMI data read from a JT File is not migrated to new version format when re-exported to another JT File format).

The valid PMI version numbers are as follows:

= 3	Version-3
= 4	Version-4
= 5	Version-5
= 6	Version-6
= 7	Version-7
= 8	Version-8

I16 : Reserved Field

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

U32 : CAD Tags Flag

CAD Tags Flag is a flag indicating whether CAD Tag data exist for the PMI.

I32: MV Property Count

Number of ModelViews in the PMI segment.

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

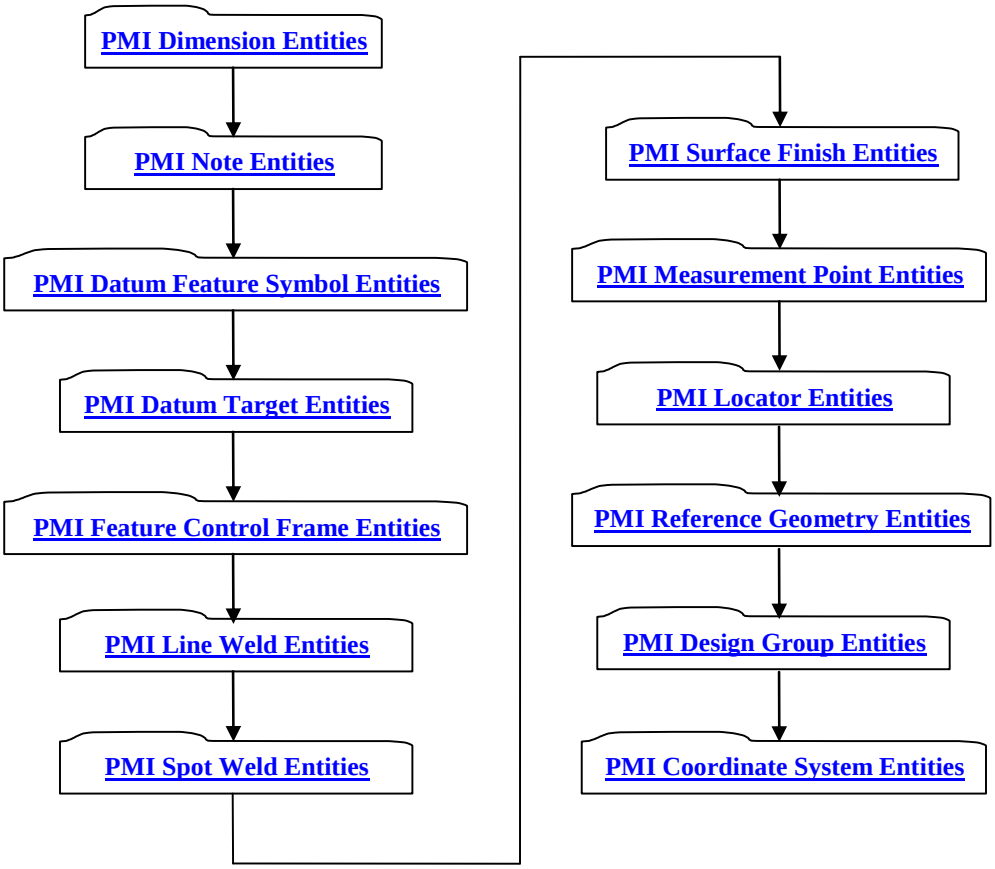
Font name specifies a representative name for the font set.

VecI32: Character Set

Integer identifiers for each character whose symbol is defined in the ensuing PolygonData segment.

7.2.6.2.1 PMI Entities

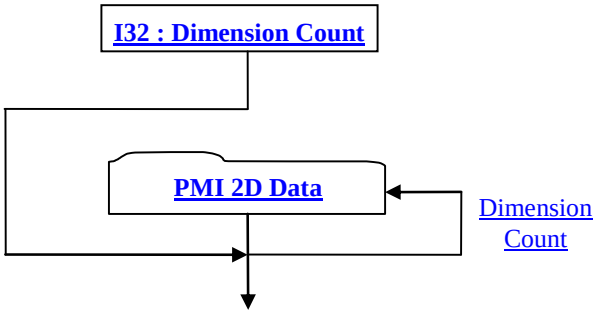
Figure 137: PMI Entities data collection



7.2.6.2.1.1 PMI Dimension Entities

The PMI Dimension Entities data collection defines data for a list of Dimensions.

Figure 138: PMI Dimension Entities data collection



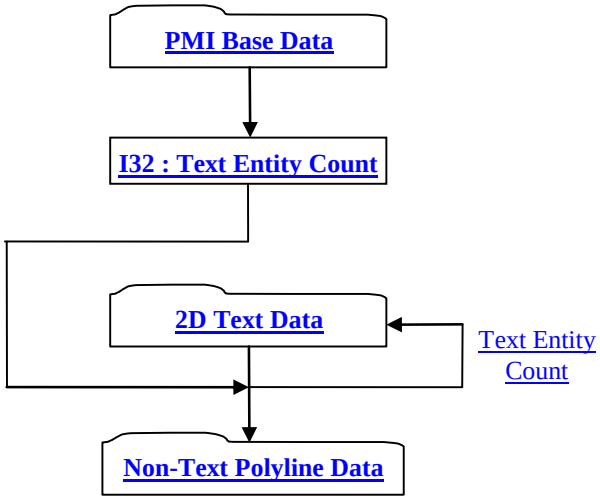
I32 : Dimension Count

Dimension Count specifies the number of Dimension entities.

7.2.6.2.1.1.1 PMI 2D Data

The PMI 2D Data collection defines a data format common to all 2D based PMI entities.

Figure 139: PMI 2D Data collection



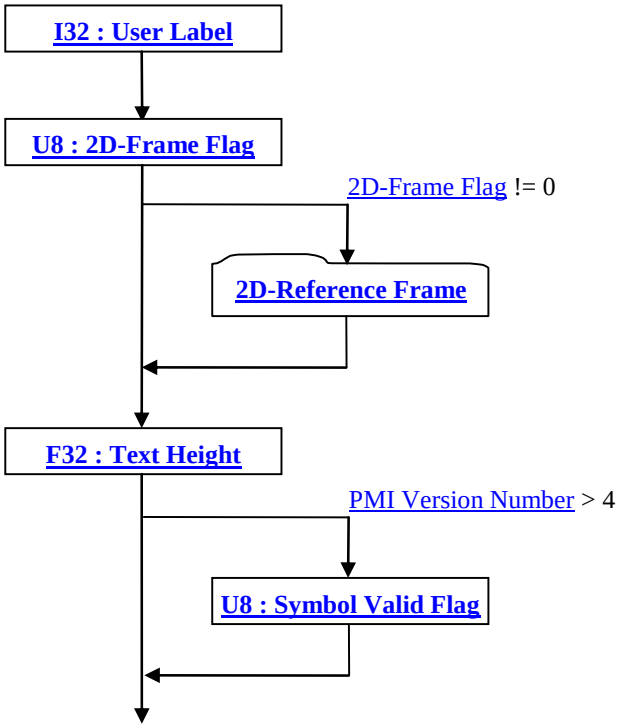
I32 : Text Entity Count

Text Entity Count specifies the number of Text entities in the particular PMI entity.

7.2.6.2.1.1.1.1 PMI Base Data

The PMI Base Data collection defines the basic/common data that every 2D and 3D PMI entity contains

Figure 140: PMI Base Data collection



I32 : User Label

User Label specifies the particular PMI entity identifier.

