Designation: E2807 – 11

Standard Specification for 3D Imaging Data Exchange, Version 1.0[1]

This standard is issued under the fixed designation E2807; the number immediately following the designation indicates the year of original adoption or, in the case of revision, the year of last revision. A number in parentheses indicates the year of last reapproval. A superscript epsilon (ε) indicates an editorial change since the last revision or reapproval.

1. Scope

1.1 This specification describes a data file exchange format for three-dimensional (3D) imaging data, known as the ASTM E57 3D file format, Version 1.0. The term "E57 file" will be used as shorthand for "ASTM E57 3D file format" hereafter.

1.2 An E57 file is capable of storing 3D point data, such as that produced by a 3D imaging system, attributes associated with 3D point data, such as color or intensity, and 2D imagery, such as digital photographs obtained by a 3D imaging system. Furthermore, the standard defines an extension mechanism to address future aspects of 3D imaging.

1.3 This specification describes all data that will be stored in the file. The file is a combination of binary and eXtensible Markup Language (XML) formats and is fully documented in this specification.

1.4 All quantities standardized in this specification are expressed in terms of SI units. No other units of measurement are included in this standard.

1.4.1 *Discussion*—Planar angles are specified in radians, which are considered a supplementary SI unit.

1.5 *This standard does not purport to address all of the safety concerns, if any, associated with its use. It is the responsibility of the user of this standard to establish appropriate safety and health practices and determine the applicability of regulatory limitations prior to use.*

1.6 *This standard does not purport to address legal concerns, if any, associated with its use. It is the responsibility of the user of this standard to comply with appropriate regulatory limitations prior to use.*

2. Referenced Documents

2.1 *ASTM Standards:*[2]

E2544 Terminology for Three-Dimensional (3D) Imaging Systems

2.2 *IEEE Standard:*[3]

754-1985 IEEE Standard for Binary Floating-Point Arithmetic

2.3 *IETF Standard:*[4]

RFC 3720 Internet Small Computer Systems Interface (iSCSI)

2.4 *W3C Standard:*[5]

XML Schema Part 2: Datatypes Second Edition

3. Terminology

3.1 *Definitions*—Terminology used in this specification conforms to the definitions included in Terminology E2544.

3.2 *Definitions of Terms Specific to This Standard:*

3.2.1 *backward compatibility, n*—ability of a file reader to understand a file created by a writer of an older version of a file format standard.

3.2.2 *byte, n*—grouping of 8 bits, also known as an octet.

3.2.3 *camel case, n*—naming convention in which compound words are joined without spaces with each word's initial letter capitalized within the component and the first letter is either upper or lowercase.

3.2.4 *camera image, n*—regular, rectangular grid of values that stores data from a 2D imaging system, such as a camera.

3.2.5 *camera projection model, n*—mathematical formula used to convert between 3D coordinates and pixels in a camera image.

3.2.6 *file offset, n*—see *physical file offset*.

3.2.7 *file-level coordinate system, n*—coordinate system common to all 2D and 3D data sets in a given E57 file.

3.2.8 *forward compatibility, n*—ability of a file reader to read a file that conforms to a newer version of a format specification than it was designed to read, specifically having the capability to understand those aspects of the file that were defined in the version it was designed to read, while ignoring those portions that were defined in later versions of the format specification.

[1] This specification is under the jurisdiction of ASTM Committee E57 on 3D Imaging Systems and is the direct responsibility of Subcommittee E57.04 on Data Interoperability.

Current edition approved Feb. 1, 2011. Published March 2011.

[2] For referenced ASTM standards, visit the ASTM website, www.astm.org, or contact ASTM Customer Service at service@astm.org. For *Annual Book of ASTM Standards* volume information, refer to the standard's Document Summary page on the ASTM website.

[3] For referenced IEEE standards, visit http://grouper.ieee.org/groups/754.

[4] For referenced Internet Engineering Task Force (IETF) standards, visit the IETF website, www.ietf.org.

[5] String representations (the lexical space) of the numeric datatypes are documented in the W3C standard: "XML Schema Part 2: Datatypes Second Edition", available on the website http://www.w3.org/TR/xmlschema-2/.

3.2.9 *logical length, n*—number of bytes used to describe some entity in an E57 file, not including CRC checksum bytes.

3.2.10 *physical file offset, n*—number of bytes preceding the specified byte location in an E57 file, counting payload bytes and checksums.

3.2.10.1 *Discussion*—This term is also known as the *file offset*.

3.2.11 *physical length, n*—number of bytes used to describe some entity in an E57 file, including CRC checksum bytes.

3.2.12 *record, n*—single collection in a sequence of identically-typed collections of elements.

3.2.13 *rigid body transform, n*—type of coordinate transform that preserves distances between all pairs of points that furthermore does not admit a reflection.

3.2.13.1 *Discussion*—A rigid body transform can be used, for example, to convert points from the local coordinates of a 3D data set (for example, a single laser scan) to a common coordinate system shared by multiple 3D data sets (for example, a set of laser scans).

3.2.14 *XML namespace, n*—method for qualifying element names in XML to prevent the ambiguity of multiple elements with the same name.

3.2.14.1 *Discussion*—XML namespaces are used in an E57 file to support the definition of extensions.

3.2.15 *XML whitespace, n*—sequence of one or more of the following Unicode characters: the space character (20 hexadecimal), the carriage return (0D hexadecimal), line feed (0A hexadecimal), or tab (09 hexadecimal).

3.2.16 *zero padding, n*—one or more zero-valued bytes appended to the end of a sequence of bytes.

4. Acronyms

4.1 *ASCII*—American Standard Code for Information Interchange

4.2 *CRC*—Cyclic redundancy check

4.3 *GUID*—Globally unique identifier

4.4 *IEEE*—Institute of Electrical and Electronics Engineers

4.5 *IETF*—Internet Engineering Task Force

4.6 *iSCSI*—Internet small computer system interface

4.7 *JPEG*—Joint Photographic Experts Group

4.8 *PNG*—Portable network graphics

4.9 *URI*—Uniform resource identifier

4.10 *UTC*—Coordinated universal time

4.11 *UTF*—Unicode Transformation Format

4.12 *W3C*—WorldWide Web Consortium

4.13 *XML*—eXtensible Markup Language

5. Notation and Mathematical Concepts

5.1 The following notation and established mathematical concepts are used in this specification.

5.2 *Intervals:*

5.2.1 A closed interval is denoted $[a, b]$, where $a \leq b$. A closed interval includes the endpoints a and b and all numbers in between. An open interval is denoted (a, b), where $a \leq b$. An open interval includes the numbers between the endpoints a and b, but does not include the endpoints themselves. The half-open intervals $(a, b]$ and $[a, b)$ do not include the a and b endpoints, respectively.

5.3 *Cartesian Coordinate System:*

5.3.1 Points in Cartesian coordinates are represented by an ordered triplet (x, y, z), where x, y and z are coordinates along the X, Y, and Z axes, respectively. The coordinate system is right-handed.

5.4 *Cylindrical Coordinate System:*

5.4.1 Points in cylindrical coordinates are represented by an ordered triplet (ρ, θ, z), where ρ is the radial distance (in meters), θ is the azimuth angle (in radians), and z is the height (in meters).

5.4.1.1 The azimuth angle is measured as the counterclockwise rotation of the positive X-axis about the positive Z-axis of a Cartesian reference frame.

5.4.2 The following restrictions on cylindrical coordinates are applied:

$$\rho \geq 0 \tag{1}$$

$$-\pi < \theta \leq \pi \tag{2}$$

5.4.3 The conversion from Cartesian to cylindrical coordinates is accomplished through the formulas (note that the z coordinate is the same in both systems):

$$\rho = \sqrt{(x^2 + y^2)} \tag{3}$$

$$\theta = \text{atan2}(y, x) \tag{4}$$

5.4.3.1 The function "atan2(y, x)" is defined as the function returning the arc tangent of y/x, in the range $(-\pi, +\pi]$ radians. The signs of the arguments are used to determine the quadrant of the result.

5.4.3.2 In degenerate cases, the following convention is observed:

If $x = y = 0$, then $\theta = 0$.

5.4.4 Conversely, cylindrical coordinates can be converted to Cartesian coordinates using the formulas:

$$x = \rho \cos(\theta) \tag{5}$$

$$y = \rho \sin(\theta) \tag{6}$$

5.5 *Spherical Coordinate System:*

5.5.1 Points in spherical coordinates are represented by an ordered triplet (r, θ, φ), where r is the range (in meters), θ is the azimuth angle (in radians), and φ is the elevation angle (in radians).

5.5.2 The following restrictions on spherical coordinates are applied:

$$r \geq 0 \tag{7}$$

$$-\pi < \theta \leq \pi \tag{8}$$

$$-\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2} \tag{9}$$

5.5.3 The conversion from spherical to Cartesian coordinates is accomplished through the formulas:

$$x = r\ \cos(\varphi)\cos(\theta) \quad (10)$$

$$y = r\ \cos(\varphi)\sin(\theta) \quad (11)$$

$$z = r\ \sin(\varphi) \quad (12)$$

5.5.4 Conversely, in non-degenerate cases, Cartesian coordinates can be converted to spherical coordinates via the formulas:

$$r = \sqrt{(x^2+y^2+z^2)} \quad (13)$$

$$\theta = \text{atan2}(y,x) \quad (14)$$

$$\varphi = \arcsin\left(\frac{z}{r}\right) \quad (15)$$

5.5.4.1 In degenerate cases, the following conventions are observed:

If $x = y = 0$, then $\theta = 0$;

If $x = y = z = 0$, then both $\theta = 0$ and $\varphi = 0$.

5.5.5 *Discussion*—The elevation is measured with respect to the *XY*-plane, with positive elevations towards the positive *Z*-axis. The azimuth is measured as the counterclockwise rotation of the positive *X*-axis about the positive *Z*-axis. This definition of azimuth follows typical engineering usage. Note that this differs from traditional use in navigation or surveying.

5.6 *Quaternions:*

5.6.1 A quaternion is a generalized complex number. A quaternion, q, is represented by an ordered four-tuple (w,x,y,z), where $q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$. The coordinate w defines the scalar part of the quaternion, and the coordinates (x, y, z) define the vector part.

5.6.2 The norm of a quaternion, $\| q \|$, is defined as:

$$\| q \| = \sqrt{w^2+x^2+y^2+z^2}.$$

5.6.3 A unit quaternion, q, has the further restriction that its norm $\| q \| = 1$.

5.6.4 Rotation of a point $\boldsymbol{p}$ by a unit quaternion q is given by the matrix formula:

$$\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p} \quad (16)$$

where:

$$\boldsymbol{R} = \begin{bmatrix} w^2+x^2-y^2-z^2 & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & w^2+y^2-x^2-z^2 & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & w^2+z^2-x^2-y^2 \end{bmatrix} \quad (17)$$

5.6.5 *Discussion*—Unit quaternions are used in this standard to represent rotations in rigid body transforms.

5.7 *Rigid Body Transforms:*

5.7.1 A rigid body transform converts points from one coordinate reference frame to another, preserving distances between pairs of points and, furthermore, not admitting a reflection. A rigid body transform can be represented as a 3×3 rotation matrix $\boldsymbol{R}$ and a translation 3-vector $\boldsymbol{t}$.

5.7.2 A 3D point is transformed from the source coordinate system to the destination coordinate system by first applying the rotation and then the translation. More formally, the transformation operation $T(.)$ of a point $\boldsymbol{p}$ is defined as:

$$\boldsymbol{p}' = T(\boldsymbol{p}) = \boldsymbol{R}\boldsymbol{p}+\boldsymbol{t} \quad (18)$$

The rotation matrix $\boldsymbol{R}$ can be computed from a unit quaternion q using Eq 17.

5.7.3 *Discussion*—Rigid body transforms are used in this standard to support the transformation of data represented in a local coordinate system, such as the coordinate system of a sensor used to acquire a 3D data set, to a common file-level coordinate system shared by all 3D data sets.

5.8 *Trees:*

5.8.1 A tree is data structure that represents an acyclic graph. A tree consists of nodes, which store some information, and edges (also known as arcs) that connect the nodes. The single topmost node is called the root node. A node may have zero or more nodes connected below it, which are called child nodes. Each node, except the root node, has exactly one node connected above it, which is called the parent node. Nodes with no children are called leaf nodes. A descendant is a direct or indirect child of a given node.

5.8.2 *Discussion*—Trees are used in this standard to describe the structure of XML data, as well as index data in binary sections.

5.9 *XML Elements and Attributes :*

5.9.1 An XML element is the fundamental building block of an XML file. An element consists of a start tag, optional attributes, optional child elements, optional child text, and an end tag. Element names in an E57 file are case sensitive. Element names in this specification are written in camel case with a lowercase initial character. Type names in this specification are written in camel case with an upper case initial character.

5.9.2 *Discussion*—See Fig. 1 for an excerpt of XML that illustrates the parts of an XML element.

5.9.3 XML elements that have child elements form a tree, with each element being a node.

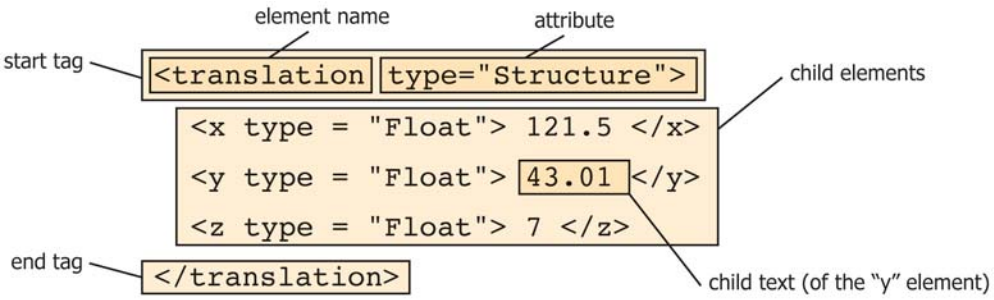


FIG. 1 XML Elements and Attributes

5.9.4 A pathname is a string that specifies the sequence of elements names that are encountered when traversing from a given origin element to a destination element in an XML tree. In this standard, pathnames are only defined for destination elements that are descendants of the origin element. A relative pathname is formed by concatenating the sequence of element names traversed using the forward slash ("/") as a separator. Each successive element in the sequence shall be a child of the previous element. Note that the element name of the origin element does not appear in the pathname. An absolute pathname has an origin that is the root element of the tree, and is formed by prepending a forward slash to the relative pathname.

