

STANDARD**27 February 2014****Multi-Dimensional Array Pack**

1 Scope

This Standard (ST) describes the method for formatting multi-dimensional arrays of data in KLV (Key Length Value). A multi-dimensional array is used to store and organize related data, and typically the array is processed as a unit of data (i.e. matrixes, etc.). This Standard provides a method using Packs to format a multi-dimensional array, so that a minimum amount of bytes is used. In addition to providing the structure for storing the elements of an array, this Standard includes options for specifying predefined methods of element processing, such as using MISB ST 1201 (Floating Point to Integer Mapping) for element compression.

The method in this ST requires further context from an invoking document. To provide consistency, a syntax for invoking this ST within MISB documents is described.

2 References

2.1 Normative References

The following references and the references contained therein are normative.

- [1] SMPTE ST 336:2007, Data Encoding Protocol Using Key-Length Value
- [2] MISB ST 1201.1 Floating Point to Integer Mapping, Feb 2014
- [3] MISB RP 0701 Common Metadata Structure, Aug 2007
- [4] MISB ST 0807.13 MISB KLV Metadata Dictionary, Feb 2014
- [5] ISO/IEC, 8825-1:2008 (ITU-T X.690) Information Technology – ASN.1 Encoding Rules - Sub-Identifier defined in Section 8.19.2
- [6] IEEE, 754-2008 Standard for Floating-Point Arithmetic [and Floating-Point formats]

3 Modifications and Changes

Revision	Date	Summary of Changes
Initial Draft	02/27/2014	<ul style="list-style-type: none">• Initial Draft of ST 1303

4 Definitions and Acronyms

IMAPA	Integer Mapping Starting Point A; defined in MISB ST 1201
IMAPB	Integer Mapping Starting Point B; defined in MISB ST 1201
KLV	Key-Length-Value
MISB	Motion Imagery Standards Board
MISP	Motion Imagery Standards Profile
ST	Standard

5 Introduction

During storage, transmission and processing of data within programs, data is organized in many ways including single values, record structures, and arrays. Singular values and record structures are easily converted to KLV data formats through the use of Sets and Packs. When formatting singular values with KLV, the value is in accordance with its type, name, etc., which is registered in a KLV dictionary. A set or pack is a collection of values that can be used to form a record structure. Formatting arrays of data is not formally specified in the KLV standard [1]; therefore, this Standard defines an efficient and flexible method for formatting multi-dimensional arrays of KLV Values (the Value component of the KLV construct) for the MISB community.

When transferring an array of values, support information is needed to ensure that the organization of the data is consistent between the sender and receiver of data. Arrays of data are composed of multiple elements and the elements can be organized with one or more dimensions. An invoking document of this Standard describes the number of dimensions, the size of each dimension, the element size, and the linkage of the elements to the KLV dictionary. The elements in an array can be of any type (e.g. integers, floating point values, strings, KLV pack or set values, etc.), which is defined by the linkage to the KLV dictionary; the only requirement is that all of the elements use the same number of bytes. When elements are of different sizes then the elements need to be normalized to the same length by either inflating or shrinking selected elements (see Appendix B). When grouping data into an array there is an opportunity to use the information about the group to reduce the number of bytes needed for the array. For example, if an array is composed of integers that are originally defined as four bytes, yet all the values in the array only need one byte then the array can be formed from one-byte elements instead of four.

Requirement	
ST 1303-01	All elements in an Array shall use the same number of bytes.
ST 1303-02	The invoking standard shall specify the number of dimensions of the Array.
ST 1303-03	The invoking standard shall specify the size of each dimension or how to compute the size of each dimension in the Array.
ST 1303-04	The invoking standard shall specify the data contained in the Array using KLV Keys or KLV symbols.

Figure 1, Figure 2 and Figure 3 show illustrations of a single dimension, two-dimensional and three-dimensional array respectively.



Figure 1: Illustration of a One-Dimensional Array of N Elements.

Elements in a one-dimensional array are indexed only by columns, so column 0 is element 0, etc.