U8 : 2D-Frame Flag

2D-Frame Flag is a flag specifying whether 7.2.6.2.1.1.1.1.1 2D-Reference Frame data is stored. If 2D-Frame Flag has a non-zero value then 2D-Reference Frame data is included. If 2D-
2D-Reference Frame -Frame Flag = = case is used by 7.2.6.2.6 Generic PMI Entities because for Generic PMI Entities all the 7.2.6.2.1.1.1.3 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.

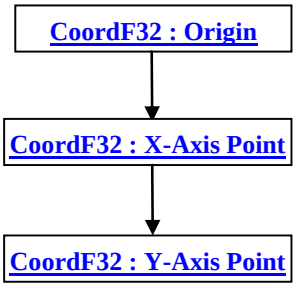
U8 : Symbol Valid Flag

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 7.2.6.2PMI Manager Meta Data Element

7.2.6.2.1.1.1.1.1 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. yline data is assumed to lie on the defined plane.

Figure 141: 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.

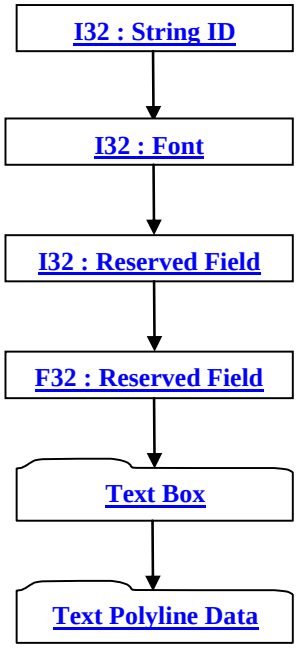
CoordF32 : Y-Axis Point

Y-Axis Point defines a point along the Y-Axis of the 2D coordinate system.

7.2.6.2.1.1.1.2 2D Text Data

The 2D Text Data collection defines a 2D text entity/primitive.

Figure 142: 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 7.2.6.2.4 PMI String Table. An identifier value -

I32 : Font

Font identifies the font to be used for this text. Valid values include the following:

= 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 : Reserved Field

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

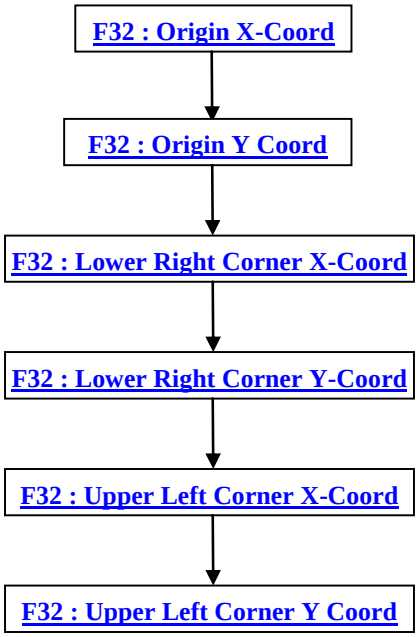
F32 : Reserved Field

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

7.2.6.2.1.1.1.2.1 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 7.2.6.2.1.1.1.1.1 2D-Reference Frame.

Figure 143: 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.

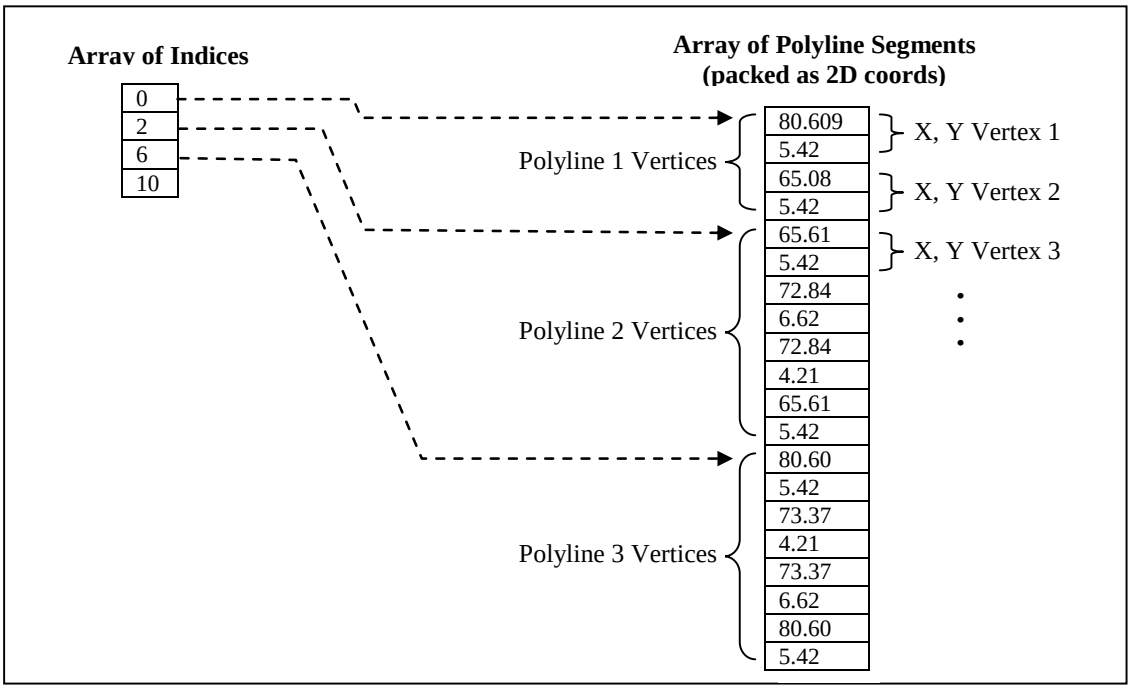
F32 : Upper Left Corner Y Coord

Upper Left Corner Y-Coord defines the 2D Y-coordinate of the upper left corner of the text with respect to 2D-Reference Frame.

7.2.6.2.1.1.1.2.2 Text Polyline Data

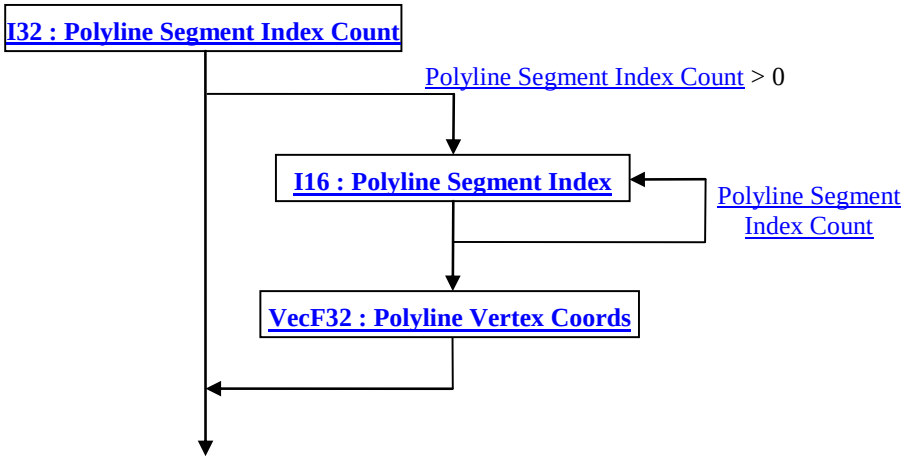
The Text Polyline Data collection defines any polyline segments which are part of the text representation. This existence of this polyline data is conditional (i.e. not all text has it) and is made up of an array of indices into an array of polyline segments packed as 2D vertex coordinates, specifying where each polyline segment begins and ends. Polylines are constructed from these arrays of data as follows:

Figure 144: Constructing Text Polylines from data arrays



This data is represented in JT file in the following format:

Figure 145: Text Polyline Data collection

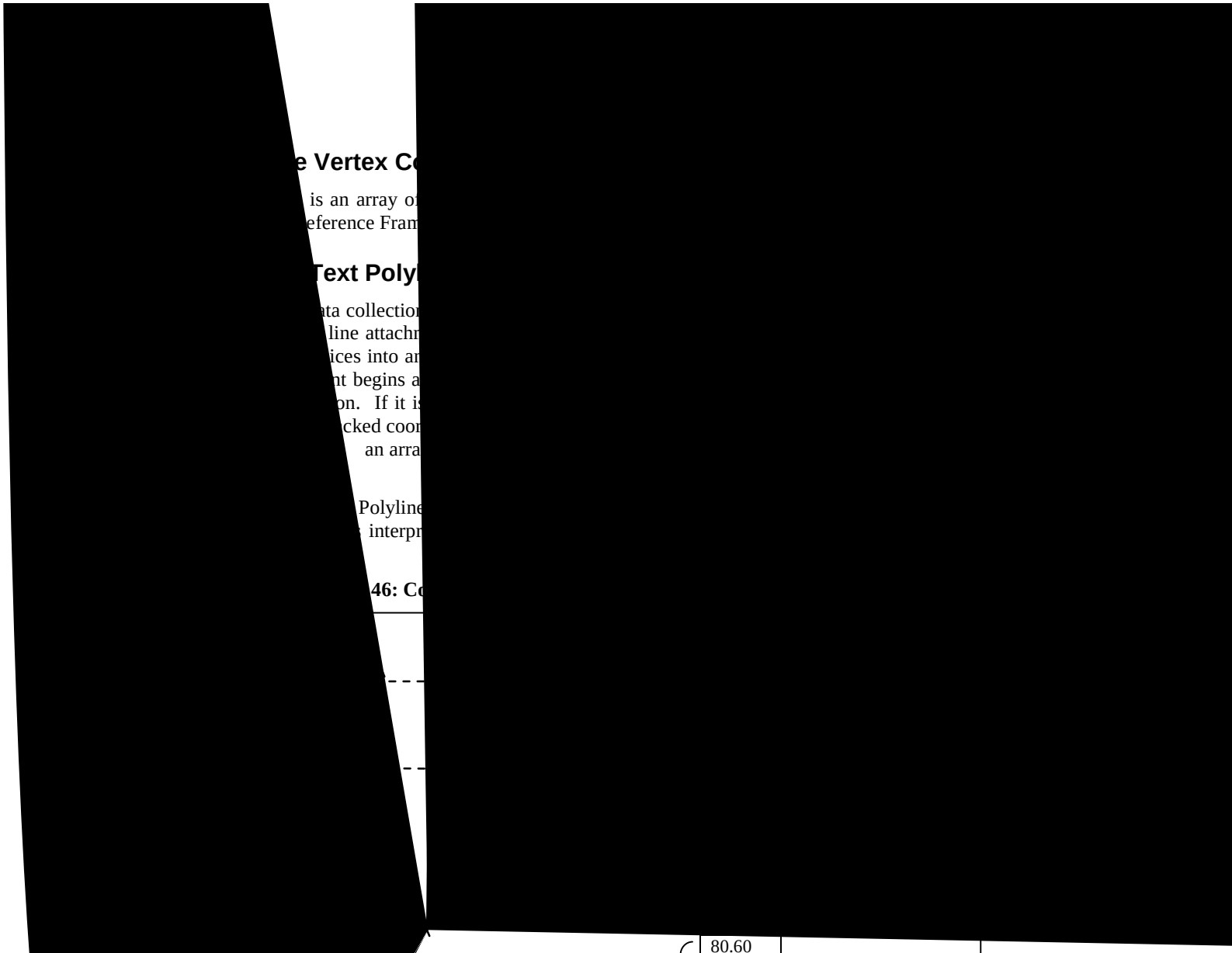


I32 : Polyline Segment Index Count

Polyline Segment Index Count specifies the number of polyline segment indices.

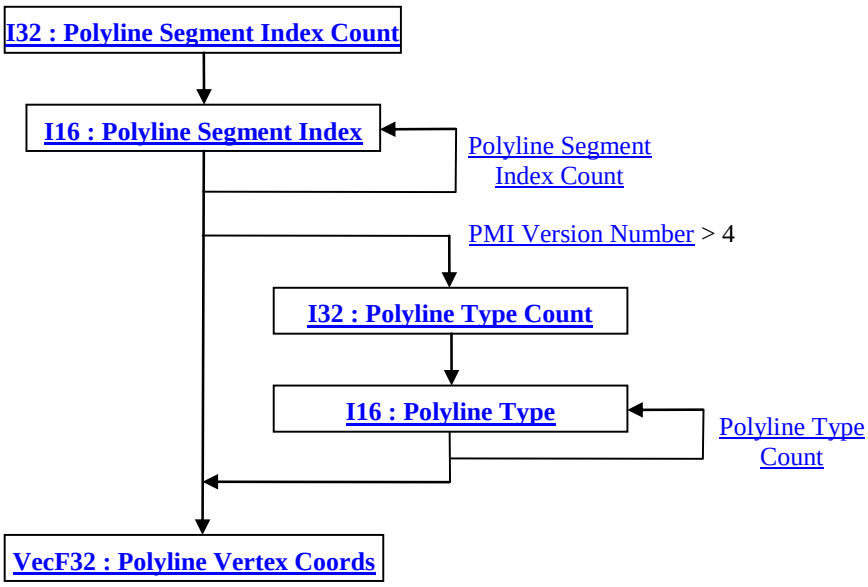
I16 : Polyline Segment Index

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



Polyline 3 Vertices	80.60
	5.42
	73.37
	4.21
	73.37
	6.62
	80.60
	5.42

Figure 147: 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 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 146: 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:

= 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

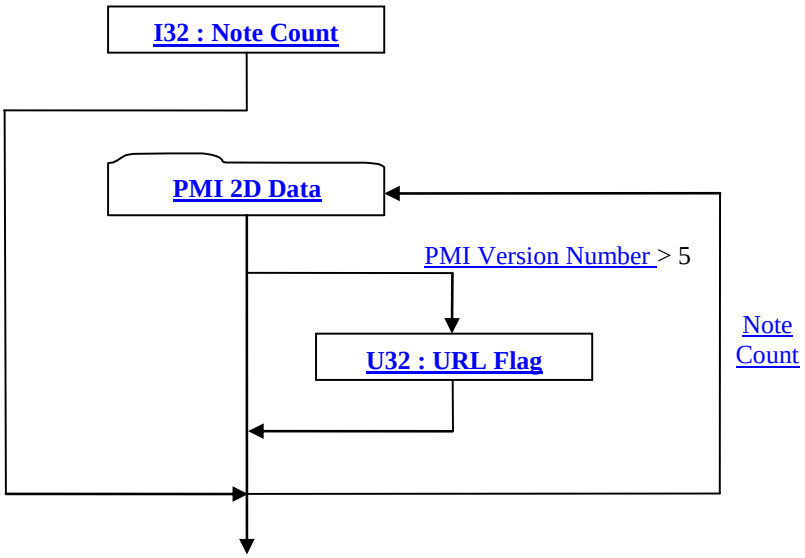
VecF32 : Polyline Vertex Coords

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 7.2.6.2.1.1.1.1.1 2D-Reference Frame.