5.9.5 *Discussion*—As an example, consider a hypothetical E57 file consisting of a root element named `e57Root` which contains a child element named `data3D`, which contains a child element named `0`, which contains a child element named `pose`, which contains a child element named `translation`, which contains a child element named `x`. Then the absolute pathname of the `x` element is `"/data3D/0/pose/translation/x"`, and the relative pathname of the `x` element relative to the `pose` element is `"translation/x"`.

6. General File Structure

6.1 E57 files shall use the filename extension ".e57" (note lowercase e).

6.2 This specification defines a binary file format composed of a sequence of pages.

6.2.1 Each page shall be composed of 1020 bytes of data (known as the payload) followed by a 32-bit cyclic redundancy check (CRC) checksum computed on the preceding payload.

6.2.2 The length of an E57 file shall be an integral multiple of 1024 bytes. Any unused bytes in the payload of the final page in a file shall be filled with 0 values.

6.2.3 The CRC checksum shall be computed on the 1020 bytes of data using the iSCSI polynomial CRC32C (CRC 32-bit Castagnioli) as documented in IETF RFC 3720, Section 12.1 (http://tools.ietf.org/html/rfc3720).

6.2.4 *Discussion*—Sequences of data without the CRC checksum bytes are known as logical sequences, while sequences of data with the CRC checksum bytes included are known as physical sequences. All sequences of characters (in XML section) or bytes (in binary sections) described in this standard are logical sequences. The physical sequence representation of a logical sequence may have an intervening checksum if the logical sequence crosses a page boundary. Page boundaries occur every 1020 bytes of logical data.

6.3 An E57 file shall be composed of two or more sections in the following order:

6.3.1 File header section (required, see Section 7),

6.3.2 Binary sections (optional, see Section 9), and

6.3.3 XML section (required, see Section 8).

6.4 Binary portions (including the header and binary sections) of an E57 file are encoded using the little-endian byte order.

7. File Header Section

7.1 The file header section begins at file offset 0.

7.2 The file header section is 48 bytes in length, with the format given in Table 1.

8. XML Section

8.1 The XML section of the file describes the data hierarchy. The data hierarchy contains a set of XML elements in a specific format, and arbitrary XML is not allowed. The elements are built upon a set of fundamental data types: Integer, ScaledInteger, Float, String, Structure, Blob, Vector, and CompressedVector. Additional composite data types are defined in the standard or can be defined by an extension to the standard.

8.2 The XML section of the E57 file contains a single well-formed XML 1.0 document using UTF-8 encoding. However, arbitrary XML is not allowed. The elements in the XML section shall be E57 elements, which are XML elements of a specific format, as will be described in 8.3. Furthermore, the elements shall follow particular grammatical rules, which are described in 8.4.

8.3 *E57 Element Data Types:*

8.3.1 The E57 file format supports eight fundamental E57 element data types—five terminal types and three non-terminal types. Non-terminal types are composed of other non-terminal or terminal types to an arbitrary, but finite, level of nesting. Terminal types shall not contain any child E57 elements. The terminal types are Integer, ScaledInteger, Float, String, and Blob. The non-terminal types are Structure, Vector, and CompressedVector. Every element in an E57 file shall be one of these types. Some or all of the data associated with an E57 element may be encoded in a binary section.

8.3.1.1 The data type of an E57 element is indicated by the `type` XML attribute, which is required. Depending on the data type, there may be other XML attributes that are required or optional, and there may be restrictions on child elements.

TABLE 1 Format of the E57 File Header Section

Bytes	Field name	Data type	Description
1-8	fileSignature	8-bit characters	The file type signature. Shall contain the ASCII characters "ASTM-E57".
9-12	versionMajor	Unsigned 32-bit integer	The file format major version number. The value shall be 1.
13-16	versionMinor	Unsigned 32-bit integer	The file format minor version number. The value shall be 0.
17-24	fileLength	Unsigned 64-bit integer	The physical length of the file, in bytes. Note that this length includes CRC bytes and any zero padding as described in 6.2.2. Shall be in the open interval (0, 2^{63}).
25-32	xmlOffset	Unsigned 64-bit integer	The physical file offset, in bytes, to the beginning of the XML section of the file. As defined in 3.2.10, this value includes CRC bytes. Shall be in the open interval (0, 2^{63}).
33-40	xmlLength	Unsigned 64-bit integer	The logical length, in bytes, of the XML section of the file, excluding CRC bytes and zero padding. Shall be in the open interval (0, 2^{63}).
41-48	pageSize	Unsigned 64-bit integer	The size a page, in bytes, as defined in 6.2. The value shall be 1024.

8.3.1.2 String representations of the numeric data types are documented in the W3C standard[6] "XML Schema Part 2: Data types Second Edition." The following XML built-in data types from that standard are referenced below: xsd:integer, xsd:float, xsd:double. Instances of xsd:float and xsd:double shall not use the special values: -0 (negative zero), +INF (positive infinity), -INF (negative infinity), and NaN (not a number).

8.3.1.3 The rules for each E57 element data type are detailed in following sections. The order of the XML attributes is not important.

8.3.2 *Integer Type:*

8.3.2.1 Integer type E57 elements (Integer hereafter) are used for storing integer values. The XML attributes for an Integer are listed in Table 2.

8.3.2.2 The value of an Integer is represented as child text of the XML element. This child text shall be zero or one occurrence of the xsd:integer representation, with optional leading and trailing XML whitespace. If no value is specified, the default value of the Integer is 0.

8.3.2.3 The value of the Integer is restricted to be in the range [`minimum`, `maximum`].

8.3.3 *ScaledInteger Type:*

8.3.3.1 For efficiency, it is possible to store numbers with fractional parts using a ScaledInteger type E57 element (ScaledInteger hereafter). A ScaledInteger stores an integer "raw value," and the actual floating point "scaled value" is computed from the raw value by applying a scaling and offset. The XML attributes for a ScaledInteger are listed in Table 3.

8.3.3.2 The rawValue of a ScaledInteger is encoded as child text of the XML element. The child text shall be zero or one occurrence of the xsd:integer representation, with optional leading and trailing XML whitespace. The raw value is restricted to be in the closed interval [`minimum`, `maximum`]. If raw value is unspecified, the default raw value is 0. The scaled value (*SV*) is computed from the raw value (*RV*) using the formula

$$SV = \mathrm{r}(\mathrm{r}\ (\mathrm{r}(RV) \times \mathrm{r}(\mathrm{scale}) + \mathrm{r}(\mathrm{offset})) \qquad (19)$$

where the r() function means rounding to the nearest representable double precision (53-bit mantissa) IEEE 754-1985 floating-point number using the "Round To Nearest" rounding mode (as described in IEEE 754-1985), and the '×' and '+' operators are considered to be infinitely precise.

8.3.4 *Float Type:*

8.3.4.1 Float type E57 elements (Float hereafter) are used for storing floating point values. The XML attributes for a Float are listed in Table 4.

8.3.4.2 The value of a Float is represented as child text of the XML element. This child text shall be zero or one occurrence of the xsd:float representation (if `precision` is single) or xsd:double (if `precision` is double), with optional leading and trailing XML whitespace. If no value is specified, the default value of the Float is 0. The number represented is the nearest representable single precision (or double precision if `precision` is double) IEEE 754-1985 floating point number (including denormals, but excluding NaNs, +INF, -INF, and -0) to the text representation.

8.3.5 *String Type:*

8.3.5.1 String type E57 elements (String hereafter) are used for storing text. The XML attributes for a String are listed in Table 5.

8.3.5.2 The value of a String shall be encoded in UTF-8 and shall be represented as child text of the XML element. Because the content of a String may include any combination of characters, the XML child text shall appear inside a Character data (CDATA) section. If the character sequence "]]>" appears in the String value, the characters shall be split across two or more CDATA sections such that each "]]>" in the String value is split into "]]" and ">" in adjacent CDATA sections. There shall be no XML whitespace before the first CDATA section, between CDATA sections, or after the last CDATA section.

8.3.6 *Blob Type:*

8.3.6.1 Blob type E57 elements (Blob hereafter) are used for storing opaque blocks of binary data that will be interpreted by the reader. The XML attributes for a Blob are listed in Table 6.

8.3.6.2 A Blob is divided into two parts within an E57 file, an XML portion, documented here, and a binary section. The XML portion indicates the size and location of the binary section of the Blob. The binary section, described in 9.2, stores the actual data content.

8.3.6.3 A Blob shall not contain any child elements or child text.

8.3.6.4 *Discussion*—The format of the Blob's data content is not defined in this specification. A Blob may be used, for example, to embed an image from a camera within an E57 file.

8.3.7 *Structure Type:*

8.3.7.1 Structure-type E57 elements (Structure hereafter) are used to represent unordered groups of potentially heterogeneous E57 elements. The XML attributes for a Structure are listed in Table 7.

8.3.7.2 A Structure shall contain zero or more child E57 elements of any type. The names of the child elements shall be unique within the Structure.

8.3.7.3 A Structure shall not contain any child text.

8.3.8 *Vector Type:*

8.3.8.1 Vector-type E57 elements (Vector hereafter) are used for storing ordered lists of items, known as records.

[6] See "http://www.w3.org/TR/xmlschema-2/

TABLE 2 Attributes for an Integer Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be "Integer".
minimum	optional	-2^{63}	xsd:integer	The smallest value that can be encoded. Shall be in the interval [-2^{63}, 2^{63}-1].
maximum	optional	$2^{63}-1$	xsd:integer	The largest value that can be encoded. Shall be in the interval [minimum, 2^{63}-1].

TABLE 3 Attributes for a ScaledInteger Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be “ScaledInteger”.
minimum	optional	-2^{63}	xsd:integer	The smallest rawValue that can be encoded. Shall be in the interval [-2^{63}, 2^{63}-1].
maximum	optional	$2^{63}-1$	xsd:integer	The largest rawValue that can be encoded. Shall be in the interval [minimum, 2^{63}-1].
scale	optional	1.0	xsd:double	The scale value for the ScaledInteger. Shall be non-zero.
offset	optional	0.0	xsd:double	The offset value for the ScaledInteger.

TABLE 4 Attributes for a Float Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be “Float”.
precision	optional	double	string	Shall be either “single” for 32 bit IEEE 754-1985 floating point values, or “double” for 64 bit IEEE 754-1985 floating point values.
minimum	optional	-3.402823466e+38 if precision is single, or -1.7976931348623158e+308 if precision is double.	xsd:double	The smallest value that can be encoded. Shall be in the interval [-3.402823466e+38, 3.402823466e+38] if precision is single, or [-1.7976931348623158e+308, 1.7976931348623158e+308] if precision is double.
maximum	optional	3.402823466e+38 if precision is single, or 1.7976931348623158e+308 if precision is double.	xsd:double	The largest value that can be encoded. Shall be in the interval [minimum, 3.402823466e+38] if precision is single, or [minimum, 1.7976931348623158e+308] if precision is double.

TABLE 5 Attributes for a String Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be “String”.

TABLE 6 Attributes of a Blob Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be “Blob”.
fileOffset	required	n/a	xsd:integer	The physical file offset of the start of the associated binary Blob section in the E57 file. Shall be in the interval [0, 2^{63}).
length	required	n/a	xsd:integer	The logical length of the associated binary Blob section, in bytes. Shall be in the interval (0, 2^{63}).

TABLE 7 Attributes for a Structure Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be “Structure”.

Vectors are intended to store identical or substantially identically typed records. The XML attributes for a Vector are listed in Table 8.

8.3.8.2 A Vector shall have zero or more child elements, all of which shall use the tag name `vectorChild`.

8.3.8.3 If the `allowHeterogeneousChildren` flag is set to 0, then all the child elements shall have identical structure in terms of number of children, element names, element types, attributes, and descendent elements recursively

TABLE 8 Attributes for a Vector Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be “Vector”.
allowHeterogeneousChildren	optional	1	xsd:integer	Indicates whether the child elements may have different structure. Set to 1 to enable, set to 0 to disable. Shall be either 0 or 1.

(that is, identical except for values), and the Vector is considered to be homogeneous. If the `allowHeterogeneousChildren` flag is set to 1, then the types of the child elements are unconstrained, and the Vector is considered to be heterogeneous.

8.3.8.4 A Vector shall not contain any child text.

8.3.8.5 The element names of the child E57 elements of a Vector shall be string representations of integers beginning with "0" for the first defined child element and incrementing by one for each subsequently defined child element.

8.3.9 *CompressedVector Type:*

8.3.9.1 CompressedVector-type E57 elements (CompressedVector hereafter) are used for storing ordered lists of identically typed items, known as records, in a compressed binary format. The XML attributes for a CompressedVector-type E57 element are listed in Table 9.

8.3.9.2 A CompressedVector is divided into two parts within an E57 file, an XML portion, documented here, and a binary section. The XML portion describes the format of the records using a prototype structure, specifies what compression scheme is used for the data, and indicates the size and location of the binary section of the CompressedVector. The binary section, described in 9.3, stores the actual data content.

8.3.9.3 The child elements for a CompressedVector are listed in Table 10. A CompressedVector shall not contain any child text.

(1) The `prototype` child element specifies the structure of the data that will be stored in the CompressedVector, as well as the possible range of values that the data may take. The `prototype` shall be any E57 element type (with potential sub-children) except Blob and CompressedVector. The values of the `prototype` elements and sub-elements are ignored, and need not be specified.

(2) Discussion—The `prototype` child element describes the abstract requirements of a representation of the data contents. The abstract requirements, in combination with an encoding technique (provided by the `codecs` child element), specify the format of the contents of the binary section of the file. The `prototype` will typically be a Structure with a single level of child elements. A `prototype` with more than one level of child elements is allowed, but not recommended.