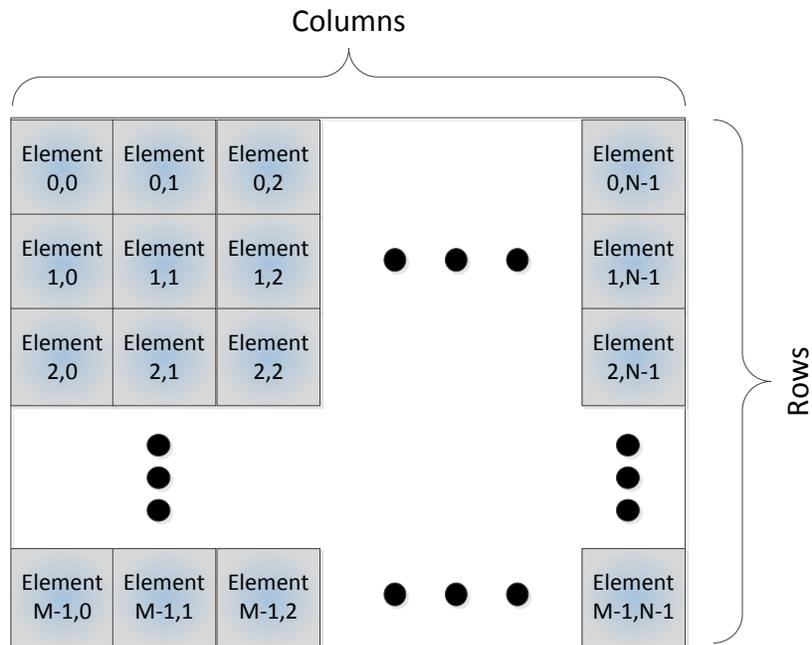


Figure 2: Illustration of a Two-Dimensional Array of M Rows by N Columns

Elements in two-dimensional arrays are indexed by the row and column of the element.

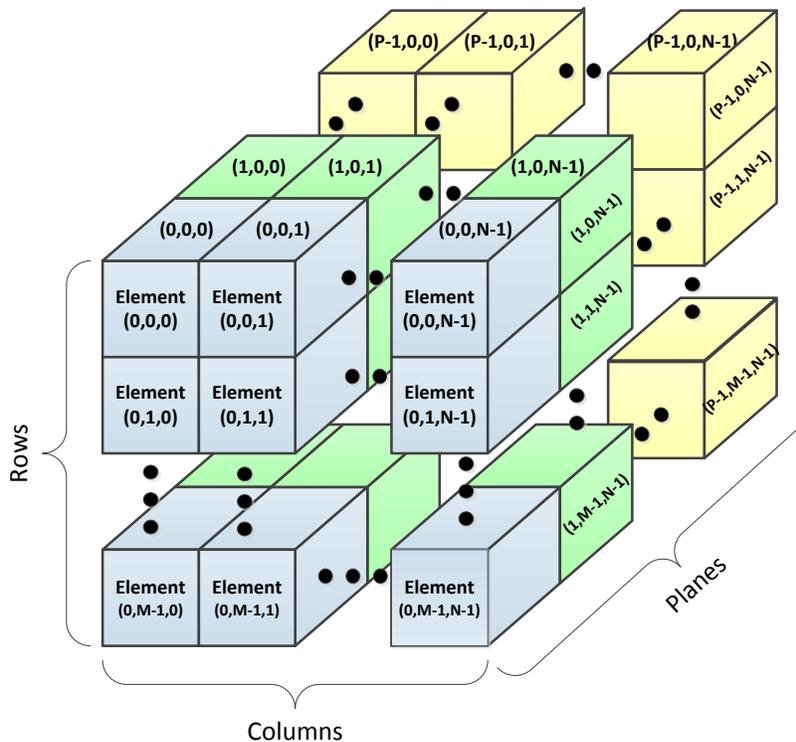


Figure 3: Illustration of a Three-Dimensional Array of P Planes by M Rows by N Columns

Elements in three-dimensional arrays are indexed by planes, rows and columns. Each plane is shown as a separate colored two-dimensional array in Figure 3.

There are two types of elements that can be placed into the array: Dictionary and Processed. Dictionary elements match the type and bit structure as defined by the KLV dictionaries. Processed elements are manipulated before they are inserted into the array. The manipulation is typically a method of compression; such an example is using MISB ST 1201[2] to compress floating point values before they are inserted into the array. Processed elements require additional support information to specify the processing method along with any parameters needed to do the processing. Each processing method, called an Element Processing Algorithm (EPA), is assigned a value, which is documented in Table 4 of Appendix C.

6 Multi-Dimensional Array Truncation Pack

To provide the most bit-efficient method for transmitting the array, plus its required support information and its optional EPA information, a KLV Truncation Pack (See MISB RP 0701 [3]) is used. The parameters of the pack structure are show in Table 1.