7.2.6.2.1.2 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 148: PMI Note Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

I32 : Note Count

Note Count specifies the number of Note entities.

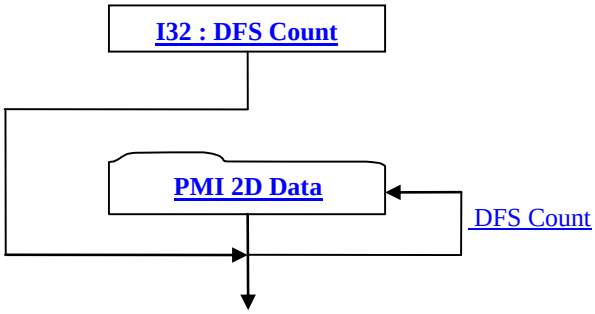
U32 : URL Flag

URL Flag specifies whether Note is an URL. This data field is only present if Version Number, as defined in 7.2.6.2 PMI Manager Meta Data Element The URL is the actual text of the note as specified in PMI 2D Data.

7.2.6.2.1.3 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
referenced by a Feature Control Frame.

Figure 149: PMI Datum Feature Symbol Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

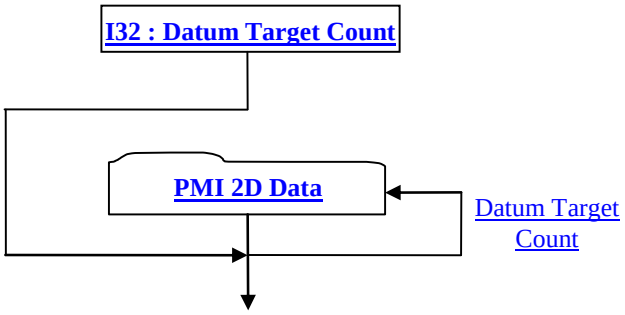
I32 : DFS Count

DFS Count specifies the number of Datum Feature Symbol entities.

7.2.6.2.1.4 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 manufacturing and inspection operations.

Figure 150: PMI Datum Target Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

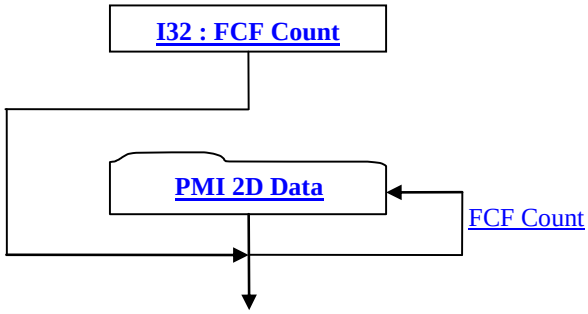
I32 : Datum Target Count

Datum Target Count specifies the number of Datum Target entities.

7.2.6.2.1.5 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.

Figure 151: PMI Feature Control Frame Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

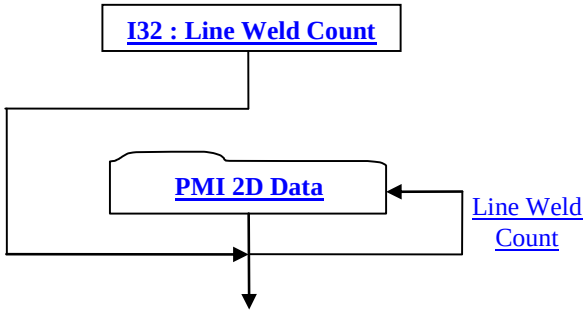
I32 : FCF Count

FCF Count specifies the number of Feature Control Frame entities.

7.2.6.2.1.6 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 152: PMI Line Weld Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

I32 : Line Weld Count

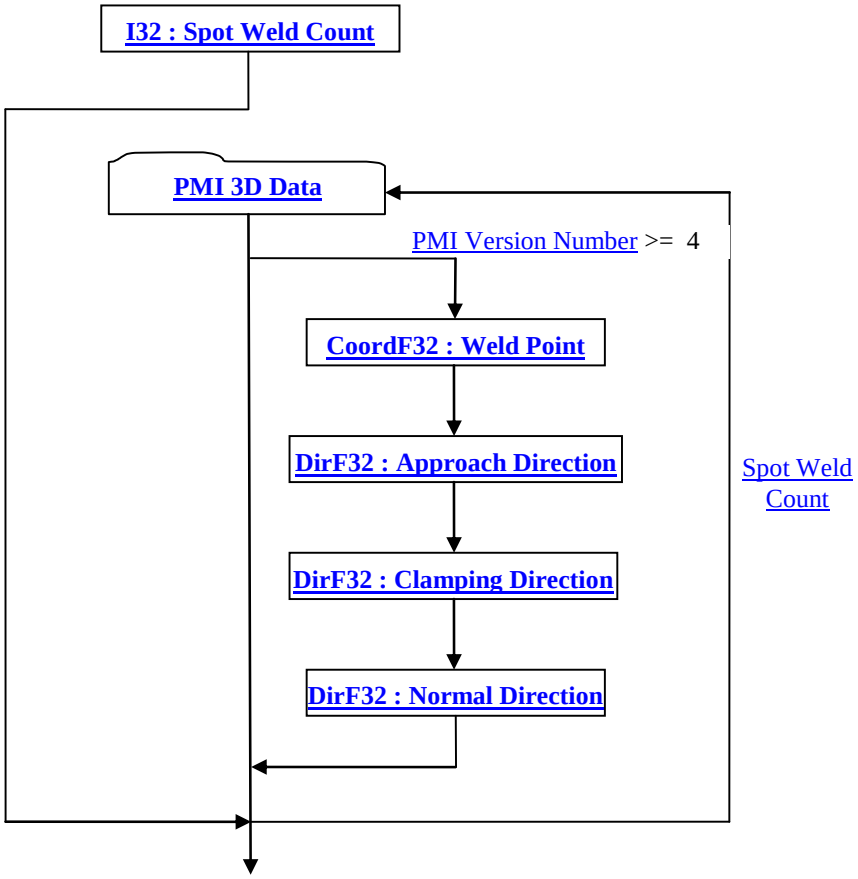
Line Weld Count specifies the number of Line Weld entities.

7.2.6.2.1.7 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 7.2.6.2PMI Manager Meta Data Element

Figure 153: 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.

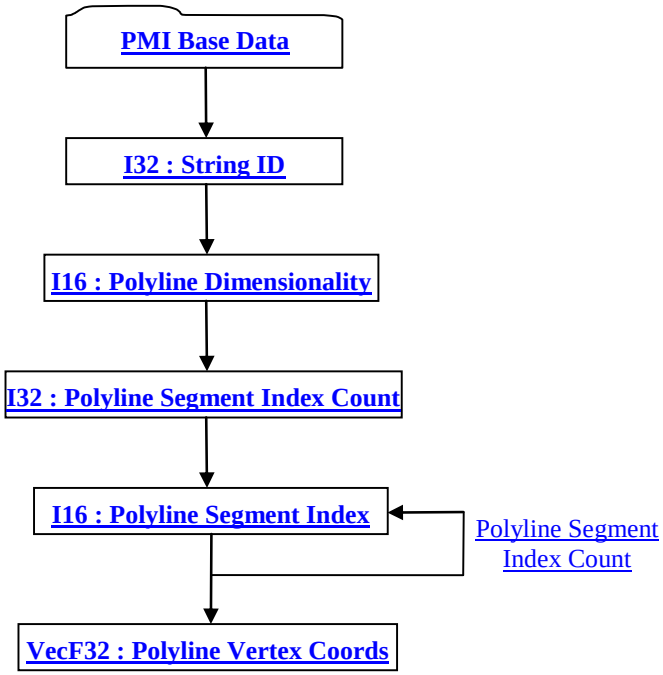
7.2.6.2.1.7.1 PMI 3D Data

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, 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 144 of 7.2.6.2.1.1.1.2.2 Text Polyline Data.