(3) The `codecs` child element is a heterogeneous vector of Codec Structures. Each Codec Structure specifies how a set of E57 elements within a record will be compressed in the associated binary section. The child elements for a Codec Structure are listed in Table 11.

(a) The `inputs` child element is a Vector of Strings. Each string is a relative path name of an element in the `prototype` Structure that the codec will compress. The relative pathnames shall be specified with respect to the `prototype` element.

(b) The `bitPackCodec` child element is a Structure with no child elements. Defining this empty Structure specifies that all the elements described by the `inputs` element shall be encoded in the binary section using the bitPackCodec. Operation of the bitPackCodec is described in 9.7.

(4) Each terminal element in the prototype shall be listed at most once as an input to a codec. If a terminal element in the prototype is not listed as an input to a codec, it is implied that the element is compressed by the bitPackCodec.

8.4 *XML Data Hierarchy:*

8.4.1 The XML section of an E57 file shall follow a particular format. A set of data types is defined by this standard to support the storage of 3D point data and 2D imagery in a common, file-level coordinate system. These data types are defined in the following sub-sections. They are constructed from the eight fundamental data types defined in 8.3.

8.4.1.1 *Discussion*—An example instance of an XML data hierarchy is shown in Fig. 2. A more extensive example, in XML format, is given in Appendix X1.

8.4.2 *E57Root:*

8.4.2.1 An E57Root Structure stores the top-level information for the XML section of the file. The child elements for the E57Root Structure are listed in Table 12.

8.4.2.2 The root element of the XML tree shall be an instance of an E57Root Structure with the element name `e57Root`.

8.4.2.3 The E57 XML namespace shall be declared as the default namespace in the E57Root element as:

```
xmlns="http://www.astm.org/COMMIT/E57/2010-e57-v1.0"
```

No other default namespaces shall be declared in child elements of the E57Root element.

8.4.2.4 All XML namespaces (with the exception of the default namespace) shall be declared in the E57Root element as an XML attribute in the following format:

```
xmlns:<namespace>="<uri>"
```

where `<namespace>` is the namespace prefix and `<uri>` is a uniform resource identifier (URI). No XML namespaces shall be declared in child elements of the E57Root element.

8.4.2.5 *Discussion*—The `e57LibraryVersion` String is determined by the developer of the low-level software or library that writes E57 files, not by high-level applications that use the E57 file writing software/library.

8.4.3 *Data3D:*

8.4.3.1 A Data3D Structure represents a collection of 3D points and any associated attributes, as well as metadata about

TABLE 9 Attributes for a CompressedVector Type E57 Element

Attribute Name	Required/ Optional	Default Value	Format	Description
type	required	n/a	string	Shall be "CompressedVector".
fileOffset	required	n/a	xsd:integer	The physical file offset of the start of the CompressedVector binary section in the E57 file (an integer). Shall be in the interval (0, 2^{63}).
recordCount	required	n/a	xsd:integer	The number of records in the compressed binary block (an integer). Shall be in the interval [0, 2^{63}).

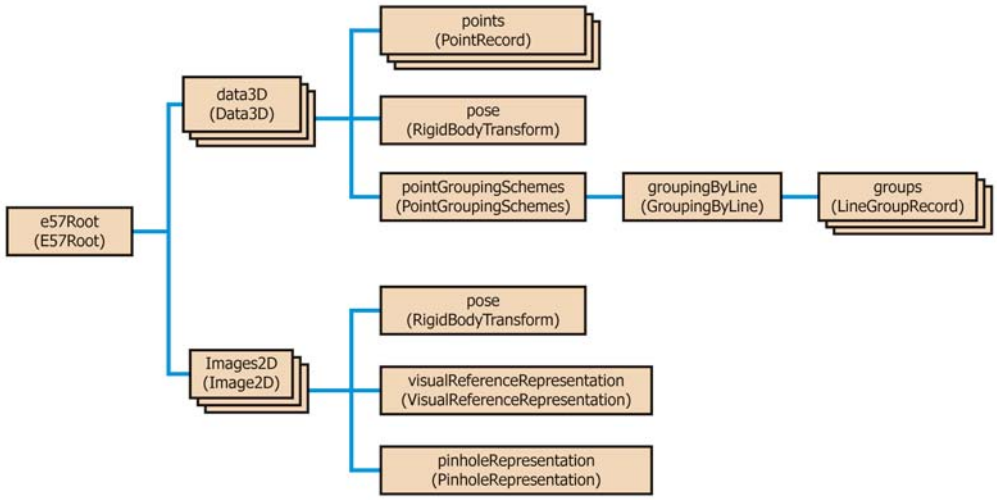
TABLE 10 Child Elements for a CompressedVector Type E57 Element

Element Name	Type	Required/ Optional	Description
prototype	Structure, Integer, Float, ScaledInteger, String, or Vector	required	Specifies the fields of the CompressedVector records and their range limits.
codecs	Vector of Codec Structures	optional	A heterogeneous Vector specifying the compression method to be used for fields within the CompressedVector.

TABLE 11 Child Elements for the Codec Structure

Element Name	Type	Required/ Optional[A]	Description
inputs	Vector of Strings	required	A Vector listing the relative path names of elements in the prototype that this codec will compress.
bitPackCodec	Structure	optional*	Specifies that the bitPackCodec will be used for compressing the data.

[A]Optional fields with an asterisk have additional constraints. See text for details.



NOTE 1—For clarity, not all elements of the hierarchy are shown. Stacked boxes indicate Vectors or Compressed Vectors of elements.

FIG. 2 An Example XML Data Hierarchy Instance

TABLE 12 Child Elements for the E57Root Structure

Element Name	Type	Required/ Optional	Description
formatName	String	required	Shall contain the string “ASTM E57 3D Imaging Data File”.
guid	Guid String	required	A globally unique identification (GUID) String for the current version of the file (see 8.4.22).
versionMajor	Integer	required	The major version number of the file format. Shall be 1.
versionMinor	Integer	required	The minor version number of the file format. Shall be 0.
e57LibraryVersion	String	optional	The version identifier for the E57 file format library that wrote the file.
creationDateTime	DateTime Structure	optional	Date and time that the file was created.
data3D	Vector of Data3D Structures	optional	A heterogeneous Vector of Data3D Structures for storing 3D imaging data.
images2D	Vector of Image2D Structures	optional	A heterogeneous Vector of Image2D Structures for storing 2D images from a camera or similar device.
coordinateMetadata	CoordinateMetadata String	optional	Information describing the Coordinate Reference System to be used for the file.

the collection of points. The child elements for the Data3D Structure are listed in Table 13.

8.4.3.2 The 3D points shall be stored either in a local coordinate system relative to the sensor or in a file-level coordinate system common to all the 3D data sets in an E57 file. If the points are stored using the local coordinate system, the `pose` child element shall be present and shall store the transform that, when applied to the 3D points, will place them in the file-level coordinate system. If the points are stored using the file-level coordinate system, the `pose` child element shall be omitted, and the identity transform shall be implied for the `pose`.

(1) Points shall be stored in the local coordinate system relative to the sensor when possible.

(2) Discussion—This statement implies that storing the points in the file-level coordinate system and discarding the

TABLE 13 Child Elements for the Data3D Structure

Element Name	Type	Required/ Optional[A]	Description
guid	Guid String	required	A globally unique identifier for the current version of the Data3D object (see 8.4.22).
points	CompressedVector of PointRecord Structures	required	A compressed vector of PointRecord Structures referring to the binary data that actually stores the point data.
pose	RigidBodyTransform Structure	optional	A rigid body transform that transforms data stored in the local coordinate system of the points to the file-level coordinate system.
originalGuids	Vector of Guid Strings	optional	A Vector of globally unique identifiers identifying the data set (or sets) from which the points in this Data3D originated.
pointGroupingSchemes	PointGroupingSchemes Structure	optional	The defined schemes that group points in different ways.
name	String	optional	A user-defined name for the Data3D.
description	String	optional	A user-defined description of the Data3D.
cartesianBounds	CartesianBounds Structure	optional*	The bounding region (in Cartesian coordinates) of all the points in this Data3D (in the local coordinate system of the points).
sphericalBounds	SphericalBounds Structure	optional*	The bounding region (in spherical coordinates) of all the points in this Data3D (in the local coordinate system of the points).
indexBounds	IndexBounds Structure	optional*	The bounds of the row, column, and return number of all the points in this Data3D.
intensityLimits	IntensityLimits Structure	optional*	The limits for the value of signal intensity that the sensor is capable of producing.
colorLimits	ColorLimits Structure	optional*	The limits for the value of red, green, and blue color that the sensor is capable of producing.
acquisitionStart	DateTime Structure	optional	The start date and time that the data was acquired.
acquisitionEnd	DateTime Structure	optional	The end date and time that the data was acquired.
sensorVendor	String	optional	The name of the manufacturer for the sensor used to collect the points in this Data3D.
sensorModel	String	optional	The model name or number for the sensor.
sensorSerialNumber	String	optional	The serial number for the sensor.
sensorHardwareVersion	String	optional	The version identifier for the sensor hardware at the time of data collection.
sensorSoftwareVersion	String	optional	The version identifier for the software used for the data collection.
sensorFirmwareVersion	String	optional	The version identifier for the firmware installed in the sensor at the time of data collection.
temperature	Float	optional	The ambient temperature, measured at the sensor, at the time of data collection (in degrees Celsius). Shall be ≥ –273.15° (absolute zero).
relativeHumidity	Float	optional	The percentage relative humidity, measured at the sensor, at the time of data collection. Shall be in the interval [0, 100].
atmosphericPressure	Float	optional	The atmospheric pressure, measured at the sensor, at the time of data collection (in Pascals). Shall be positive.

[A]Optional fields with an asterisk have additional constraints. See text for details.

pose transform is prohibited if the points are known in the local coordinate system, since this practice results in loss of information.

8.4.3.3 The `originalGuids` child element identifies the data set (or sets) from which the data originated (that is, not the most recent version, but the first version). If the `originalGuids` element is present, the strings stored in the Vector shall contain the GUIDs that identify the source of the data in the Data3D object. The absence of the `originalGuids` element shall indicate that the Data3D object has not been modified from its original version. In this case, the original version shall be indicated by the `guid` element.

8.4.3.4 If the points stored in the `pointRecord` element are represented in Cartesian coordinates (that is, if `cartesianX`, `cartesianY`, and `cartesianZ` are defined), then the `cartesianBounds` element shall be defined.

8.4.3.5 If the points in the `pointRecord` element are represented in spherical coordinates (that is, if `sphericalRange`, `sphericalElevation`, and `sphericalAzimuth` are defined), then the `sphericalBounds` element shall be defined.

8.4.3.6 If the `rowIndex` element is defined for the points in the `pointRecord`, then the `indexBounds` element shall be defined and within the `indexBounds` element, the `rowMinimum` and `rowMaximum` shall be defined. If the `columnIndex` element is defined for the points in the `pointRecord`, then the `indexBounds` element shall be defined and within the `indexBounds` element, the `columnMinimum` and `columnMaximum` shall be defined. If the `returnIndex` element is defined for the points in the `pointRecord` , then the `indexBounds` element shall be defined and within the `indexBounds element`, the `returnMinimum` and `returnMaximum` shall be defined.

8.4.3.7 If the `intensity` element is defined for the points in the `pointRecord`, and if the minimum and maximum intensity values that can be produced by the device that

obtained the intensity measurements are known, then the `intensityLimits` element shall be defined.

8.4.3.8 If the `colorRed` , `colorGreen`, and `colorBlue` elements are defined for the points in the `pointRecord`, and if the minimum and maximum color values that can be produced by the device that obtained the color measurements are known, then the `colorLimits` element shall be defined.

8.4.4 *PointRecord:*

8.4.4.1 A PointRecord Structure stores the information for an individual point measurement from a 3D imaging system. The child elements for the PointRecord Structure are listed in Table 14.

8.4.4.2 One or more of the following elements shall be defined: `cartesianX`, `sphericalRange`. If any elements in the set {`cartesianX`, `cartesianY`, `cartesianZ`} are defined, then all elements in that set shall be defined. If any elements in the set {`sphericalRange`, `sphericalAzimuth`, `sphericalElevation` } are defined, then all elements in that set shall be defined.

8.4.4.3 The values for `sphericalRange`, `sphericalAzimuth`, and `sphericalElevation` are restricted to the limits described for spherical coordinates in 5.5.

8.4.4.4 If `returnIndex` or `returnCount` are defined, then both `returnIndex` and `returnCount` shall be defined. If `returnIndex` is defined, the PointRecords from an emitted single pulse shall be contiguous, and shall be sorted in ascending `returnIndex` values.

8.4.4.5 The `intensity` element shall encode the strength of the signal for a point. The intensity value shall not include compensation for signal strength reduction as a function of distance, surface orientation, or other properties of the surface being sensed.

8.4.4.6 If any elements in the set {`colorRed`, `colorGreen` , `colorBlue`} are defined, then all elements in that set shall be defined. The units of `colorRed`, `colorGreen`, and `colorBlue` are not specified, but they shall all be the same.

8.4.4.7 If `cartesianInvalidState` is defined, its value shall have the following interpretation. If the value is 0, the values of `cartesianX` , `cartesianY`, and `cartesianZ` shall all be meaningful. If the value is 1, only the direction component of the vector (`cartesianX`, `cartesianY`, `cartesianZ`) shall be meaningful, and the magnitude of the vector shall be considered non-meaningful. If the value is 2, the values of `cartesianX`, `cartesianY`, and `cartesianZ` shall all be considered non-meaningful.