Table 1: Multidimensional Array Truncation Pack

Name	Required Optional	Type	Min Value	Description
N _{Dim}	Required	BER-OID Subidentifier Integer	1	Number of dimensions of the Array
Dim _i	Required	BER-OID Subidentifier Integer	1	Size of i th dimension - There are N _{Dim} number of these values
E _{Bytes}	Required	BER-OID Subidentifier Integer	1	Number of bytes for each element in array
EPA	Required	BER-OID Subidentifier Integer	1	Element Processing Algorithm
EPAS	Optional	Defined by EPA	N/A	Element Processing Algorithm Support
Array of Elements	Optional	Defined by Invoking Document	N/A	Array of data, serialized into a one-dimensional array using row major ordering

Requirement	
ST 1303-05	A Multidimensional Array Truncation Pack shall have its mandatory parameters N _{DIM} , Dim _i , Element Size and Element Processing Algorithm (EPA) and its optional elements ordered as defined in MISB ST 1303 Table 1: Multidimensional Array Truncation Pack.

Figure 4 illustrates the Multidimensional Array Truncation Pack, along with its Key and Length. The parameters in blue are the required support information (see Table 2); the yellow EPA Support parameters are optional; the dark green Array of Elements is the array data, which is also optional.



Figure 4: Illustration of a Multidimensional Pack

Table 2 lists the minimum data needed to form a Multidimensional Array Pack.

Table 2: Support information needed to send an array of data

Name	Description
N _{Dim}	Number of dimensions of the Array
Dim _i	Size of the i th dimension, there are N _{Dim} number of dimensions defined. There is always at least one dimension so this value will always be defined.
E _{Bytes}	Number of bytes for each element - the minimum value for this number is zero
EPA	Indicator of Element Processing Algorithm to perform on each array element

The following sections describe the components of the Multidimensional Truncation Pack.

6.1 Multidimensional Array Truncation Pack - Key

Requirement	
ST 1303-06	The 16 byte Key for the Multidimensional Array Truncation Pack shall be 06.0E.2B.34.02.05.01.01.0E.01.03.03.06.00.00.00 (CRC 39697).

This key is registered in the MISB ST 0807 [4] dictionary with the symbol name of flp_multidimensional_array.

6.2 Multidimensional Array Truncation Pack - Length

The Length of the Pack is BER length encoded as specified in [1]. The length consists of the number of bytes for the N_{Dim}, all the dimensions (Dim₁...Dim_N), the Element Size, the EPA, the EPAS and the Array itself. The Length is computed as follows:

$$\text{Pack}_{\text{Length}} = L(N_{\text{Dim}}) + L(\text{Dim}_1) + \dots + L(\text{Dim}_N) + L(\text{Element Size}) + L(\text{EPA}) + L(\text{EPAS}) + L(\text{Array})$$

Where L(x) is the length of value x in bytes.

As discussed below, the L(EPAS) and L(Array) can both be zero depending on usage and circumstances. See Section 10 on interpreting the EPAS if included.

6.3 Multidimensional Array Truncation Pack - Value

The Value of the Multidimensional Array Truncation Pack consists of a collection of required and optional parameters as listed in the following sections. All parameters are required unless noted.

6.3.1 Parameter - N_{Dim}

The N_{Dim} parameter is the count of dimensions in the array of data. For example, a simple list of elements is a one-dimensional array; therefore N_{Dim} equals 1. A rectangle of elements is a two-dimensional array; therefore N_{Dim} equals 2. A cube of elements is a three-dimensional array; therefore N_{Dim} equals 3, etc. A value of zero for N_{Dim} is not allowed.

Requirement	
ST 1303-07	The invoking document shall specify a value for N_{Dim} that is greater than or equal to one (1).
ST 1303-08	The encoding of N_{Dim} shall be BER-OID Subidentifier [5].

Because N_{Dim} is BER-OID Subidentifier [5] encoded the number of bytes is based on the value of the parameter. For example, if the value of this parameter is less than 127 dimensions, then one byte is used, so $L(N_{Dim}) = 1$.

The value of the N_{Dim} parameter determines the number of Dimensional parameters ($Dim_1 \dots Dim_N$) used following the N_{Dim} parameter. For example, if N_{Dim} is 3 (a cube of elements) then Dim_1 , Dim_2 and Dim_3 parameters will be defined following the N_{Dim} parameter.