Figure 154: PMI 3D Data collection



Complete description for PMI Base Data can be found in 7.2.6.2.1.1.1.1 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 7.2.6.2.4 PMI String Table. An identifier -

I16 : Polyline Dimensionality

Polyline Dimensionality specifies the dimensionality of the polyline coordinates packed in Polyline Vertex Coords. Valid values include the following:

= 2	Indicates 2-
= 3	Indicates 3-

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.

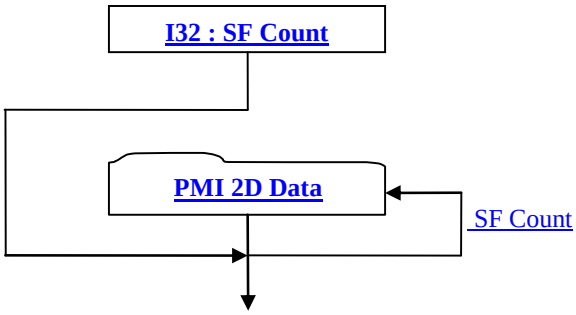
VecF32 : Polyline Vertex Coords

Polyline Vertex Coords is an array of polyline segments packed as Polyline Dimensionality point coordinates.

7.2.6.2.1.8 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 155: PMI Surface Finish Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

I32 : SF Count

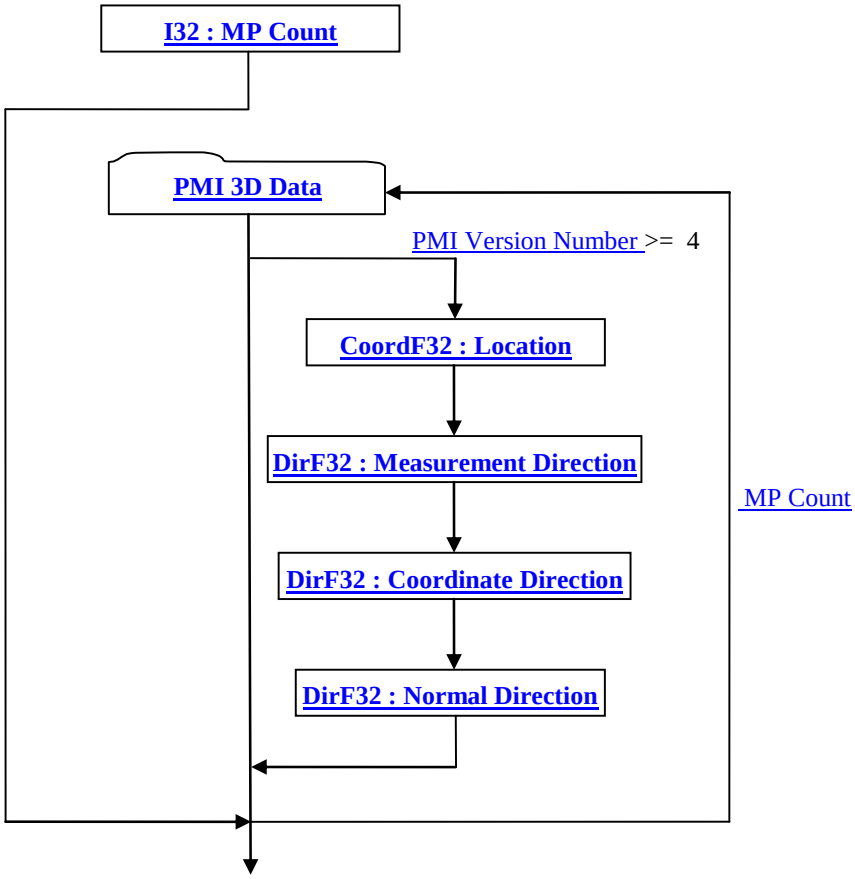
SF Count specifies the number of Surface Finish symbol entities.

7.2.6.2.1.9 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 7.2.6.2PMI Manager Meta Data Element

Figure 156: PMI Measurement Point Entities data collection



Complete description for PMI 3D Data can be found in 7.2.6.2.1.7.1 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.

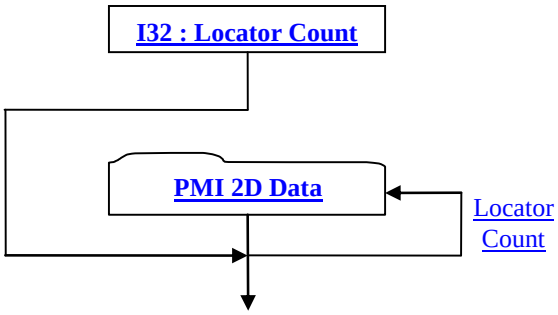
DirF32 : Normal Direction

Normal Direction specifies the components of the direction vector normal to the actual Measurement Point.

7.2.6.2.1.10 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.

Figure 157: PMI Locator Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 PMI 2D Data.

I32 : Locator Count

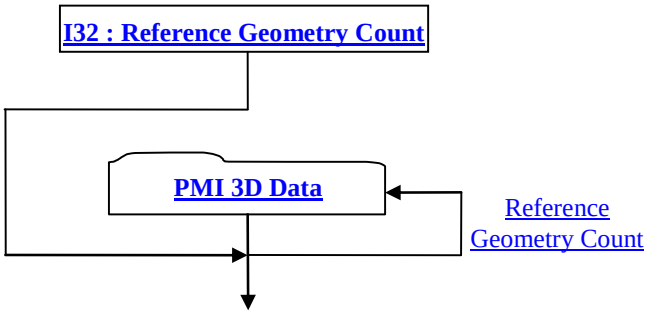
Locator Count specifies the number of Locator symbol entities.

7.2.6.2.1.11 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 7.2.6.2.1.7.1 PMI 3D Data) as follows:

Polyline Segment Index[1]	Implied Reference Geometry Type
= = 1	Point
= = 2	Polyline
> 2	Polygon

Figure 158: PMI Reference Geometry Entities data collection



Complete description for PMI 3D Data can be found in 7.2.6.2.1.7.1 PMI 3D Data.