8.4.4.8 If `sphericalInvalidState` is defined, its value shall have the following interpretation. If the value is 0, the values of `sphericalRange` , `sphericalAzimuth`, and `sphericalElevation` shall all be meaningful. If the

TABLE 14 Child Elements for the PointRecord Structure

Element Name	Type	Required/ Optional[A]	Description
cartesianX	Float/ScaledInteger/Integer	optional*	The X coordinate (in meters) of the point in Cartesian coordinates.
cartesianY	Float/ScaledInteger/Integer	optional*	The Y coordinate (in meters) of the point in Cartesian coordinates.
cartesianZ	Float/ScaledInteger/Integer	optional*	The Z coordinate (in meters) of the point in Cartesian coordinates.
sphericalRange	Float/ScaledInteger/Integer	optional*	The range (in meters) of points in spherical coordinates. Shall be non-negative.
sphericalAzimuth	Float/ScaledInteger	optional*	Azimuth angle (in radians) of point in spherical coordinates (see 5.5 for restrictions on values).
sphericalElevation	Float/ScaledInteger	optional*	Elevation angle (in radians) of point in spherical coordinates (see 5.5 for restrictions on values).
rowIndex	Integer	optional	The row number of point (zero-based). This is useful for data that is stored in a regular grid.Shall be in the interval [0, 2^{63}).
columnIndex	Integer	optional	The column number of point (zero-based). This is useful for data that is stored in a regular grid. Shall be in the interval [0, 2^{63}).
returnCount	Integer	optional*	Only for multi-return sensors. The total number of returns for the pulse that this corresponds to. Shall be in the interval (0, 2^{63}).
returnIndex	Integer	optional*	Only for multi-return sensors. The number of this return (zero based). That is, 0 is the first return, 1 is the second, and so on. Shall be in the interval [0, returnCount).
timeStamp	Float/ScaledInteger/Integer	optional	The time (in seconds) since the start time for the data, which is given by acquisitionStart in the parent Data3D Structure. Shall be non-negative.
intensity	Float/ScaledInteger/Integer	optional	Point response intensity. Unit is unspecified.
colorRed	Float/ScaledInteger/Integer	optional*	Red color coefficient. Unit is unspecified.
colorGreen	Float/ScaledInteger/Integer	optional*	Green color coefficient. Unit is unspecified.
colorBlue	Float/ScaledInteger/Integer	optional*	Blue color coefficient. Unit is unspecified.
cartesianInvalidState	Integer	optional	Indicates whether the Cartesian coordinate vector or its magnitude is meaningful. Shall be in the interval [0, 2].
sphericalInvalidState	Integer	optional	Indicates whether the spherical coordinate vector or its range value are meaningful. Shall be in the interval [0, 2].
isTimeStampInvalid	Integer	optional	Indicates whether the timeStamp element is meaningful. Shall be in the interval [0, 1].
isIntensityInvalid	Integer	optional	Indicates whether the intensity element is meaningful. Shall be in the interval [0, 1].
isColorInvalid	Integer	optional	Indicates whether the colorRed, colorBlue, and colorGreen elements are meaningful. Shall be in the interval [0, 1].

[A] Optional fields with an asterisk have additional constraints. See text for details.

value is 1, the value of sphericalRange shall be considered non-meaningful, and the value of sphericalAzimuth, and sphericalElevation shall be meaningful. If the value is 2, the values of sphericalRange, sphericalAzimuth , and sphericalElevation shall all be considered non-meaningful.

8.4.4.9 If isTimeStampInvalid is defined and its value is 1, the value of timeStamp shall be considered non-meaningful. Otherwise, the value of timeStamp shall be meaningful.

8.4.4.10 If isIntensityInvalid is defined and its value is 1, the value of intensity shall be considered non-meaningful. Otherwise, the value of intensity shall be meaningful.

8.4.4.11 If isColorInvalid is defined and its value is 1, the values of colorRed, colorGreen , and colorBlue shall be considered non-meaningful. Otherwise, the values of colorRed, colorGreen , and colorBlue shall all be meaningful.

8.4.4.12 If the value of cartesianInvalidState is 1 or 2, the point shall not be used for computing the cartesianBounds values.

8.4.4.13 If the value of sphericalInvalidState is 1 or 2, the point shall not be used for computing the sphericalBounds values.

8.4.4.14 *Discussion*—Elements that are considered non-meaningful should be set to a consistent value, such as the minimum allowable value for that element, for all points.

8.4.5 *PointGroupingScheme:*

8.4.5.1 The PointGroupingSchemes Structure supports the division of points within a Data3D object into logical groupings. The child elements for the PointGroupingSchemes Structure are listed in Table 15.

8.4.5.2 The PointGroupingSchemes Structure stores the defined grouping schemes of a Data3D object. Each scheme has a unique element name in the Structure. The standard defines one grouping scheme: GroupingByLine.

8.4.5.3 *Discussion*—Additional types of grouping schemes may be defined in future versions of the standard or via the extension mechanism. It is recommended that a grouping scheme document the following:

(1) Does the scheme completely cover the entire set of points, or can there be some points missing?

(2) Can the groups overlap by sharing some points, or are the groups non-intersecting?

(3) Is there a rule that determines membership in a group based solely on a point's attributes?

8.4.6 *GroupingByLine:*

8.4.6.1 The GroupingByLine Structure stores a set of point groups organized by the rowIndex or columnIndex element of the PointRecord. The child elements for the GroupingByLine Structure are listed in Table 16.

8.4.6.2 The groupingByLine partitions the points of a Data3D object. Membership in a line group is determined by a rule (described below) based on a point's attributes.

8.4.6.3 The term "associated points" refers to the set of PointRecords stored in the points element of the data3D element of which the groupingByLine Structure is a descendant. If idElementName is "rowIndex," the associated points are grouped according to their rowIndex, and the number of records in the groups CompressedVector shall equal the number of distinct values taken by the rowIndex field in all associated points. If idElementName is "columnIndex," the associated points are grouped according to their columnIndex, and the number of records in the groups CompressedVector shall equal the number of distinct values taken by the columnIndex field in all associated points. The records in the groups CompressedVector may be in any order (that is, they are not required to be sorted).

8.4.7 *LineGroupRecord:*

8.4.7.1 A LineGroupRecord Structure stores information about a single group of points in a row or column. The child elements for the LineGroupRecord Structure are listed in Table 17.

8.4.7.2 If the idElementName of the parent Structure is "rowIndex," all points with rowIndex equal to the value of idElementValue are considered to be in the group. If the idElementName of the parent Structure is "columnIndex," all points with columnIndex equal to the value of idElementValue are considered to be in the group. The idElementName value for each record in a group's CompressedVector shall be unique within that CompressedVector.

8.4.7.3 If the startPointIndex element is defined, then the pointCount element shall also be defined, and the members of the group (as determined by idElementValue) shall be in consecutive records in the points CompressedVector starting with the record number given by startPointIndex. In this case, the group is considered to be contiguous. If startPointIndex is not defined, then the PointRecord members of the group may be noncontiguous.

(1) Discussion—Because a LineGroupRecord is stored in a CompressedVector, if the startPointIndex is defined for one LineGroupRecord, it shall be defined for all other LineGroupRecords in the same CompressedVector, and therefore the contiguous grouping encoding can only be used if all groups in the parent GroupingByLine structure are contiguous.

8.4.8 *RigidBodyTransform:*

8.4.8.1 A RigidBodyTransform Structure that defines a rigid body transform in Cartesian coordinates. Child elements for the RigidBodyTransform Structure are listed in Table 18.

8.4.8.2 The transform consists of a rotation, stored as a unit-length quaternion, and a translation.

8.4.8.3 *Discussion*—See 5.7 for the mathematical background on rigid body transforms.

TABLE 15 Child Elements for the PointGroupingSchemes Structure

Element Name	Type	Required/ Optional	Description
groupingByLine	GroupingByLine Structure	optional	Grouping information by row or column index.

TABLE 16 Child Elements for the GroupingByLine Structure

Element Name	Type	Required/ Optional	Description
idElementName	String	required	The name of the PointRecord element that identifies which group the point is in. The value of this string shall be "rowIndex" or "columnIndex" (see 8.4.4.2).
groups	CompressedVector of LineGroupRecord Structures	required	A compressedVector of LineGroupRecord Structures.

TABLE 17 Child Elements for the LineGroupRecord Structure

Element Name	Type	Required/ Optional[A]	Description
idElementValue	Integer	required	The value of the identifying element of all members in this group. Shall be in the interval [0, 2^{63}).
startPointIndex	Integer	optional*	The record number of the first point in the continuous interval. Shall be in the interval [0, 2^{63}).
pointCount	Integer	optional*	The number of PointRecords in the group. Shall be in the interval [1, 2^{63}).
cartesianBounds	CartesianBounds Structure	optional	The bounding box (in Cartesian coordinates) of all points in the group (in the local coordinate system of the points).
sphericalBounds	SphericalBounds Structure	optional	The bounding region (in spherical coordinates) of all the points in the group (in the local coordinate system of the points).

[A] Optional fields with an asterisk have additional constraints. See text for details.

TABLE 18 Child Elements for the RigidBodyTransform Structure

Element Name	Type	Required/ Optional	Description
rotation	Quaternion Structure	required	A unit quaternion representing the rotation, *R*, of the transform.
translation	Translation Structure	required	The translation, *t*, of the transform.

8.4.9 *Quaternion:*

8.4.9.1 A Quaternion Structure stores a quaternion. The child elements for the Quaternion Structure are listed in Table 19.

8.4.9.2 The child elements, w, x, y, and z, shall be represented using double precision Float type E57 elements.

8.4.9.3 A quaternion shall be considered to be unit length if the norm of the quaternion lies in the interval [1 – 10*ulp*, 1 + 10*ulp*], where the value of *ulp*, the "unit in last place," is 2^{-53}.

8.4.9.4 *Discussion*—See 5.6 for the mathematical background on quaternions.

8.4.10 *Translation:*

8.4.10.1 A translation Structure defines a rigid body translation in Cartesian coordinates. The child elements for the Translation Structure are listed in Table 20.

8.4.10.2 The child elements, x, y, and z, shall be represented using double precision Float type E57 elements.

8.4.10.3 The values of *x*, *y*, and *z* are the translation distances in the *X*, *Y*, and *Z* directions, respectively.

8.4.11 *Image2D:*

8.4.11.1 An Image2D Structure stores a two-dimensional image, such as an image produced by a camera. The child elements for the Image2D Structure are listed in Table 21.

8.4.11.2 The image data is stored as part of a child element, known as an image representation. Four image representations are possible: VisualReferenceRepresentation (8.4.12), PinholeRepresentation (8.4.13), SphericalRepresentation (8.4.14), and CylindricalRepresentation (8.4.15). At least one of the following elements shall be defined: `visualReferenceRepresentation`, `pinholeRepresentation`, `sphericalRepresentation`, and `cylindricalRepresentation`. At most one of the elements from the set {`pinholeRepresentation`, `sphericalRepresentation`, `cylindricalRepresentation`} shall be defined, and if any element from this set is defined, then the `pose` element shall also be defined.

8.4.11.3 An image may be used for visual reference, for matching 3D points to image pixels (projection), or for both purposes. If the image is intended to be used for visual

TABLE 19 Child Elements for the Quaternion Structure

Element Name	Type	Required/ Optional	Description
w	Float (double precision)	required	The scalar part of the quaternion. Shall be nonnegative.
x	Float (double precision)	required	The *i* coefficient of the quaternion.
y	Float (double precision)	required	The *j* coefficient of the quaternion.
z	Float (double precision)	required	The *k* coefficient of the quaternion.

TABLE 20 Child Elements for the Translation Structure

Element Name	Type	Required/ Optional	Description
x	Float (double precision)	required	The *X* coordinate of the translation (in meters)
y	Float (double precision)	required	The *Y* coordinate of the translation (in meters)
z	Float (double precision)	required	The *Z* coordinate of the translation (in meters)

TABLE 21 Child Elements for the Image2D Structure

Element Name	Type	Required/ Optional[A]	Description
guid	Guid String	required	A globally unique identifier for the current version of the Image2D object (see 8.4.22).
visualReferenceRepresentation	VisualReferenceRepresentation Structure	optional*	Representation for an image that does not define any camera projection model. The image is to be used for visual reference only.
pinholeRepresentation	PinholeRepresentation Structure	optional*	Representation for an image using the pinhole camera projection model.
sphericalRepresentation	SphericalRepresentation Structure	optional*	Representation for an image using the spherical camera projection model.
cylindricalRepresentation	CylindricalRepresentation Structure	optional*	Representation for an image using the cylindrical camera projection model.
pose	RigidBodyTransform Structure	optional*	A rigid body transform that transforms data stored in the local coordinate system of the image to the file-level coordinate system.
associatedData3DGuid	Guid String	optional	The globally unique identifier for the Data3D object that was being acquired when the picture was taken (see 8.4.22).
name	String	optional	A user-defined name for the Image2D.
description	String	optional	A user-defined description of the Image2D.
acquisitionDateTime	DateTime Structure	optional	The date and time that the image was acquired.
sensorVendor	String	optional	The name of the manufacturer for the sensor used to collect the image in this Image2D.
sensorModel	String	optional	The model name or number for the sensor.
sensorSerialNumber	String	optional	The serial number for the sensor.

[A] Optional fields with an asterisk have additional constraints. See text for details.

reference, then the VisualReferenceRepresentation shall be used. If the image is intended to be used for projection, then one of the other three image representations shall be used.

8.4.12 *VisualReferenceRepresentation :*

8.4.12.1 A VisualReferenceRepresentation Structure stores an image that is to be used only as a visual reference. The child elements for the VisualReferenceRepresentation Structure are listed in Table 22.

8.4.12.2 The image data, which may be stored either in JPEG or PNG format, is stored in a Blob.

8.4.12.3 Exactly one of `jpegImage` or `pngImage` shall be defined, and the choice indicates the format in which the image is stored.

8.4.12.4 In the case in which the image data represents a nonrectangular image, the `imageMask` child element should be used to indicate which pixels in the image are valid. The `imageMask` is a PNG format image with the same dimensions as the image, but with non-zero-valued pixels at locations where the image is valid and zero-valued pixels at locations where it is invalid.

8.4.13 *PinholeRepresentation:*

8.4.13.1 A PinholeRepresentation Structure stores an image that is mapped from 3D using the pinhole camera projection model. The child elements for the PinHoleRepresentation Structure are listed in Table 23.

8.4.13.2 The image data, which shall be stored either in JPEG for PNG format, is stored in a Blob. Exactly one of `jpegImage` or `pngImage` shall be defined, and the choice indicates the format in which the image is stored.