6.3.2 Parameter - Dim_i

The Dim_i parameter specifies the number of elements in the i^{th} dimension, which is greater than or equal to one (i.e. zero is not allowed). The number of Dim_i parameters is specified by the N_{Dim} value. For example, if a three dimensional cube array of 100x200x300 elements is being formatted then $N_{Dim} = 3$, $Dim_1 = 100$, $Dim_2 = 200$ and $Dim_3 = 300$.

Requirement	
ST 1303-09	The value of Dim_i shall be greater than or equal to one (1).
ST 1303-10	The encoding of each Dim_i shall be BER-OID Subidentifier [5].

Because each of these parameters is BER-OID Subidentifier [5] encoded the number of bytes used for each parameter is based on its value. In the example above this would be 1, 2 and 2 bytes respectively, i.e. $L(Dim_1) = 1$, $L(Dim_2) = 2$, $L(Dim_3) = 2$.

6.3.3 Parameter - Element Size

The Element Size parameter specifies the number of bytes used by each element within an array. For example, if the array consists of single precision floating point numbers, then the Element Size is 4 bytes. Zero is a valid value for the element size. When Element Size equals zero it indicates that each element of the array is of zero length, and therefore, the Array of Elements parameter has no data. An Element Size of zero can be used to signal information without passing any array data; for example, indicating that there was no change in the last set of measurements.

Requirement	
ST 1303-11	The encoding of Element Size shall be BER-OID Subidentifier [5].
ST 1303-12	The value of Element Size shall be greater than or equal to zero (0).
ST 1303-13	An Element Size equal to zero (0) shall signal that the Array data is not included with the Multidimensional Truncation Pack.

Because this parameter is BER-OID Subidentifier encoded [5] the number of bytes needed is based on the value of the parameter. For example, if the Element Size is 4 bytes then only one byte is used i.e. $L(\text{Element Size}) = L(4) = 1$. Since this parameter is BER-OID Subidentifier encoded, Strings longer than 127 bytes can be used as array elements; however, all standard numerical values (i.e. Integers, Unsigned Integer, Floats) will usually be in the range of 1 to 8 bytes.

The values of Element Size along with the values of each Dim_i 's are used in the computation of the length of the Array parameter.

6.3.4 Parameter - Element Processing Algorithm (EPA)

The Element Processing Algorithm (EPA) parameter specifies the method of processing used when forming the elements of the array data, and the method for interpreting the values when parsing data from the array. A value of one (0x01) indicates that no processing is performed, and the data elements of the array match the definitions as specified in KLV dictionary. All other values indicate that some processing is performed on the elements, so they match the definitions in the KLV Dictionary. Section 10 lists the different Element Processing Algorithms and their parameters.

Requirement	
ST 1303-14	Element Processing Algorithm (EPA) parameter value shall be assigned a value only from MISB ST 1303 Table 4: Element Processing Algorithms.

6.3.5 Parameters - Element Processing Algorithm Support (EPAS) (Optional)

The Element Processing Algorithm Support (EPAS) parameters specify values that may be needed for a particular Element Processing Algorithm. The values are specified in the description for each EPA listed in Section 10. If the EPA value is set to one (0x01) then there is no corresponding Element Processing Algorithm Support parameter. The Element Processing Algorithm Support parameters are specified before the array, so that parsers will be able to use the information to interpret the array values as the array is read.

6.3.6 Parameter - Array of Elements (Optional)

The Array of Elements parameter is the multidimensional data being described by all required and optional parameters. The Array is a serialized block of data that contains $\text{Dim}_1 * \text{Dim}_2 * \dots * \text{Dim}_N$ elements, with each element exactly Element Size bytes. If the Element Size is set to zero the array data is considered empty (e.g. all zeros), and it is truncated from the Multidimensional Array Pack.

The total length of the Array in bytes is: $\text{Dim}_1 * \text{Dim}_2 * \dots * \text{Dim}_N * \text{Element Size}$.

To enable accessing the elements within the block of data, the array is serialized in row major order.

Requirement	
ST 1303-15	The data in Array shall be organized in row major order.

Since the block of data is organized in row major order, any element of the Array can be accessed by computing an offset from the start of the Array (see Section 8 - Row Major Computation for more information.)

As is done in many software languages (such as C, C++ or Java) elements in an array are indexed using zero-based notations, so the first element of each dimension is indexed as the zeroth element. For example, Array(10,0) is the first column element of the 11th row for Array(r, c) where r = 10 and c= 0.