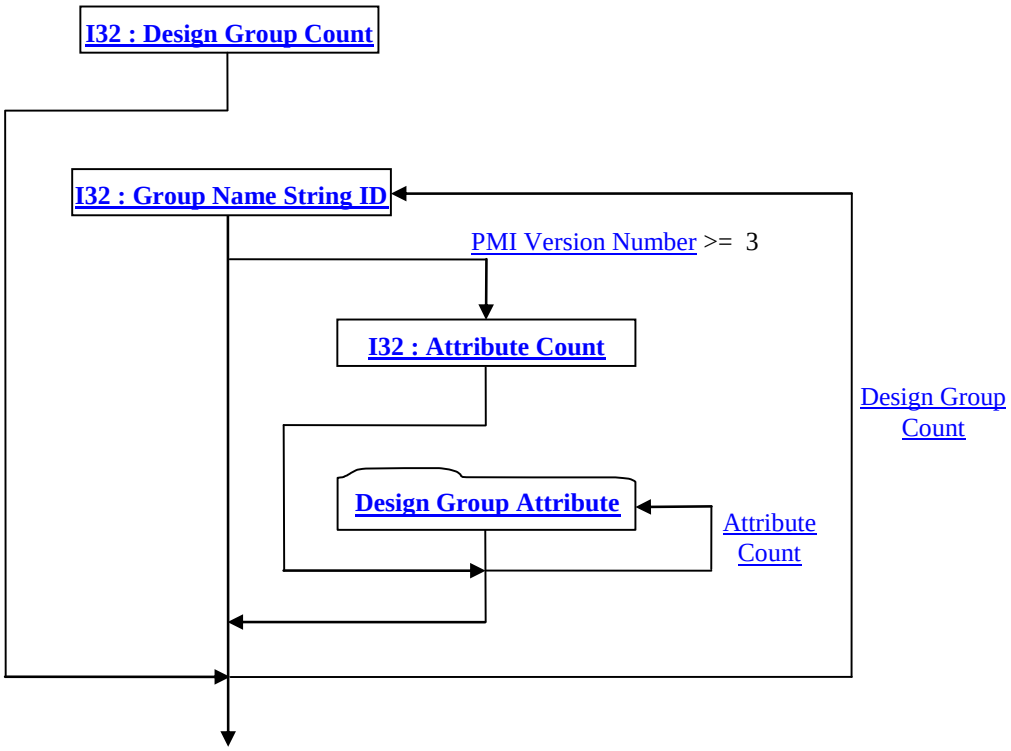
I32 : Reference Geometry Count

Reference Geometry Count specifies the number of Reference Geometry entities.

7.2.6.2.1.12 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 achieved via PMI Associations (see 7.2.6.2.2 PMI Associations
Group.

Figure 159: 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 7.2.6.2.4 PMI String Table. An identifier -
string.

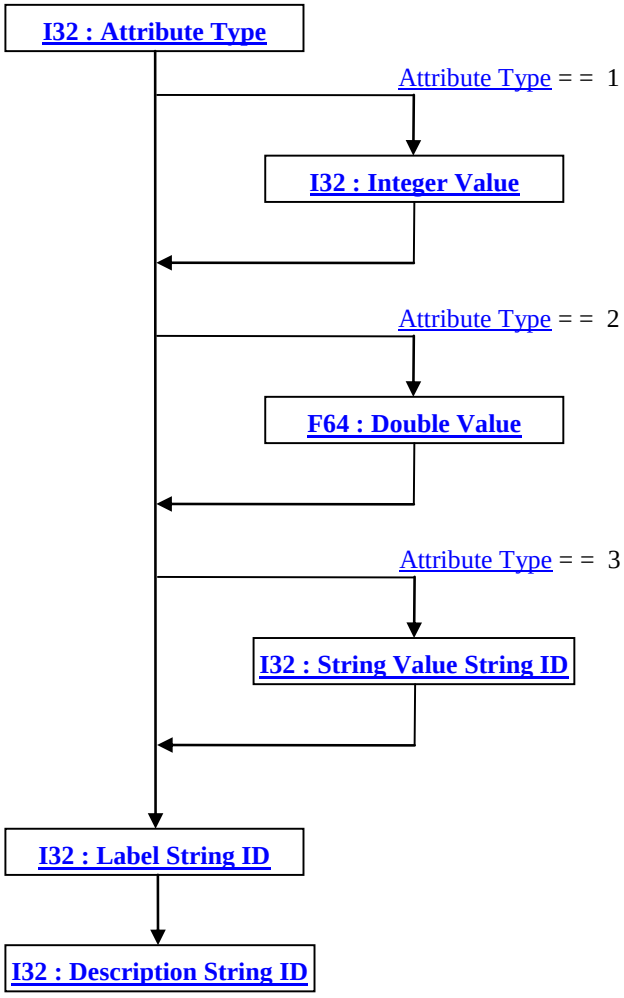
I32 : Attribute Count

Attribute Count specifies the number of Design Group Attribute data collections

7.2.6.2.1.12.1 Design Group Attribute

The Design Group Attribute data collection defines a group property/attribute.

Figure 160: Design Group Attribute data collection



I32 : Attribute Type

Attribute Type specifies the attribute type. Valid types include the following:

= 1	Integer
= 2	Double
= 3	String

I32 : Integer Value

i Types.

F64 : Double Value

I32 : String Value String ID

This identifier is an index to a particular character string in the PMI String Table as defined in 7.2.6.2.4 PMI String Table. An identifier - indicates no string.

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 7.2.6.2.4 PMI String Table. An identifier -

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 7.2.6.2.4 PMI String Table. An identifier - string.

7.2.6.2.1.13 PMI Coordinate S

CoordF32 : X-Axis Point

X-Axis Point defines a point along the X-Axis of the coordinate system.

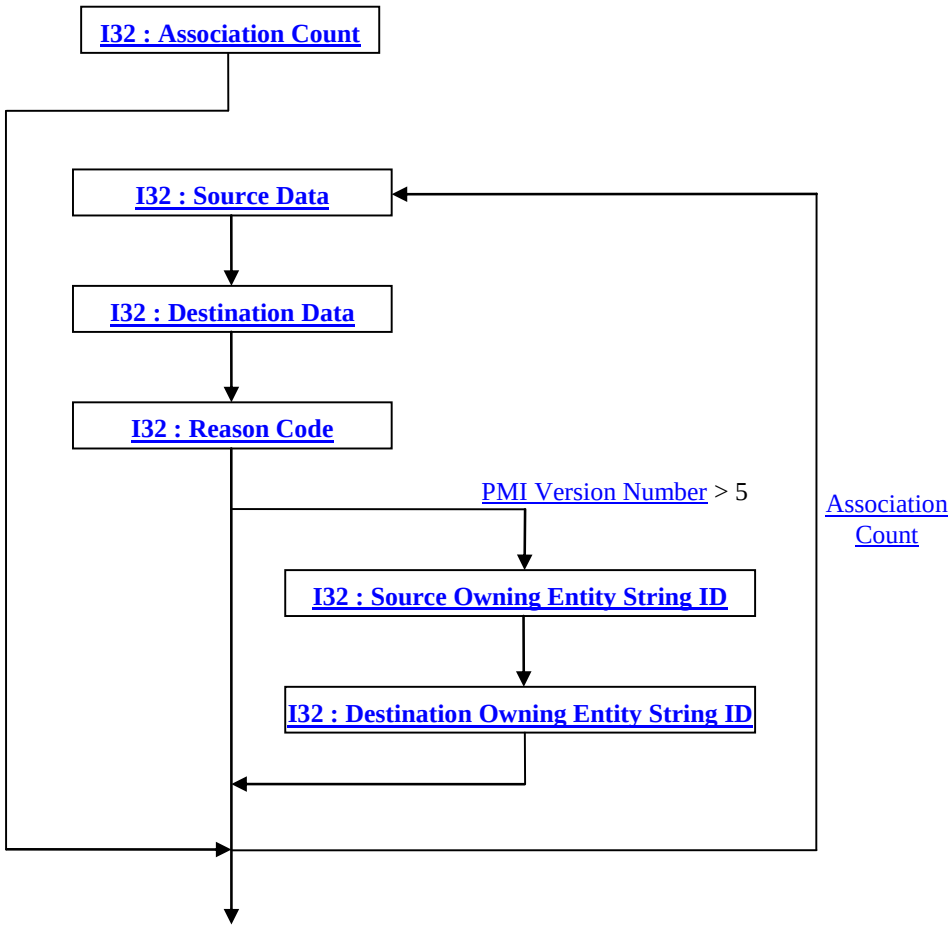
CoordF32 : Y-Axis Point

Y-Axis Point defines a point along the Y-Axis of the coordinate system.