8.4.13.3 An image shall be undistorted before storage in the E57 file. Correcting a raw image for distortion may produce a nonrectangular result. To address this issue, the undistorted image shall be stored in a rectangular image that encompasses the undistorted image data. The `imageMask` is a PNG format

TABLE 22 Child Elements for the VisualReferenceRepresentation Structure

Element Name	Type	Required/ Optional[A]	Description
jpegImage	Blob	optional*	JPEG format image data.
pngImage	Blob	optional*	PNG format image data.
imageMask	Blob	optional	PNG format image mask.
imageWidth	Integer	required	The image width (in pixels). Shall be positive.
imageHeight	Integer	required	The image height (in pixels). Shall be positive.

[A] Optional fields with an asterisk have additional constraints. See text for details.

TABLE 23 Child Elements for the PinholeRepresentation Structure

Element Name	Type	Required/ Optional[A]	Description
jpegImage	Blob	optional*	JPEG format image data.
pngImage	Blob	optional*	PNG format image data.
imageMask	Blob	optional	PNG format image mask.
imageWidth	Integer	required	The image width (in pixels). Shall be positive.
imageHeight	Integer	required	The image height (in pixels). Shall be positive.
focalLength	Float	required	The camera's focal length (in meters). Shall be positive.
pixelWidth	Float	required	The width of the pixels in the camera (in meters). Shall be positive.
pixelHeight	Float	required	The height of the pixels in the camera (in meters). Shall be positive.
principalPointX	Float	required	The *X* coordinate in the image of the principal point, (in pixels). The principal point is the intersection of the z axis of the camera coordinate frame with the image plane.
principalPointY	Float	required	The *Y* coordinate in the image of the principal point (in pixels).

[A] Optional fields with an asterisk have additional constraints. See text for details.

image with the same dimensions as the image, but with non-zero-valued pixels at locations where the undistorted image is valid and zero-valued pixels at locations where it is invalid.

8.4.13.4 The pinhole camera projection model uses a camera coordinate frame in which the origin is located at the camera's center of projection, the imaging plane is located at a distance–`focalLength` along the z-axis, and the x- and y-axes are parallel with the x- and negative y-axes of the imaging plane, as illustrated in Fig. 3. The model assumes that the image has been corrected for radial and tangential distortion and that the pixels have no skew (that is, the pixels are rectangular). Radial distortion is the displacement of image points in a radial direction from the principal point. Tangential distortion is the displacement of image points perpendicular to a radius from the principal point.

8.4.13.5 Given a point (x, y, z) in Cartesian coordinates in the camera frame of reference, where $z < 0$, the image coordinates (x_{image}, y_{image}) are given by the following equations of projection:

$$x_{image} = principalPointX - \left(\frac{x}{z}\right)\left(\frac{focalLength}{pixelWidth}\right) \quad (20)$$

$$y_{image} = principalPointY + \left(\frac{y}{z}\right)\left(\frac{focalLength}{pixelHeight}\right) \quad (21)$$

8.4.13.6 Image coordinate (0,0) is the top, left corner of the pixel at the top, left corner of the image.

8.4.14 *SphericalRepresentation:*

8.4.14.1 A SphericalProjection Structure stores an image that is mapped from 3D using a spherical projection model. The child elements for the SphericalRepresentation Structure are listed in Table 24.

8.4.14.2 The image data, which shall be stored either in JPEG or PNG format, is stored in a Blob. Exactly one of `jpegImage` or `pngImage` shall be defined, and the choice indicates the format in which the image is stored.

8.4.14.3 The conversion of an image into the spherical projection model may cause the image to be non-rectangular. To address this issue, the image shall be stored in a rectangular image that encompasses the nonrectangular image data. The `imageMask` child element shall be used to indicate which pixels in the image are valid. The `imageMask` is a PNG format image with the same dimensions as the image, but with non-zero-valued pixels at locations where the image is valid and zero-valued pixels at locations where it is invalid.

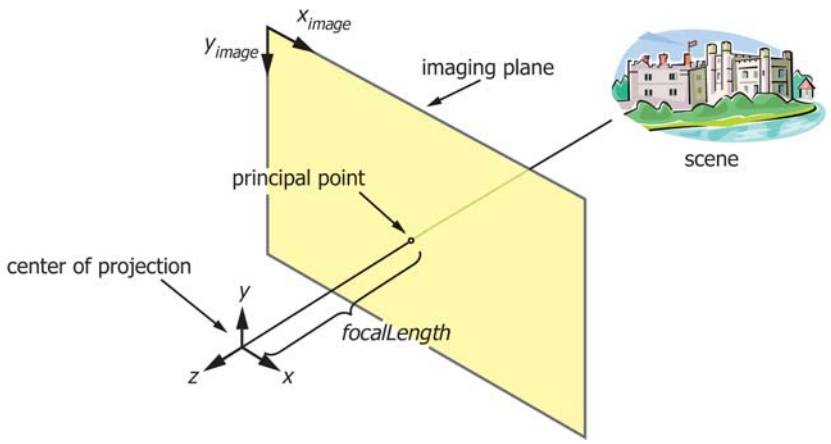


FIG. 3 Pinhole Camera Projection Model

TABLE 24 Child Elements for the SphericalRepresentation Structure

Element Name	Type	Required/ Optional[A]	Description
jpegImage	Blob	optional*	JPEG format image data.
pngImage	Blob	optional*	PNG format image data.
imageMask	Blob	optional	PNG format image mask.
imageWidth	Integer	required	The image width (in pixels). Shall be positive.
imageHeight	Integer	required	The image height (in pixels). Shall be positive.
pixelWidth	Float	required	The width of a pixel in the image (in radians). Shall be positive.
pixelHeight	Float	required	The height of a pixel in the image (in radians). Shall be positive.

[A] Optional fields with an asterisk have additional constraints. See text for details.

8.4.14.4 In a spherical projection model, the imaging surface is part of the surface of a sphere, bounded by constant values of elevation and azimuth. The model uses a camera coordinate frame in which the origin is located at the camera's center of projection and the positive *x*-axis passes through the center of the image, as illustrated in Fig. 4.

8.4.14.5 Given a point in spherical coordinates (r, θ, φ), the image coordinates (x_{image}, y_{image}) are given by the following equations of projection:

$$x_{image} = \frac{imageWidth}{2} - \frac{\theta}{pixelWidth} \quad (22)$$

$$y_{image} = \frac{imageHeight}{2} - \frac{\varphi}{pixelHeight} \quad (23)$$

8.4.14.6 Image coordinate (0,0) is the top, left corner of the pixel at the top, left corner of the image.

8.4.15 *CylindricalRepresentation:*

8.4.15.1 A CylindricalProjection Structure stores an image that is mapped from 3D using a cylindrical projection model. The child elements for the CylindricalRepresentation Structure are listed in Table 25.

8.4.15.2 The image data, which shall be stored either in JPEG for PNG format, is stored in a Blob. Exactly one of `jpegImage` or `pngImage` shall be defined, and the choice indicates the format in which the image is stored.

8.4.15.3 The conversion of an image into the cylindrical projection model may cause the image to be non-rectangular. To address this issue, the image shall be stored in a rectangular image that encompasses the non-rectangular image data. The `imageMask` child element shall be used to indicate which pixels in the image are valid. The `imageMask` is a PNG format image with the same dimensions as the image, but with non-zero-valued pixels at locations where the image is valid and zero-valued pixels at locations where it is invalid.

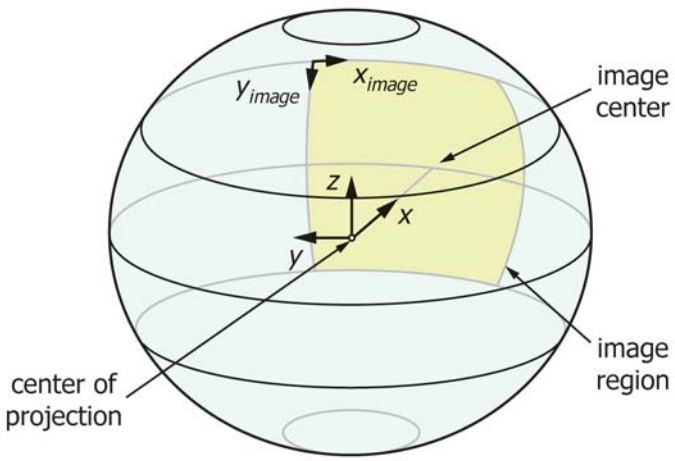


FIG. 4 Spherical Camera Projection Model

8.4.15.4 In a cylindrical projection model, the imaging surface is part of the surface of a cylinder, bounded by constant values of height and azimuth. The model uses a cylindrical camera coordinate frame in which the origin is located at the camera's center of projection and the *x*-axis passes through the horizontal center of the image, as illustrated in Fig. 5. The image *y*-axis is parallel with the camera's negative *z*-axis, and the *x*-axis of the image corresponds to negative azimuth angle.

8.4.15.5 Given a point in cylindrical coordinates (ρ, θ, z), the image coordinates (x_{image}, y_{image}) are given by the following equations of projection:

$$x_{image} = \frac{imageWidth}{2} - \frac{\theta}{pixelWidth} \quad (24)$$

$$y_{image} = principalPointY - z(radius/pixelHeight)/\rho \quad (25)$$

8.4.15.6 Image coordinate (0,0) is the top, left corner of the pixel at the top, left corner of the image.

8.4.16 *CartesianBounds:*

8.4.16.1 A CartesianBounds Structure specifies a box with smallest volume enclosing a specific set of points, subject to the constraint that the edges of the box are parallel to the Cartesian coordinate axes. The child elements describe the minimum and maximum coordinates of any data point in each dimension and are listed in Table 26.

8.4.16.2 All maximum values for child elements of CartesianBounds shall be not less than the corresponding minimum values in each direction.

8.4.17 *SphericalBounds:*

8.4.17.1 A SphericalBounds Structure stores the bounds of a set of points in spherical coordinates. The child elements for the SphericalBounds Structure are listed in Table 27.

8.4.17.2 Child elements describe the minimum and maximum coordinates in range, azimuth, and elevation. The `azimuthStart` and `azimuthEnd` elements shall be omitted if there are no bounds in the azimuth direction (for example, if the data represents a full circular sweep of a rotating sensor). If either `azimuthStart` or `azimuthEnd` is present, then both elements are required to be present.

TABLE 25 Child Elements for the CylindricalRepresentation Structure

Element Name	Type	Required/ Optional[A]	Description
jpegImage	Blob	optional*	JPEG format image data.
pngImage	Blob	optional*	PNG format image data.
imageMask	Blob	optional	PNG format image mask.
imageWidth	Integer	required	The image width (in pixels). Shall be positive.
imageHeight	Integer	required	The image height (in pixels). Shall be positive.
radius	Float	required	The closest distance from the cylindrical image surface to the center of projection (that is, the radius of the cylinder) (in meters). Shall be non-negative.
principalPointY	Float	required	The *Y* coordinate in the image of the principal point (in pixels). This is the intersection of the z = 0 plane with the image.
pixelWidth	Float	required	The width of a pixel in the image (in radians). Shall be positive.
pixelHeight	Float	required	The height of a pixel in the image (in meters). Shall be positive.

[A] Optional fields with an asterisk have additional constraints. See text for details.

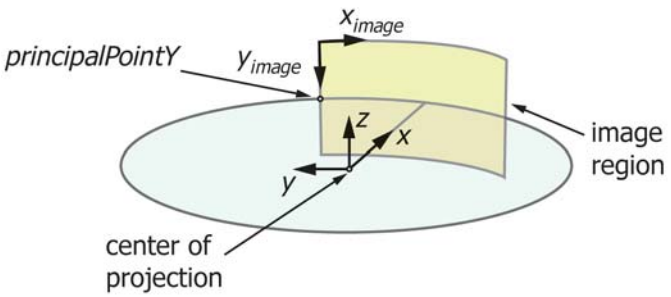


FIG. 5 Cylindrical Camera Projection Model

8.4.17.3 The ordered triplets (`rangeMinimum`, `azimuthStart`, `elevationMinimum`) and (`rangeMaximum`, `azimuthEnd`, `elevationMaximum`) represent points in a spherical coordinate system, and as such, they are subject to the restrictions on range, units, and degenerate representations described in 5.5.

8.4.17.4 Given a set of points, $\{p_i\}$, represented in spherical coordinates, $(r_i, \theta_i, \varphi_i)$, the bounds for range and elevation are defined as follows:

$$rangeMinimum = \min\{r_i\}$$
$$rangeMaximum = \max\{r_i\}$$
$$elevationMinimum = \min\{\varphi_i\}$$
$$elevationMaximum = \max\{\varphi_i\}$$

8.4.17.5 The elements `azimuthStart` and `azimuthEnd` are defined such that the bounded data is contained between these angles when traversed from `azimuthStart` to `azimuthEnd` in the direction of increasing azimuth (crossing the discontinuity at π if necessary). The values of `azimuthStart` and `azimuthEnd` shall be chosen to minimize the angular difference measured from `azimuthStart` to `azimuthEnd` in the direction of increasing azimuth.

8.4.17.6 *Discussion*—The azimuth bounds of a set of points in spherical coordinates are not guaranteed to be unique.

8.4.18 *IndexBounds:*

8.4.18.1 An IndexBounds Structure stores the minimum and maximum of rowIndex, columnIndex, and returnIndex fields for a set of points. The child elements for the IndexBounds Structure are listed in Table 28.

8.4.18.2 If any elements in the set {`rowMinimum`, `rowMaximum`} are defined, then all elements in that set shall be defined. If any elements in the set {`columnMinimum`, `columnMaximum`} are defined, then all elements in that set shall be defined. If any elements in the set {`returnMinimum`, `returnMaximum`} are defined, then all elements in that set shall be defined.

8.4.18.3 *Discussion*—The IndexBounds is intended to be used with 3D data that is aligned on a grid or with multiple-return data. The information can be used, for example, to pre-allocate space for storing point data in a 2D array.