7 MISB and MISB Document Standard Notation

When using or “invoking” the Multi-Dimensional Array Pack it is important to be clear as to the type, size and optional packing used.

The following notation will ensure consistency and completeness of the Array definition:

<p>MDARRAY(<Data Identifier>,<N_{Dims}>, <Dim₁>,...,<Dim_N>, [Elem Size])</p> <p>Where:</p> <ul style="list-style-type: none"> Parameters in <.> are required. Parameters in [.] are optional. <Data Identifier> is the KLV Dictionary identifier(s) to use for the elements (see Section 7.1). Other parameters are defined in Section 6.

There are cases when some of the values in the notation will be unknown as an invoking MISB document is written (i.e. values computed at runtime); those values are to be labeled with a foot note (i.e. “Note_x”) that references text which provides a further explanation. Footnotes can be used for multiple parameters and multiple uses of the MDARRAY definition - See Examples in Section 7.2. Element Size can be determined at runtime based on the data values in the array; however, there may be times within an invoking standard where a predetermined size will be required, therefore, the Element Size is optional.

Requirement	
ST 1303-16	When using MISB ST 1303, the invoking standard shall use the MDARRAY(...) notation to ensure consistency and completeness of the Multidimensional Array Truncation Pack definition in accordance with MISB ST 1303.

7.1 Data Identifier

The Data Identifier is used to link the elements of the array to a specific item in a KLV Dictionary. The KLV Dictionary defines the details of the element, such as the type, name, description, etc. There are two Data Identifiers that can be used; either the Key or the Symbol Name. The Key is the 16-byte Key that is associated with all KLV data items. The Symbol name is the unique text identifier that is associated with a KLV data item.

Data Identifiers can be associated to the whole array (homogeneous), or to parts of the array (heterogeneous) as long as the elements all share the same length in bytes. A heterogeneous

example is a grid of points with values representing latitude, longitude and height above ellipsoid (HAE). This data could be represented as a three-dimensional array of points with a first plane used for latitude values, a second plane used for longitude values and a third plane for HAE values as illustrated in Figure 5. This example assumes that all of the latitude, longitude and HAE values use the same number of bytes, e.g. all single precision floating point values.

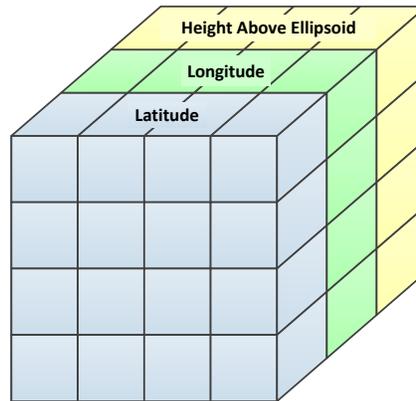


Figure 5: Illustration of 3-D Array with Different Plane Types

When the array is heterogeneous additional footnotes describe the meaning and usage of all of the dimensions. The footnotes describe how the different dimensions are linked to the KLV Dictionary. For example, in Figure 5, Array(0, all rows, all columns) is linked to Latitude; Array(1, all rows, all columns) is linked to Longitude, etc.

When the array is heterogeneous and an Element Processing Algorithm other than direct mapping is used, all of the element values must be compatible with the Element Processing Algorithm. Care must be taken in choosing the EPAS parameters, because the data may have different value ranges. Examples 3, 4, and 5 show three different approaches to defining a heterogeneous array.

7.2 Examples

Example 1: An array of homogeneous Range Depth data.

```
MDARRAY(06.0E.2B.34.01.01.01.01.0E.01.01.03.25.00.00.00, 2, 100, 100)
```

or,

```
MDARRAY(range_depth, 2, 100, 100)
```

This is a two-dimensional array of 100x100 elements of Range Depth data.

Example 2: An array of homogeneous Range Depth data with footnotes describing the “run-time” values.

MDARRAY(06.0E.2B.34.01.01.01.01.0E.01.01.03.25.00.00.00, 2, Note_A, Note_A)

Note_A: This value is dependent on the size of the sensor.

Implementations should use EPA 2, where Min and Max are the bounds of the array data and the element size is determined by IMAPA(Min, Max, 1.0e-4).