7.2.6.2.2 PMI Associations

The PMI Associations data collection defines data for a list of associations. relationship between two PMI, B-Rep, or Wireframe Rep entity is

Figure 162: 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 undocumented bits are reserved.

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 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 7.2.6.2.7 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 spe

= 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	
= 13	
= 14	Specifying PMI grouping, source is PMI/B-Rep entity and destination is design group.
= 15	Association is to a weld line entity
= 16	
= 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. 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 7.2.6.2.4 PMI String Table. An identifier va - implies that the entity is to be 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 7.2.6.2 PMI Manager Meta Data Element

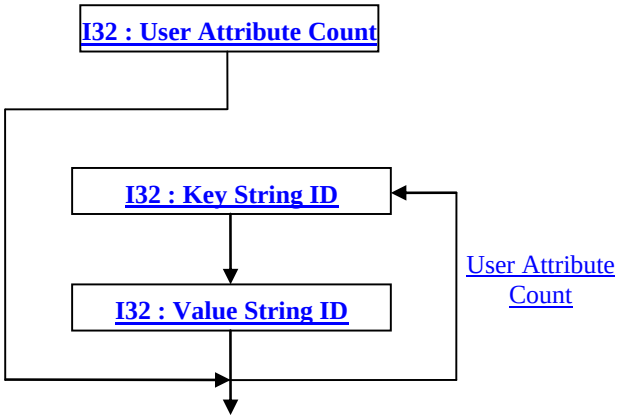
I32 : Destination Owning Entity String ID

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 7.2.6.2.4 PMI String Table. An identifier - string and implies that the entity is to be found on the current 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 7.2.6.2 PMI Manager Meta Data Element

7.2.6.2.3 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.

Figure 163: 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 7.2.6.2.4 PMI String Table. An identifier -

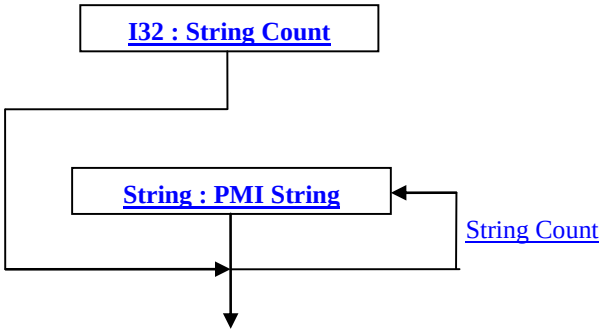
I32 : Value String ID

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 7.2.6.2.4 PMI String Table. An identifier -

7.2.6.2.4 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 (i.e. String ID).

Figure 164: PMI String Table data collection



I32 : String Count

String Count specifies the number of character strings in the string table.

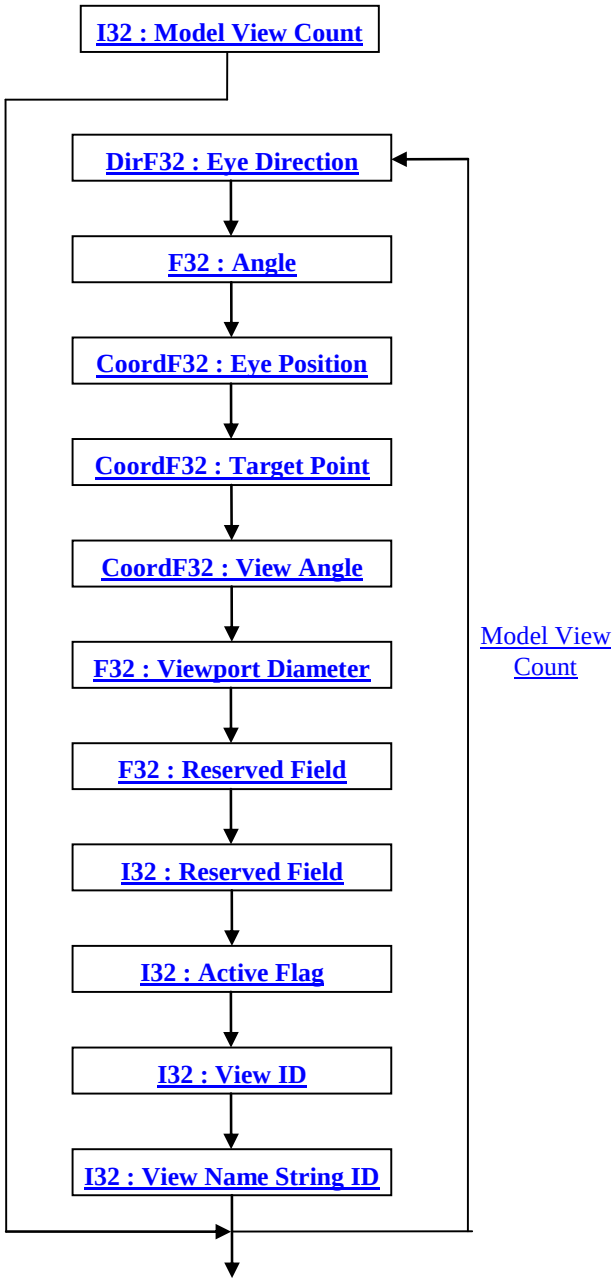
String : PMI String

PMI String specifies the character string.

7.2.6.2.5 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 7.2.6.2.2 PMI Associations)
View.

Figure 165: PMI Model Views data collection



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 position.

CoordF32 : Target Point

Target Point specifies

CoordF32 : View Angle

View angle specifies the X, Y, Z rotation angles (in degrees) of the model axis. The rotations are defined with respect to an initial orientation where s axis are aligned
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- - .

F32 : Reserved Field

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

I32 : Reserved Field

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

I32 : Active Flag

Active Flag is a flag specifying whether this

= 0	Is not the active Model View.
= 1	Is the active Model View

I32 : View ID

View ID specifies the Model View unique identifier.