8.4.19 *IntensityLimits:*

8.4.19.1 An IntensityLimits Structure specifies the limits for the value of signal intensity that a sensor is capable of producing. In this section, the term `points` refers to the points element of the parent Data3D Structure for this IntensityLimits Structure, and the term PointRecord refers to the corresponding PointRecord defined by the `prototype` element of `points`. See Table 29.

8.4.19.2 The `intensityMinimum` and `intensityMaximum` child elements shall correspond to the minimum and maximum intensity values that the sensor is capable of producing in the configuration used to collect the data stored in `points`.

8.4.19.3 The units of `intensityMinimum` and `intensityMaximum` are unspecified, but they shall be the same as the units used for the `intensity` element in the PointRecord.

8.4.19.4 It is recommended that the element type of `intensityMinimum` and `intensityMaximum` be the same as the `intensity` element in the PointRecord.

8.4.19.5 The intensity values stored in `points` shall be restricted to the range [`intensityMinimum`, `intensityMaximum`].

8.4.19.6 *Discussion*—The purpose of the `intesityLimits` element is to facilitate scaling of the intensity values of the stored points such that they are comparable across different data sets from the same type of sensor. The `intensityLimits` values should be set based on the range of *possible* measurable values for a sensor in a given configuration, not the *actual* measured values for a particular data set.

8.4.20 *ColorLimits:*

TABLE 26 Child Elements for the CartesianBounds Structure

Element Name	Type	Required/ Optional	Description
xMinimum	Float	required	The minimum extent of the bounding region in the *X* direction (in meters).
xMaximum	Float	required	The maximum extent of the bounding region in the *X* direction (in meters).
yMinimum	Float	required	The minimum extent of the bounding region in the *Y* direction (in meters).
yMaximum	Float	required	The maximum extent of the bounding region in the *Y* direction (in meters).
zMinimum	Float	required	The minimum extent of the bounding region in the *Z* direction (in meters).
zMaximum	Float	required	The maximum extent of the bounding region in the *Z* direction (in meters).

TABLE 27 Child Elements for the SphericalBounds Structure

Element Name	Type	Required/ Optional[A]	Description
rangeMinimum	Float	required	The minimum extent of the bounding region in the r direction (in meters). Shall be non-negative.
rangeMaximum	Float	required	The maximum extent of the bounding region in the r direction (in meters). Shall be $\geq$ rangeMinimum.
elevationMinimum	Float	required	The minimum extent of the bounding region in the φ direction (in radians). Shall be in the interval [$-\pi/2$, $\pi/2$].
elevationMaximum	Float	required	The maximum extent of the bounding region in the φ direction (in radians). Shall be in the interval [elevationMinimum, $\pi/2$].
azimuthStart	Float	optional*	The starting azimuth angle defining the extent of the bounding region in the θ direction (in radians). Shall be in the interval ($-\pi$, π].
azimuthEnd	Float	optional*	The ending azimuth angle defining the extent of the bounding region in the θ direction (in radians). Shall be in the interval ($-\pi$, π].

[A] Optional fields with an asterisk have additional constraints. See text for details.

TABLE 28 Child Elements for the IndexBounds Structure

Element Name	Type	Required/ Optional[A]	Description
rowMinimum	Integer	optional*	The minimum rowIndex value of any point represented by this IndexBounds object.
rowMaximum	Integer	optional*	The maximum rowIndex value of any point represented by this IndexBounds object.
columnMinimum	Integer	optional*	The minimum columnIndex value of any point represented by this IndexBounds object.
columnMaximum	Integer	optional*	The maximum columnIndex value of any point represented by this IndexBounds object.
returnMinimum	Integer	optional*	The minimum returnIndex value of any point represented by this IndexBounds object.
returnMaximum	Integer	optional*	The maximum returnIndex value of any point represented by this IndexBounds object.

[A] Optional fields with an asterisk have additional constraints. See text for details.

TABLE 29 Child Elements for IntensityLimits Structure

Element Name	Type	Required/ Optional	Description
intensityMinimum	Float/ScaledInteger/Integer	required	The minimum producible intensity value. Unit is unspecified.
intenstiyMaximum	Float/ScaledInteger/Integer	required	The maximum producible intensity value. Unit is unspecified.

8.4.20.1 A ColorLimits Structure specifies the limits for the value of red, green, and blue color that a sensor is capable of producing. In this section, the term `points` refers to the points element of the parent Data3D Structure for this ColorLimits Structure, and the term PointRecord refers to the corresponding PointRecord defined by the `prototype` element of `points`. See Table 30.

8.4.20.2 The `colorRedMinium` and `colorRedMaximum` child elements of the `colorLimits` element shall correspond to the minimum and maximum `colorRed` values that the sensor is capable of producing in the configuration used to collect the data stored in `points`. Similarly, the `colorGreenMinimum`, `colorGreenMaximum`, `colorBlueMinimum` , and `colorBlueMaximum` child elements

TABLE 30 Child Elements for the ColorLimits Structure

Element Name	Type	Required/ Optional	Description
colorRedMinimum	Float/ScaledInteger/Integer	required	The minimum producible red color value. Unit is unspecified.
colorRedMaximum	Float/ScaledInteger/Integer	required	The maximum producible red color value. Unit is unspecified.
colorGreenMinimum	Float/ScaledInteger/Integer	required	The minimum producible green color value. Unit is unspecified.
colorGreenMaximum	Float/ScaledInteger/Integer	required	The maximum producible green color value. Unit is unspecified.
colorBlueMinimum	Float/ScaledInteger/Integer	required	The minimum producible blue color value. Unit is unspecified.
colorBlueMaximum	Float/ScaledInteger/Integer	required	The maximum producible blue color value. Unit is unspecified.

of the colorLimits element shall correspond to the minimum and maximum producible colorGreen and colorBlue values respectively.

8.4.20.3 The units of colorRedMinimum and colorRedMaximum are unspecified, but they shall be the same as the units used for the colorRed element in the PointRecord. Similarly, the units of colorGreenMinimum , colorGreenMaximum, colorBlueMinimum, and colorBlueMaximum are unspecified, but they shall be the same as the units used for the corresponding colorGreen and colorBlue elements in the PointRecord.

8.4.20.4 It is recommended that the element type of colorRedMinimum and colorRedMaximum be the same as the element type of the colorRed element in the PointRecord. Similarly, it is recommended that the element type of colorGreenMinimum, colorGreenMaximum, colorBlueMinimum, and colorBlueMaximum be the same as the element type of the corresponding colorGreen and colorBlue elements in the PointRecord.

8.4.20.5 The colorRed values stored in the points shall be restricted to the range [colorRedMinimum, colorRedMaximum]. Similarly, the colorGreen and colorBlue values shall be restricted to the range [colorGreenMinimum, colorGreenMaximum] and [colorBlueMinimum, colorBlueMaximum] respectively.

8.4.20.6 *Discussion*—The purpose of the colorLimits element is to facilitate scaling of the color values of the stored points such that they are comparable across different data sets from the same type of sensor. The colorLimits values should be set based on the range of *possible* measurable values for a sensor in a given configuration, not the *actual* measured values for a particular data set.

8.4.21 *DateTime:*

8.4.21.1 A DateTime Structure encodes date and time. The child elements for the DateTime Structure are listed in Table 31.

8.4.21.2 The date and time are encoded using a single floating point number that is based on the Global Positioning System (GPS) time scale.

8.4.21.3 The dateTimeValue element stores the number of seconds since GPS start epoch, and may include fractional seconds, if needed.

8.4.21.4 *Discussion*—The GPS start epoch occurred at 0 h UTC (12:00 midnight) on January 6, 1980.

8.4.21.5 If a 3D imaging system uses a GPS device to set the time, then the child element isAtomicClockReferenced shall be defined, and its value set to 1. For devices that use another atomic clock reference – such as the GLObal Navigation Satellite System (GLONASS) – the time value shall be converted to GPS time, and isAtomicClockReferenced shall be defined and set to 1. For devices that use an atomic clock reference in a different time scale, such as UTC, the time value shall be converted to GPS time based on the current leap-second offset, and isAtomicClockReferenced shall be defined and set to 1. For devices that have no atomic clock time reference (for example, devices that use an internal crystal-based clock only), or if the time reference is unknown, the isAtomicClockReferenced element shall either be omitted or set to 0.

8.4.22 *Globally Unique Identifiers (GUIDs):*

8.4.22.1 A GUID String stores a value that is unique in any context. In this standard, GUIDs are used to uniquely identify an E57 file or structures within the file.

8.4.22.2 This standard does not specify the format of GUIDs or how they should be generated.

8.4.22.3 A Data3D or Image2D Structure shall be assigned a GUID when it is created. If any change is applied to the Structure (or to any of its descendents), then the Structure is considered to be a new version and should be assigned a new GUID. Similarly, every E57 file shall be assigned a file-level GUID, which is stored in the guid element of the Data3D Structure. If any change is made to the file, this change shall be regarded as creating a new version of the file, and the file shall be assigned a new file-level GUID.

8.4.22.4 *Discussion—Guidance for Generating GUIDs*—A GUID can be generated in several ways. A GUID can be a hexadecimal string representation of a Universally Unique Identifier (UUID), as documented in IETF RFC4122 (http://tools.ietf.org/html/rfc4122). Often a network address (MAC or IP) is used in a UUID to locate the generator uniquely in space. For equipment that is not networked, other schemes are possible. The make/model/serial number of an instrument, in combination with time, can be used to generate a GUID–for example "XYZCorp:ScanMaster:0001234:2009-06-23-T06:10:08.123456" may represent a legal GUID.

8.4.23 *CoordinateMetadata:*

8.4.23.1 A CoordinateMetadata String encodes a geodetic datum, geoid, coordinate system, and map projection for an E57 file to allow the stored Data3D and Image2D data to be referenced in a standardized Coordinate Reference System (CRS).

8.4.23.2 The CRS is specified by a string in well-known text (WKT) format for a spatial reference system as defined by the Coordinate Transformation Service specification developed by the Open Geospatial Consortium (OGC) (http://www.opengeospatial.org).

8.4.23.3 Point data in a Data3D object shall first be transformed into the file-level coordinate system (by applying the

TABLE 31 Child Elements for DateTime Structure

Element Name	Type	Required/ Optional	Description
dateTimeValue	Float	required	The time, in seconds, since GPS start epoch. This time specification may include fractions of a second.
isAtomicClockReferenced	Integer	optional	This element shall be present, and its value set to 1 if, and only if, the time stored in the dateTimeValue element is obtained from an atomic clock time source. Shall be either 0 or 1.

rigid body transform stored the pose element of the Data3D object) before the transformation described by the WKT string is applied.

9. Binary Sections

9.1 *General:*

9.1.1 Binary sections are present in an E57 file whenever the XML section contains any elements of type Blob or CompressedVector. A separate binary section is required for each such element.

9.1.2 There are two types of binary sections: Blob binary sections and CompressedVector binary sections. A binary section shall start on a four-byte boundary within the file and shall be an integral multiple of four bytes in length. If the length of a binary section is not an integral multiple of four bytes, the section shall be padded with up to three zero-valued bytes to achieve an integral multiple of 4 bytes in length.

9.1.3 A binary section shall be up to 2^{63}-1 bytes long, subject to the constraint that the overall file shall be smaller than 2^{63}-1 bytes in length.

9.1.4 Identifiers defined for the different types of binary sections are listed in Table 32.

9.2 *Blob Binary Section:*

9.2.1 A Blob binary section shall contain a Blob type E57 element's binary contents as a sequence of bytes.

9.2.1.1 *Discussion*—See 8.3.6 for details about the XML portion of Blob type E57 elements.

9.2.2 The format for a Blob binary section is given in Table 33.

9.2.3 The number of used bytes in the Blob shall be determined by the `length` XML attribute in the XML part of the Blob representation, not the `sectionLength` field (which includes up to three bytes of zero padding) in the Blob binary section.

9.3 *CompressedVector Binary Section:*

9.3.1 The CompressedVector binary section shall contain the binary portion of a CompressedVector type E57 element.

9.3.1.1 *Discussion*—See 8.3.9 for details about the XML portion of CompressedVector type E57 elements.

9.3.2 A CompressedVector binary section shall be composed of a section header followed by a sequence of variable length binary packets. Numeric identifiers for the three types of binary packets are listed Table 34.

9.3.3 A CompressedVector stores a sequence of zero or more identically typed items, known as records. Each record is numbered sequentially, starting with zero, and increasing by one with each record. The records shall be divided into groups, known as chunks. At least one chunk shall be present in a CompressedVector binary section.

9.3.3.1 *Discussion*—Chunks allow rapid access to a specific record number through the use of an indexing scheme that is described in the next section. The file writer is responsible for determining the appropriate chunk size based on the data, compression algorithm, and application scenario.

9.3.4 The header of a CompressedVector binary section is composed of the fields listed in Table 35.

9.3.5 The length of each binary packet is up to 2^{16} bytes (that is, 64 kibibytes). The physical file offset of the start of each binary packet shall be a multiple of four. The logical length of each binary packet shall be a multiple of four. A sequence of binary packets shall follow the header, each of which shall be one of the defined packet types (index, data, or ignored). A CompressedVector shall contain at least one index packet and at least one data packet.

9.4 *Index Packet:*

9.4.1 The index packets form a database of the beginning locations of chunks. An index packet is composed of a header followed by a sequence of between 1 and 2048 address entries. The format of an index packet header is given in Table 36.

9.4.2 The index packet may be padded at the end with unused address entries. Such entries shall be zero. The total of used plus unused entries shall not exceed 2048.

9.4.3 The format for the index packet address entries is given in Table 37.

9.4.4 The index packets are organized into a tree using the `indexLevel` field in the header, and the tree nodes are connected using the `packetOffset` fields in the address entries. Fig. 6 illustrates the index packet tree concept with an example.

9.4.5 For leaf nodes in the tree, the `indexLevel` field shall be 0. In this case, the `packetOffset` field shall contain the file offset of the first data packet in the chunk, and `chunkRecordIndex` shall contain the record number of the first record stored in the chunk. Only the first data packet in each chunk shall have its `compressorRestart` flag set to 1. The data packet referenced by the `packetOffset` field shall be in the same binary section as the index packet.