This is a two-dimensional array of Range Depth data from a sensor (dimensions known only at runtime - Note_A). The invoker recommends the use of an EPA transformation based on the expected values that will be stored in the array. In this case, the recommendation is to use EPA two (2), so each element’s value is mapped to an integer using IMAPB by using the min/max of the source array and the Element Size. The recommended Element Size is computed based on the IMAPA computation for length using a precision value of 1.0e-4.

Example 3: An array of heterogeneous Latitude/Longitude/Altitude data with footnotes describing the dimensional use and “run-time” values.

MDARRAY(Note_A, 3, Note_B, Note_B)

Note_A: Array(0, r, c) = 06.0E.2B.34.01.01.01.01.07.01.02.01.02.04.00.00^a,

Array(1, r, c) = 06.0E.2B.34.01.01.01.01.07.01.02.01.02.06.00.00^a,

Array(2, r, c) = 06.0E.2B.34.01.01.01.01.07.01.02.01.02.02.00.00^a,

r = all rows and c = all columns.

Note_B: This value is dependent on the size of the sensor.

^athese KLV Keys, taken from the SMPTE dictionary, illustrate the use of 4 byte floats; other KLV Keys in the DoD dictionary (MISB ST 0807) should be used for normal MISB applications.

This is a three-dimensional array of Latitude, Longitude and Altitude data from a sensor (dimensions known only at runtime - Note_B). The first plane (plane 0) is the Latitude, the second plane (plane 1) is the Longitude and the third plane (plane 2) is the Altitude.

This usage is not recommended when applying EPA 2 (ST 1201 Floating Point to Integer Mapping) to reduce the element size because it is impeded by the need to represent values with ranges orders of magnitude apart using a single mapping (latitudes of +/-90° versus HAE of -900 to 19000 meters).

Example 4: The same data of Example 3, but defined as three separate homogeneous arrays.

MDARRAY(06.0E.2B.34.01.01.01.01.07.01.02.01.02.04.00.00^a, 2, Note_A, Note_A)

MDARRAY(06.0E.2B.34.01.01.01.01.07.01.02.01.02.06.00.00^a, 2, Note_A, Note_A)

MDARRAY(06.0E.2B.34.01.01.01.01.07.01.02.01.02.02.00.00^a, 2, Note_A, Note_A)

Note_A: This value is dependent on the size of the sensor.

^athese KLV Keys, taken from the SMPTE dictionary, illustrate the use of 4 byte floats; other KLV Keys in the DoD dictionary (ST0807) should be used for normal MISB applications.

These are three separate arrays: the first array contains Latitude values; the second array contains Longitude values; the third array contains Altitude values. The dimension of each array is defined at runtime from a sensor's information (dimensions known only at runtime - Note_A).

This usage is preferable when applying EPA 2 (MISB ST 1201 Floating Point to Integer Mapping) to reduce the element size because as each array of data would be transformed individually.

Example 5: An example of an array of pack values, using the Location Truncation Pack defined in MISB ST 0903.

MDARRAY(location_pack, 2, rows, cols)

This is a two-dimensional array of location_pack values, with each value potentially defining sub-components: latitude, longitude, height, sigma latitude, sigma longitude, sigma height, Rho lat/lon, Rho lat/height and Rho lon/height (see ST 0903). The element size determines which data is included in each elements pack - all of the elements must be consistently defined to maintain the required fixed element size.

This usage rigidly defines the ranges of each sub-component based on the pack definition so the advantage of adjusting the element size based on the range of data within the array cannot be used.

8 Appendix A - Row Major Computation

The following equations demonstrate how to compute the offsets into an array based on the array dimensions, $D_1, D_2 \dots D_N$ and the desired element indexes $x_1, x_2 \dots x_n$. Each element index ($x_1, x_2 \dots x_n$) is zero based, so the range of x_i is zero to D_i-1 , i.e. $x_i=[0, D_i-1]$. The Element Size of each element in the array is denoted as E_s .

1-Dimensional Array

$$\text{Offset}(\text{column}) = \text{Offset}(x_1) = x_1 E_s$$

2-Dimensional Array

$$\text{Offset}(\text{row}, \text{column}) = \text{Offset}(x_1, x_2) = (x_1 D_2 + x_2) E_s$$

3-Dimensional Array

$$\text{Offset}(\text{plane}, \text{row}, \text{column}) = \text{Offset}(x_1, x_2, x_3) = (x_1 D_2 D_3 + x_2 D_3 + x_3) E_s$$

N-Dimensional Array

$$\text{Offset}(x_1, \dots, \text{plane, row, column}) = \text{Offset}(x_1, x_2, \dots, x_N) = \left(\sum_{i=1}^N x_i \left(\prod_{j=i+1}^N D_j \right) \right) E_s$$

9 Appendix B - Normalizing Element Lengths

The Multi-Dimensional Array Pack requires that all elements within the Array be the same length. This can be accomplished by either reducing or inflating selected elements to the desired length. Reducing data length is possible depending on the application and can be performed as long as data is not lost during the processes. For example, if the data elements are null-padded strings then reducing the length could be performed by removing trailing null characters, as long as they do not provide any meaning.