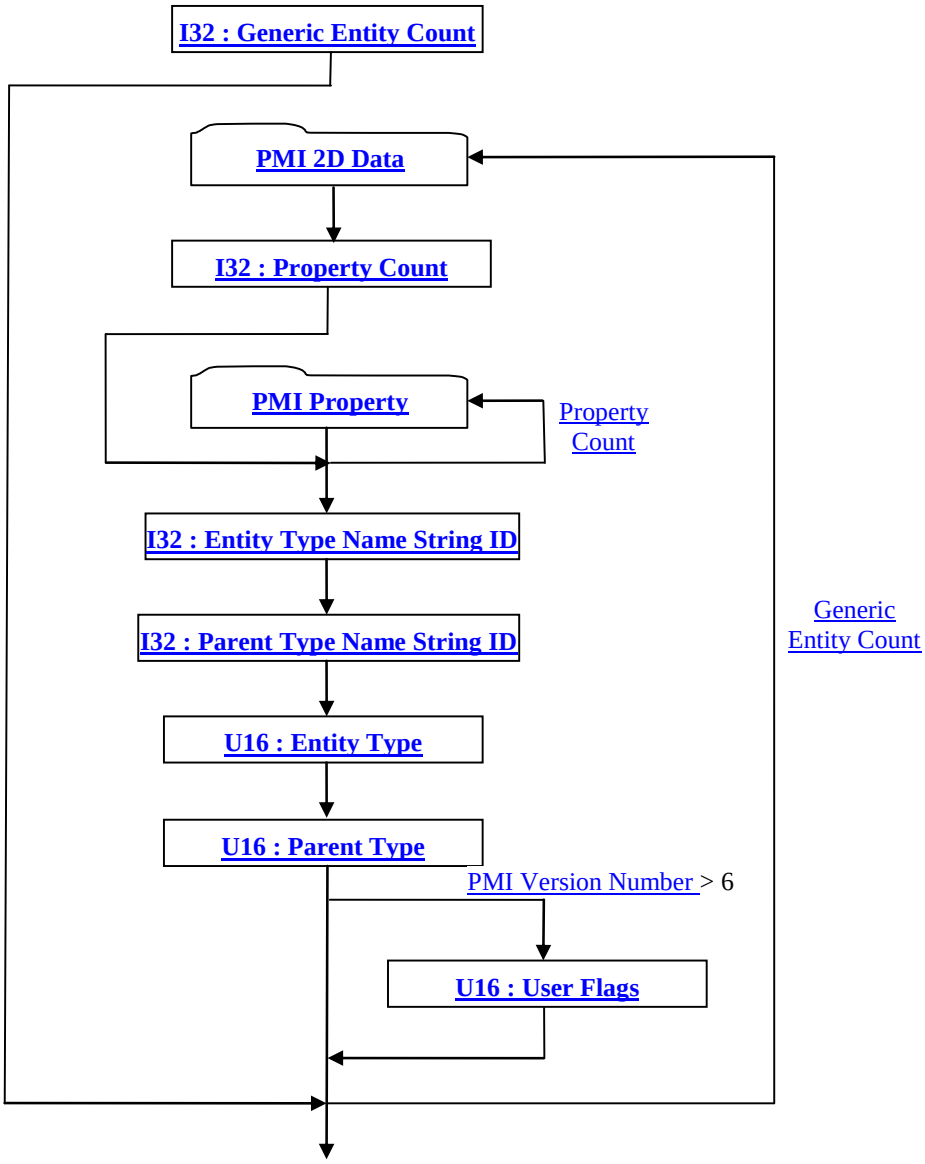
I32 : View Name String ID

View Name String ID specifie
character string in the PMI String Table as defined in 7.2.6.2.4 PMI String Table. An identifier - ndicates no string.

7.2.6.2.6 Generic PMI Entities

The Generic PMI Entities 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.

Figure 166: Generic PMI Entities data collection



Complete description for PMI 2D Data can be found in 7.2.6.2.1.1.1 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.

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 7.2.6.2.4 PMI String Table. An identifier value of -

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 7.2.6.2.4 PMI String Table. An identifier -

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
table below) should be used.

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
0x0121	Surface Vector Measurement Point
0x0122	Hole Vector Measurement Point
0x0124	Trimmed Sheet Vector Measurement Point
0x0128	Hem Vector Measurement Point

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

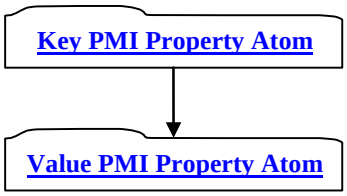
User Flags is a collection of flags. The flags are combined using the binary OR operator and store various state information for the Generic PMI Entity. All undocumented bits are reserved.

0x0001	Show PMI Entit = 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.

7.2.6.2.6.1 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 167: PMI Property data collection



Both Key PMI Property Atom and Value PMI Property Atom have the same format as that documented in 7.2.6.2.6.1.1 PMI Property Atom.

Although there is no reference compliant requirements for what the PMI Property key/value pairs must 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 7.2.6.2.6.1.1 PMI Property Atom for an explanation of what is meant by encoded string value).

Table 7: Common Property Keys and Their Value Encoding formats

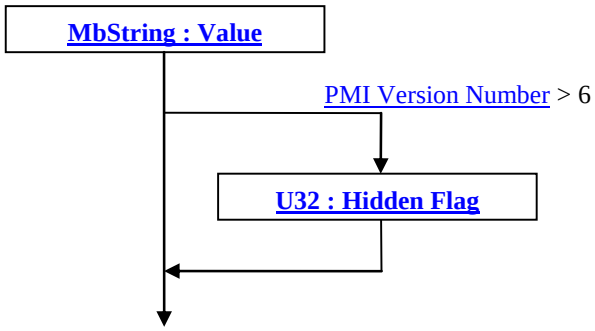
³.H\´ Property Atom Value String	³9D0XH´ Property Atom Value String Encoding Format	Decoding Notes
PMI_PROP_ANCHOR_POINT"		Each Px, Py, Pz
PMI_PROP_NOTE_HAS_URL	0 1	0 = = False; 1 = = True
PMI_PROP_NORMAL_DIR	Dx Dy Dz	Each Dx, Dy, Dz is a
PMI_PROP_APPROACH_DIR	Dx Dy Dz	Each Dx, Dy, Dz is a
PMI_PROP_CLAMPING_DIR	Dx Dy Dz	Each Dx, Dy, Dz is a
PMI_PROP_MEAS_DIR	Dx Dy Dz	Each Dx, Dy, Dz is a
PMI_PROP_COORD_DIR	Dx Dy Dz	Each Dx, Dy, Dz is a
PMI_PROP_MEAS_LEVEL		Integer representing level number
	#	Hexadecimal integer representing RGB color where -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 must be zero. The maximum value for a single byte is 0xFF (i.e. intensity value is in the range [0:255]).
	#	Same as
		Unsigned decimal integer representing opacity percentage. Actual opacity is: decoded# / 100.0
		Unsigned decimal integer: 0 = = False; 1 = = True
		Unsigned decimal integer representing text size in units of pixels.
PMITextInPlane		Unsigned decimal integer: 0 = = False; 1 = = True the plane of the entity so that it rotates with view.
PMIGeometryColor		Same as
PMIGeometryWidth		Unsigned decimal integer representing line width in units of pixels.
CLIP_NORMAL		Used for Entity Type Entity Type Name String 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 .
CLIP_POSITION		Used for Entity Type Entity Type Name String one point on the clipping plane. Each format.

³.H\´ Property Atom Value String	³9D0XH´ Property Atom Value String Encoding Format	Decoding Notes
TRANSFORMATION_MATRIX		Used for Entity Type Name String Entity Type specify a transformation matrix. Each # is a F32 value using

7.2.6.2.6.1.1 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 168: PMI Property Atom data collection



MbString : Value

Value specifies the property atom value encoded into a String. See Table 7: Common Property Keys and Their Value Encoding formats above for encoding formats of the Value string.

U32 : Hidden Flag

Hidden Flag specifies 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:

= 0	Property is not hidden.
= 1	Property is hidden.

7.2.6.2.7 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 7.2.6.2 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 169: PMI CAD Tag Data collection

Complete description for Compressed CAD Tag Data can be found in 8.1.16 Compressed CAD Tag Data.

I32 : CAD Tag Index Count

CAD Tag Index Count specifies the total number of CAD Tag indices. This value must 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
- 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.)

Figure 170: PMI Polygon Data