9.4.6 For non-leaf nodes in the tree, the `indexLevel` shall be greater than zero. In this case, the `packetOffset` field shall contain the file offset of an index packet at the next lower level in the tree (that is, with its `indexLevel` field one less than this index packet). The `chunkRecordIndex` shall contain the `chunkRecordIndex` of the first entry in the lower level index packet. The lower level packet shall be in the same binary section as upper level packet.

9.4.7 There can be up to six levels of index in a binary section, so $0 \leq$ `indexLevel` ≤ 5.

9.4.8 The ordering of the index packets within the binary section is not constrained. They can be all clustered at the beginning, at the end, or interspersed with data or ignored packets. However, the address entries within an index packet shall be ordered by ascending `chunkRecordIndex` .

9.4.9 At any given level, each index packet shall be full (that is, populated with the maximum number of address entries) before adding a new index packet at that level.

9.4.10 Each level in the index packet tree shall be full (that is, populated with the maximum number of address entries) before starting a new level.

9.5 *Data Packet:*

TABLE 32 Type Identifiers for Binary Sections

sectionId	Value
E57_BLOB_SECTION	0
E57_COMPRESSED_VECTOR_SECTION	1

TABLE 33 Format of a Binary Blob Section

Bytes	Field name	Data type	Description
1	sectionId	Unsigned 8-bit integer	E57_BLOB_SECTION (value = 0).
2-8	reserved	Unsigned 8-bit integers	Reserved bytes for future versions of the standard. Shall all be 0.
9-16	sectionLength	Unsigned 64-bit integer	The logical length of the Blob binary section (in bytes), including any zero padding. Shall be in the interval (0, 2^{63}).
17 to N	blobData	8-bit integer (byte)	The storage space for the data for the blob. The end byte (*N*) is determined by the sectionLength.

TABLE 34 Types of Binary Packets

Packet type	packetTypeId	Value
Index packet	E57_INDEX_PACKET	0
Data packet	E57_DATA_PACKET	1
Ignored packet	E57_IGNORED_PACKET	2

9.5.1 The data packets contain the actual data from the records stored in a CompressedVector. The `codecs` element of a CompressedVector E57 element specifies the algorithm to be used to encode or decode the records stored in the CompressedVector. When encoding CompressedVector data, the data from one or more fields in each record is concatenated together to form a stream of bytes, known hereafter as a bytestream. A bytestream is split into finite length sequences, known hereafter as bytestream buffers. The number of bytestream buffers used to encode a prototype Structure depends on the codec used to compress the given fields. The mapping from fields to bytestreams is detailed in the codec description in 9.7.

9.5.2 A data packet shall be composed of a header, followed by a list of bytestream buffer lengths, followed by the bytestream buffers themselves.

9.5.3 The format of a data packet header is given in Table 38.

9.5.4 A list of bytestream buffer lengths shall follow the header, with one entry corresponding to each bytestream. The number of entries shall match the value of bytestreamCount in the header. Each bytestream buffer length entry shall be stored as an unsigned 16-bit integer.

9.5.5 The bytestream buffers themselves shall follow the list of bytestream buffer lengths. Each data packet in a CompressedVector binary section shall have the same number and order of bytestream buffers. The length of the *N*th bytestream buffer shall match the length given in the *N*th entry in the bytestream buffer length list.

9.5.6 A given bytestream can be reconstructed by concatenating the contents of the corresponding bytestream buffers from each data packet in the order that the data packets occur in the file. It is legal for a data packet to contain 0 bytes of a particular bytestream. Each bytestream buffer shall contain from 0 to 2^{16}-1 bytes. The number of bytestreams used and the format of bytes within a bytestream is codec-dependent and is given in the specification of the codec.

9.5.7 The `packetFlags` field contains bit flags, numbered 1 to 8, with 1 being the least significant bit, with the format given in Table 39. The meaning of these flags is codec-dependent and is given in the specification of the codec.

9.6 *Ignored Packet:*

9.6.1 The ignored packet is used to reserve space in a binary section that may be used in the future. The format of an ignored packet is given in Table 40.

9.6.2 The length of an ignored packet shall be a multiple of four, and shall be $\leq 2^{16}$ bytes (that is, 64 kibibytes). The contents of ignored packets are unconstrained and may be non-zero.

9.7 *BitPack Codec Bytestream Format:*

9.7.1 The bitPack codec encodes E57 elements for storage in a bytestream or decodes a bytestream into E57 elements. The bitPack encoding algorithm does no actual compressing of the data–it just removes unused bits and packs the data efficiently into bytes in the file.

9.7.2 Each input field of a bitPack encoder shall be stored in a separate bytestream in the same order as the fields were declared in the `inputs` Vector. Fields that are not explicitly associated with a codec in the CompressedVector XML element shall default to the bitPack codec. The bytestreams for default codec fields shall stored after all explicit codec bytestreams, in the depth-first order the fields were declared in the CompressedVector's `prototype` Structure. Each E57 element value of a CompressedVector field shall be converted into an integral number of bits using a type-dependant algorithm described below. Bit values from the field in later records shall be added to the most significant end of the accumulated buffer.

9.7.3 If number of bits in a bytestreamBuffer is not a multiple of eight, the last byte of the last data packet shall have the unused most significant bits set to zero. If a data packet has the `compressorRestart` flag set, the previous data packet (if one exists) may contain bytestreamBuffers whose last byte has unused most significant bits, which shall be zero. If a data packet does not have the compressorRestart flag set, the bytestreamBuffers in the previous data packet (if one exists) shall not have any unused bits. The encoding of a single field value may span across two or more data packets, provided that the subsequent data packets do not have the compressorRestart flag set.

9.7.4 The next several subsections document the algorithms for encoding each allowable E57 element type in a bytestream.

9.7.4.1 *Discussion*—Note that Blob and CompressedVector E57 types are not legal within CompressedVectors. Also note that Structure and Vector E57 element types do not directly contain values and, therefore, are not represented in a separate bytestream. Only terminal element types (Integer, ScaledInteger, Float, String) are associated with a bytestream.

9.7.4.2 *Integer and ScaledInteger:*

(1) Integers and ScaledIntegers are encoded using the same algorithm. The only difference is that with Integers, the value

TABLE 35 Fields for the CompressedVector Header

Bytes	Field name	Data type	Description
1	sectionId	Unsigned 8-bit integer	E57_COMPRESSED_VECTOR_SECTION (value = 1).
2-8	reserved	Unsigned 8-bit integers	Reserved bytes for future versions of the standard. Shall all be 0.
9-16	sectionLength	Unsigned 64-bit integer	The logical length of the CompressedVector binary section (in bytes), including any zero padding. Shall be in the interval (0, 2^{63}).
17-24	dataStartOffset	Unsigned 64-bit integer	The file offset of the first data packet in this binary section (in bytes). Shall be in the interval (0, 2^{63}).
25-32	indexStartOffset	Unsigned 64-bit integer	The file offset to the root level index packet in this binary section (in bytes). Shall be in the interval (0, 2^{63}).

TABLE 36 Format of an Index Packet Header

Bytes	Field name	Data type	Description
1	packetType	Unsigned 8-bit integer	E57_INDEX_PACKET (value = 0).
2	reserved	Unsigned 8-bit integer	Reserved for future versions of the standard. Shall all be 0.
3-4	packetLengthMinus1	Unsigned 16-bit integer	One less than the logical length of the packet (in bytes). Shall be in the interval (0, 2^{16}).
5-6	entryCount	Unsigned 16-bit integer	The number of used address entries in this packet. Shall be in the interval [1, 2048].
7	indexLevel	Unsigned 8-bit integer	The level of this index packet in the tree of index packets. The bottom (leaf) level is zero. Shall be in the interval [0, 5].
8-16	reserved	Unsigned 8-bit integers	Reserved bytes for future versions of the standard. Shall all be zero.

TABLE 37 Format of Index Packet Address Entries

Bytes	Field name	Data type	Description
1-8	chunkRecordIndex	Unsigned 64-bit integer	The index of the first record stored in this chunk (for leaf nodes), or the index of the first record stored in any chunks within the sub-tree (for non-leaf nodes). Shall be in the interval [0, 2^{63}).
9-16	packetOffset	Unsigned 64-bit integer	The file offset to a data packet (for leaf nodes) or to a lower level index packet (for non-leaf nodes). Shall be in the interval (0, 2^{63}).

of the Integer is encoded, while with ScaledIntegers the rawValue is encoded. Calculate number of bits (N) required for storing the value using the `minimum` and `maximum` values, which are specified in the prototype for the element, using:

$$N = \text{ceil}(\log_2(maximum - minimum + 1)) \quad (26)$$

where:

ceil(x) = the smallest integer ≥ x.

(2) Given a value *V* (or rawValue for ScaledIntegers), encode (*V*-minimum) in an *N*-bit unsigned binary number.

(3) If $N = 0$, a bytestream shall not be used to encode that element in the binary section.

9.7.4.3 *Float:*

(1) Floating point numbers are encoded in IEEE floating point format (IEEE 754-1985) using 32 bits if `precision` =Single and 64 bits if `precision`=Double. The precision is specified by the prototype for the field. The following values shall not appear in binary section floating point numbers: NaN, +INF, -INF, and -0.

9.7.4.4 *String:*

(1) Strings are encoded using UTF-8 in one of two formats, depending on the length of the string. If *L* is the number of bytes in the 8 bit UTF-8 encoding of the string, excluding any null terminator, then for

(a) $0 \le L \le 127$, the string shall be encoded in 8+8L bits in the format given in Table 41.

(b) $128 \le L \le 2^{63}$-1, the string shall be encoded in 64+8L bits in the format given in Table 42.

(2) If the string is null terminated, the null terminator shall not be included in the encoded `stringData` .

9.7.5 *Discussion*—For illustration, an example encoding is described in the following subsections.

9.7.5.1 Consider a CompressedVector with the following prototype:

```
<prototype type="Structure">
 <valid type="Integer" minimum="0" maximum="1"/>
 <x type="Integer" minimum="0" maximum="15"/>
 <y type="Integer" minimum="0" maximum="255"/>
 <z type="Integer" minimum="0" maximum="4095"/>
</prototype>
```

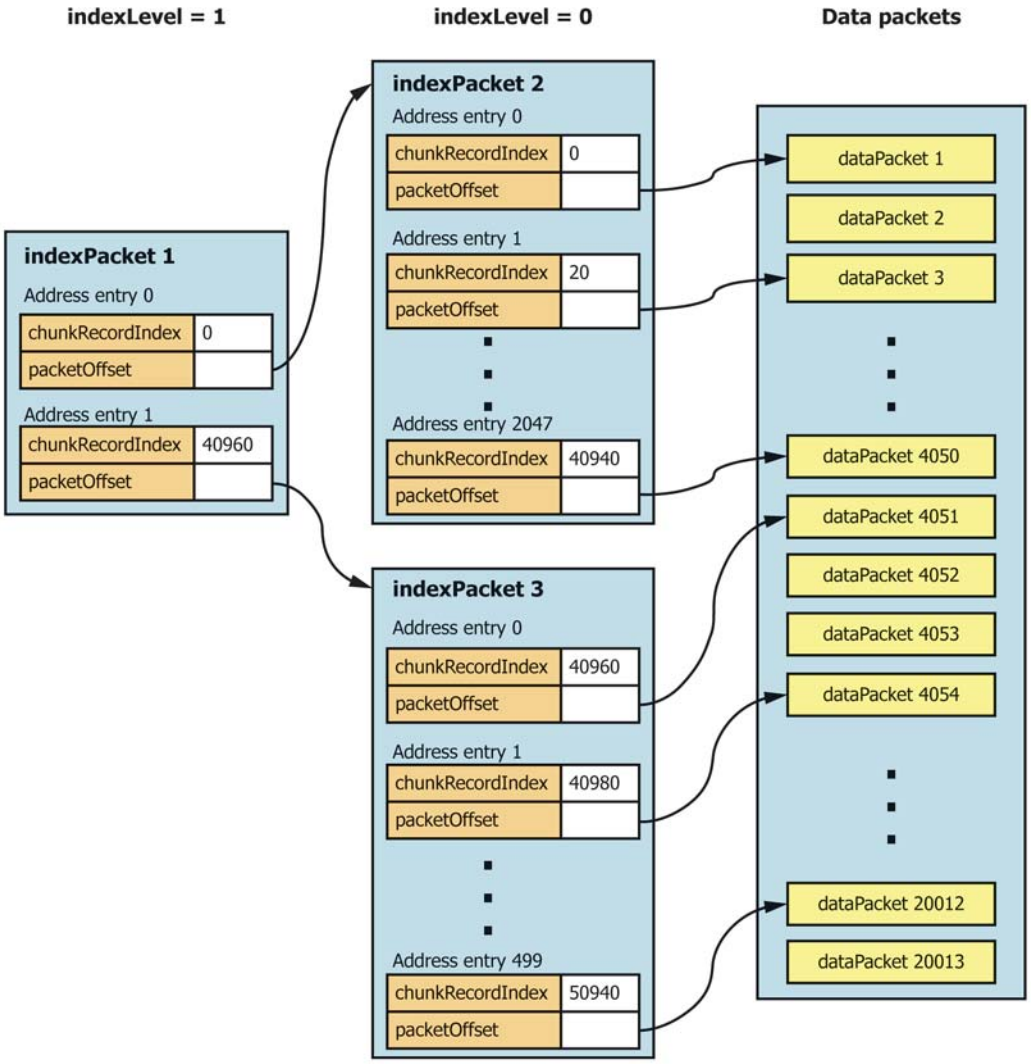
9.7.5.2 Based on this prototype, the `valid`, `x`, `y`, and `z` fields require 1, 4, 8, and 12 bits of storage, respectively (using *N* calculated from Eq 26). A bitPack codec is given the five records to be encoded shown in Table 43. (Note: field values are given in hexadecimal.)

9.7.5.3 The bitPack encoding of this data is stored in four bytestreams as shown in Fig. 7.

10. Extensions

10.1 Extensions are designed to support forward compatibility so that an older reader of the E57 file format will be able to handle files created by a new and improved writer. Note that backward compatibility is easily maintained by newer readers, which can be programmed to read older, known versions of the E57 file format.