Data elements that need to be inflated or upsized to match the lengths of larger elements must follow a consistent method of inflating. Each type of data will require a different method as shown in Table 3:

Table 3: Data Inflation Methods

Type	Inflation Method
Signed Integer	Add sign extended bytes by adding additional most significant bytes. The additional bytes shall have all bits zero (0x00) if the most significant bit of the original value is zero (0), or have all bits set (0xFF) if most significant bit of the original value is one (1).
Unsigned Integer	Add additional most significant bytes with zero bytes, until proper size is reached.
32 bit Float	Type cast 32 bit floating point value into a 64-bit floating point value.
Strings	Pad the end of the string with null (0x00) characters.

To deflate the data back to the original size, if known, the Inflation method is performed in reverse.

10 Appendix C - Element Processing Algorithms

Element Processing Algorithms enable packing or other processing on the array elements. Table 4 lists all of the available Element Processing Algorithms.

Table 4: Element Processing Algorithms

EPA Value	Algorithm	Number of Parameters
0x00	Unused	0
0x01	Direct mapping to KLV dictionary - No Element Processing	0
0x02	MISB ST 1201 Floating Point to Integer Mapping - See Section 10.1	2

10.1 Appendix C.1 – MISB ST 1201 Element Processing

When floating point arrays are formatted into KLV there is the option of using MISB ST 1201 [2] to map floating point values to integers to reduce the number of bytes per element. There are two methods specified in ST 1201 for mapping the values: IMAPA and IMAPB. IMAPA uses a minimum, maximum and a precision value to compute the length of a mapped integer value; IMAPB uses a minimum, maximum and a pre-computed length. Since the length of an element is already specified in the array support information, this Standard uses the IMAPB method of ST 1201.

Requirement	
ST 1303-17	When using EPA 0x02, then EPAS value shall contain the Minimum and Maximum values as defined in MISB ST 1303 (Table 5).

Table 5: ST 1201 Minimum/Maximum Values

Name	Description
ST 1201 Minimum Value	The Minimum Value used for mapping and reverse mapping any value in the array to its corresponding packed value - see MISB ST 1201 for further details.
ST 1201 Maximum Value	The Maximum Value used for mapping and reverse mapping any value in the array to its corresponding packed value - see MISB ST 1201 for further details.

The value of Element Size along with the Minimum and Maximum values form the IMAPB mapping parameters for each value in the Array: IMAPB(Minimum, Maximum, Element Size).

Requirement	
ST 1303-18	If EPA is 0x02, then the Array elements shall be encoded as IMAPB(Minimum, Maximum, Element Size).
ST 1303-19	The Minimum and Maximum parameters shall both be the same size: either both 32 bit or both 64 bit IEEE [6] floating point numbers.

The combined length of the Minimum and Maximum parameters is computed from the total length of the pack minus the length of the other parameters (N_{Dim} , Dim_1, \dots, Dim_N , Element Size, EPA and Array), as shown below.

$$\text{MinMax Length} = \text{Pack Length} - (\text{L}(\text{N}_{\text{Dim}}) + \text{L}(\text{Dim}_1) + \dots + \text{L}(\text{Dim}_N) + \text{L}(\text{Element Size}) + \text{L}(\text{EPA}) + \text{L}(\text{Array}))$$

Where:

$$\text{MinMax Length} = \text{L}(\text{Minimum}) + \text{L}(\text{Maximum})$$

L(x) is the length of value x in bytes.

MinMax Length will be either 8 bytes or 16 bytes. When the MinMax length is 8 bytes then the Minimum and Maximum values are single precision, so the first four bytes are the Minimum parameter and the last four bytes are the Maximum parameter. When the MinMax length is 16 bytes the Minimum and Maximum values are double precision, so the first 8 bytes are the Minimum parameter and the last 8 bytes are the Maximum parameter.