Designation: E2807 − 11 (Reapproved 2019)$^{\varepsilon 1}$

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.

ε^1 NOTE—A spelling was corrected in Table 29 and Fig. 7 was updated editorially to reflect data in Table 43 in November 2022.

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, health, and environmental 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.*

1.7 *This international standard was developed in accordance with internationally recognized principles on standardization established in the Decision on Principles for the Development of International Standards, Guides and Recommendations issued by the World Trade Organization Technical Barriers to Trade (TBT) Committee.*

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.

[1] This specification is under the jurisdiction of ASTM Committee E57 on 3D Imaging Systems and is the direct responsibility of Subcommittee E57.11 on Data Interoperability.

Current edition approved March 1, 2019. Published March 2019. Originally approved in 2011. Last previous edition approved in 2011 as E2807 – 11. DOI: 10.1520/E2807-11R19E01.

[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/.

3.2.5 *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.

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):

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

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

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:

$$x = r\cos(\varphi)\cos(\theta) \tag{10}$$

$$y = r\cos(\varphi)\sin(\theta) \tag{11}$$

$$z = r\sin(\varphi) \tag{12}$$

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

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

$$\theta = \operatorname{atan2}(y,x) \tag{14}$$

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

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:

$$\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p} \tag{16}$$

where:

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

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:

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

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.

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

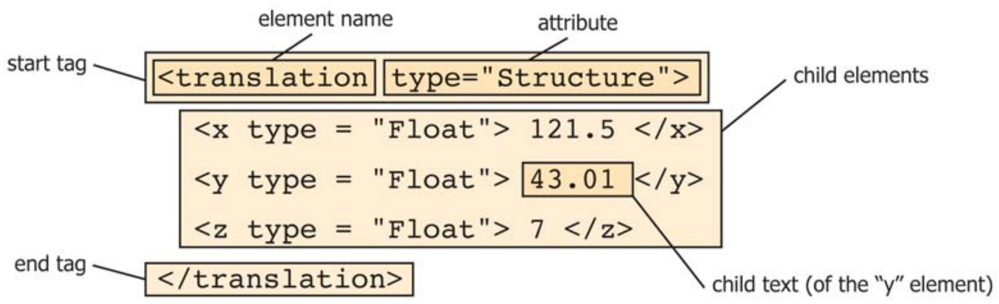

element name
attribute
start tag
<translation type="Structure">
<x type = "Float"> 121.5 </x>
<y type = "Float"> 43.01 </y>
<z type = "Float"> 7 </z>
</translation>
child elements
end tag
child text (of the "y" element)

FIG. 1 XML Elements and Attributes

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

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.

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.

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

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

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.

[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}).

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. 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 descendant elements recursively

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”.

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"
```

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}).

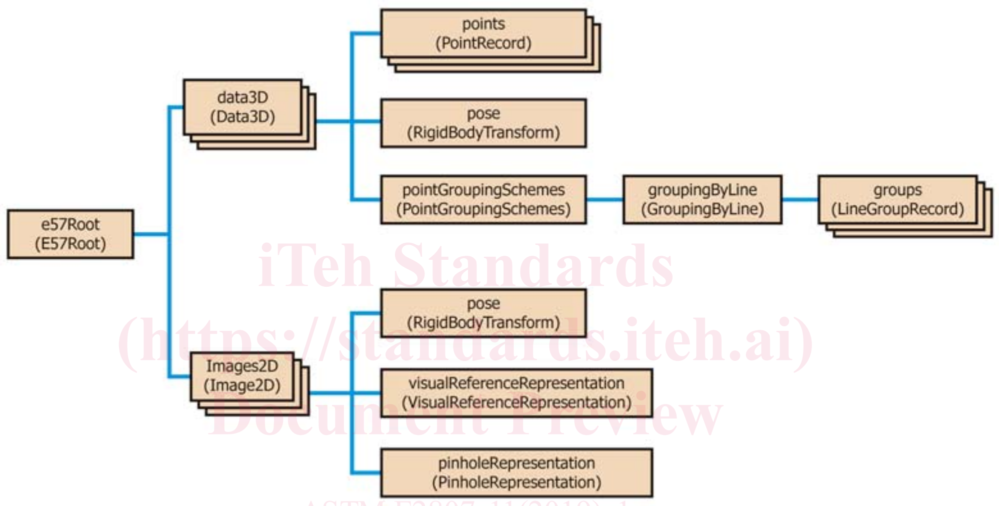
TABLE 10 Child Elements for a CompressedVector Type E57 Element

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.


points (PointRecord)
data3D (Data3D)
pose (RigidBodyTransform)
pointGroupingSchemes (PointGroupingSchemes)
groupingByLine (GroupingByLine)
groups (LineGroupRecord)
e57Root (E57Root)
pose (RigidBodyTransform)
Images2D (Image2D)
visualReferenceRepresentation (VisualReferenceRepresentation)
pinholeRepresentation (PinholeRepresentation)

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.

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 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 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

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.

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,

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.