10.2 Extensions are supported through the use of XML namespaces. Every extension to the E57 file format shall define



NOTE 1—This example contains three index packets, which index 2548 chunks. For the non-leaf nodes (indexPacket 1), the packetOffset fields link to the index packets in the next lower level of the tree (indexPackets 2 and 3).

FIG. 6 An Example of a Two-Level Tree of indexPackets

TABLE 38 Format of a Data Packet Header

Bytes	Field name	Data type	Description
1	packetType	Unsigned 8-bit integer	E57_DATA_PACKET (value = 1).
2	packetFlags	Unsigned 8-bit integer	Packet flag bit fields (described in 9.5.7).
3-4	packetLengthMinus1	Unsigned 16-bit integer	One less than the logical length of the packet (in bytes). To maintain alignment, the packet may be padded with up to three zero-valued bytes after the last used field. The length includes any zero padding that is present. Shall be in the interval (0, 2^{16}).
5-6	bytestreamCount	Unsigned 16-bit integer	The number of bytestreams in this packet. Shall be in the interval [0, 32763].

a unique namespace. The namespace for an extension shall be declared in an XML attribute of the E57Root element as described in 8.4.2. New element names defined by an extension shall be prefixed by this namespace identifier followed by a colon.

10.3 A reader of an E57 file may ignore any element with a tag name in an XML namespace that it does not recognize, including all sub-elements contained within such an element. If a reader encounters an unknown element tag name in a known namespace, then a serious error has occurred and the reader may abort.

10.3.1 *Discussion*—The ability to ignore unknown elements enables E57 file readers to be forward compatible.

10.4 An extension shall contain the following items:

TABLE 39 Format for the packetFlags Field

Bit field	Field name	Description
1	compressorRestart	At the beginning of this packet, reinitialize state of all the compressors/decompressors, discarding any adaptive statistics that have been gathered about the data being compressed/decompressed.
2-8	reserved	Reserved for future versions of the standard. Shall all be zero.

TABLE 40 Format of an Ignored Packet

Bytes	Field name	Data type	Description
1	packetType	Unsigned 8-bit integer	E57_IGNORED_PACKET (value = 2).
2	reserved	Unsigned 8-bit integer	Reserved for future versions of the standard. Shall be zero.
3-4	packetLengthMinus1	Unsigned 16-bit integer	One less than the logical length of the packet (in bytes). Shall be in the interval $(0, 2^{16})$.

TABLE 41 String Encoding for Short Strings

Bit field	Field name	Description
1	stringType	Set to 0 to indicate a short string.
2 to 8	length	L encoded as a 7-bit unsigned integer.
9 to 8+8L	stringData	The UTF-8 string data values stored as unsigned 8-bit integers.

TABLE 42 String Encoding for Long Strings

Bit field	Field name	Description
1	stringType	Set to 1 to indicate a long string.
2 to 64	length	L encoded as a 63-bit unsigned integer. Shall be in the interval [128, 2^{63}).
65 to 64+8L	stringData	The UTF-8 string data values stored as unsigned 8-bit integers.

TABLE 43 Example Records to be Encoded by the bitPack Codec

Record	valid	x	y	z
1	1_{16}	0_{16}	10_{16}	0560_{16}
2	0_{16}	1_{16}	11_{16}	0561_{16}
3	1_{16}	2_{16}	12_{16}	0562_{16}
4	0_{16}	3_{16}	13_{16}	0563_{16}
5	1_{16}	4_{16}	14_{16}	0564_{16}

10.4.1 The XML namespace for the extension, including the URI, which shall be unique.

10.4.2 A text description of the purpose and scope of the extension.

10.4.3 A text description of the E57 elements that are defined by the extension. This description shall define the extension in terms of the E57 elements defined in 8.3 or in terms of other, previously defined extensions. Each element shall be fully documented in a manner consistent with the documentation of the data types in 8.4 including the element name, type, whether it is required or optional, and a description of the element and any limitations on its content or usage. All element names shall be legal XML tag names. It is recommended that element names be restricted to the following character set: a-z, A-Z, 0-9, "_", "-". An E57 element name shall not begin with the three characters "xml" in any combination of capitalization.

10.5 The following items are recommended, but not required:

10.5.1 A description of the extension grammar using REgular LAnguage for XML Next Generation (RELAX NG).

10.5.2 Example code to read and write the data associated with the extension (if applicable).

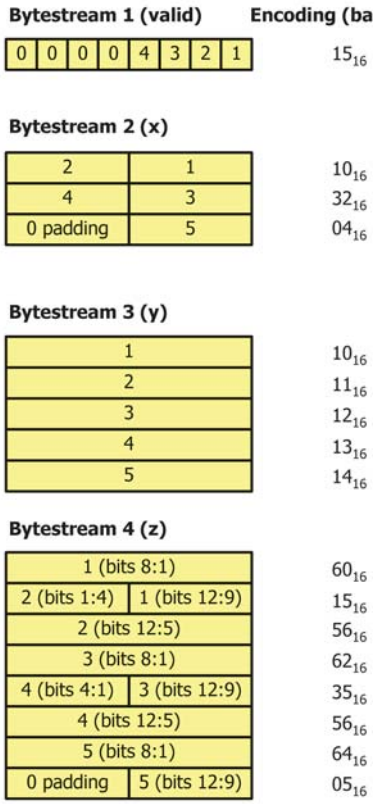
10.5.3 Example E57 files (or file excerpts) that illustrate the extension's format.

10.5.4 A suggested prefix to use in association with the URI, by convention. The prefix shall be a legal XML namespace prefix.

10.6 Extensions shall use SI units for representing quantities for which SI units are defined. Extensions should use SI base or derived units and should avoid using SI prefixes.

11. Keywords

11.1 3D image; file format; LADAR; laser scan; LIDAR; point cloud



NOTE 1—The left column shows the source record from where each bit in the bytestream is drawn. When records are divided among two bytes in the bytestream, the individual bits within the source are indicated (for example, in bytestream 4). The right column shows the encoded bytes that result from the encoding of the source data shown in the left column.

FIG. 7 Bytestream Encoding of the Example Data

APPENDIX

(Nonmandatory Information)

X1. EXAMPLE XML

```
<?xml version="1.0" encoding="UTF-8"?>
<e57Root type="Structure"
        xmlns:demo="http://www.example.com/DemoExtension"
        xmlns="http://www.astm.org/COMMIT/E57/2010-e57-v1.0">
<formatName type="String"><![CDATA[ASTM E57 3D Imaging Data File</formatName>
<guid type="String"><![CDATA[3F2504E0-4F89-11D3-9A0C-0305E82C3300]]></guid>
<versionMajor type="Integer"> 1</versionMajor>
<versionMinor type="Integer"/>
<coordinateMetadata type="String"> <![CDATA[...A WKT string here...]]></coordinateMetadata>
<creationDateTime type="Structure">
  <dateTimeValue type="Float">1.23456e+002</dateTimeValue>
</creationDateTime>
<data3D type="Vector" allowHeterogeneousChildren="1">
  <vectorChild type="Structure">
    <guid type="String"><![CDATA[3F2504E0-4F89-11D3-9A0C-0305E82C3301]]> </guid>
    <demo:extra1 type="String"><![CDATA[used by demo extension]]></demo:extra1>
    <points type="CompressedVector" fileOffset="68" recordCount="10">
      <prototype type="Structure">
        <cartesianX type="ScaledInteger" minimum="0" maximum="32767" scale="1e-003"/>
        <cartesianY type="ScaledInteger" minimum="0" maximum="32767" scale="1e-003"/>
        <cartesianZ type="ScaledInteger" minimum="0" maximum="32767" scale="1e-003"/>
        <cartesianInvalidState type="Integer" minimum="0" maximum="2"/>
        <rowIndex type="Integer" minimum="0" maximum="1"/>
        <columnIndex type="Integer" minimum="0" maximum="4"/>
        <returnIndex type="Integer" minimum="0" maximum="0"/>
        <returnCount type="Integer" minimum="1" maximum="1">1</returnCount>
```

```
    <timeStamp type="Float"/>
    <intensity type="Integer" minimum="0" maximum="255"/>
    <colorRed type="Float" precision="single" minimum="0" maximum="1"/>
    <colorGreen type="Float" precision="single" minimum="0" maximum="1"/>
    <colorBlue type="Float" precision="single" minimum="0" maximum="1"/>
    <demo:extra2 type="String"/>
    </prototype>
    <codecs type="Vector" allowHeterogeneousChildren="1">
    </codecs>
</points>
<pose type="Structure">
    <rotation type="Structure">
        <w type="Float">1</w>
        <x type="Float"/>
        <y type="Float"/>
        <z type="Float"/>
    </rotation>
    <translation type="Structure">
        <x type="Float"/>
        <y type="Float"/>
        <z type="Float"/>
    </translation>
</pose>
<pointGroupingSchemes type="Structure">
    <groupingByLine type="Structure">
        <idElementName type="String">< ![CDATA[columnIndex]]></idElementName>
        <groups type="CompressedVector" fileOffset="464" recordCount="5">
            <prototype type="Structure">
                <idElementValue type="Integer" minimum="0" maximum="4"/>
                <startPointIndex type="Integer" minimum="0" maximum="9"/>
                <pointCount type="Integer" minimum="1" maximum="2">1</pointCount>
                <cartesianBounds type="Structure">
                    <xMinimum type="Float"/>
                    <xMaximum type="Float"/>
                    <yMinimum type="Float"/>
                    <yMaximum type="Float"/>
                    <zMinimum type="Float"/>
                    <zMaximum type="Float"/>
                </cartesianBounds>
            </prototype>
            <codecs type="Vector" allowHeterogeneousChildren="1">
            </codecs>
        </groups>
    </groupingByLine>
</pointGroupingSchemes>
<name type="String"><![CDATA[Station 3F]]></name>
<description type="String"><![CDATA[An example scan]]></description>
<cartesianBounds type="Structure">
    <xMinimum type="Float"/>
    <xMaximum type="Float"> 1</xMaximum>
    <yMinimum type="Float"/>
    <yMaximum type="Float"> 1</yMaximum>
    <zMinimum type="Float"/>
    <zMaximum type="Float"> 1</zMaximum>
</cartesianBounds>
<indexBounds type="Structure">
    <rowMinimum type="Integer"/>
    <rowMaximum type="Integer"> 1</rowMaximum>
    <columnMinimum type="Integer"/>
    <columnMaximum type="Integer"> 4</columnMaximum>
    <returnMinimum type="Integer"/>
    <returnMaximum type="Integer"/>
</indexBounds>
<intensityLimits type="Structure">
    <intensityMinimum type="Integer">0</intensityMinimum>
    <intensityMaximum type="Integer">255</intensityMaximum>
</intensityLimits>
<colorLimits type="Structure">
    <colorRedMinimum type="Float" precision="single" >0</colorRedMinimum>
    <colorRedMaximum type="Float" precision="single" >1</colorRedMaximum>
    <colorGreenMinimum type="Float" precision="single" >0</colorGreenMinimum>
    <colorGreenMaximum type="Float" precision="single" >1</colorGreenMaximum>
    <colorBlueMinimum type="Float" precision="single" >0</colorBlueMinimum>
    <colorBlueMaximum type="Float" precision="single" >1</colorBlueMaximum>
</colorLimits>
<acquisitionStart type="Structure">
    <dateTimeValue type="Float"> 1.235e+003</dateTimeValue>
</acquisitionStart>
```

```
      <acquisitionEnd type="Structure">
        <dateTimeValue type="Float"> 1.235e+003</dateTimeValue>
      </acquisitionEnd>
      <sensorVendor type="String"><![CDATA[Scan Co]]></sensorVendor>
      <sensorModel type="String"><![CDATA[Scanmatic 2000]]></sensorModel>
      <sensorSerialNumber type="String"> <![CDATA[123-321]]>< /sensorSerialNumber>
      <sensorHardwareVersion type="String"> <![CDATA[3.3.0]]></sensorHardwareVersion>
      <sensorSoftwareVersion type="String"> <![CDATA[27.0.3]]></sensorSoftwareVersion>
      <sensorFirmwareVersion type="String"> <![CDATA[27.0.3]]></sensorFirmwareVersion>
      <temperature type="Float"> 2e+001</temperature>
      <relativeHumidity type="Float"> 4e+001</relativeHumidity>
    </vectorChild>
  </data3D>
  <images2D type="Vector" allowHeterogeneousChildren="1">
  <vectorChild type="Structure">
    <guid type="String"><![CDATA[3F2504E0-4F89-11D3-9A0C-0305E82C3302 ]]></guid>
    <pinholeRepresentation type="Structure">
      <imageWidth type="Integer">1024</imageWidth>
      <imageHeight type="Integer">1024</imageHeight>
      <focalLength type="Float"> 1</focalLength>
      <pixelWidth type="Float"> 1e-003</pixelWidth>
      <pixelHeight type="Float"> 1e-003</pixelHeight>
      <principalPointX type="Float"> 5.12e+002</principalPointX>
      <principalPointY type="Float"> 5.12e+002</principalPointY>
      <jpegImage type="Blob" fileOffset="40" length="10"/>
    </pinholeRepresentation>
    <pose type="Structure">
      <rotation type="Structure">
        <w type="Float"> 1</w>
        <x type="Float"/>
        <y type="Float"/>
        <z type="Float"/>
      </rotation>
      <translation type="Structure">
        <x type="Float"/>
        <y type="Float"/>
        <z type="Float"/>
      </translation>
    </pose>
    <name type="String"><![CDATA[pic123]]></name>
    <description type="String"><![CDATA[trial picture]]></description>
    <associatedData3DGuid type="String"> <![CDATA[3F2504E0-4F89-11D3-9A0C-0305E82C3301]]></associatedData3DGuid>
    <acquisitionDateTime type="Structure">
        <dateTimeValue type="Float"> 1.23456e+002</dateTimeValue>
      </acquisitionDateTime>
    </vectorChild>
  </images2D>
</e57Root>
```