Weave TLV Format

Revision History

Revision	Date	Description
4	2013-05-20	Fixed incorrect control byte value for end of container encoding.
3	2013-04-22	Renamed dictionary to structure.
2	2013-04-17	Normalized the naming for ‘container’ types.
1	2013-04-15	Initial revision.

Introduction

This document describes the Weave TLV (*Tag-Length-Value*) format. Weave TLV is a generalized encoding method for simple structured data. It shares many properties with the commonly used JSON serialization format while being considerably more compact over the wire.

TLV Elements and Encodings

Values in the Weave TLV format are encoded as *TLV elements*. Each TLV element has a type. Element types fall into two categories: *primitive types* and *container types*. Primitive types convey fundamental data values such as integers and strings. Container types convey collections of elements that themselves are either primitives or containers. The Weave TLV format supports three different container types: structures, arrays and paths.

All valid *TLV encodings* consist of a single top-level element. This value can be either a primitive type or a container type.

Tags

A TLV element includes an optional numeric tag that identifies its purpose. Two categories of tags are defined: *profile-specific* and *context-specific*. A TLV element without a tag is called an *anonymous* element.

Profile-Specific Tags

Profile-specific tags identify elements globally. A profile-specific tag is a 64-bit number composed of the following fields:

- 16-bit vendor id
- 16-bit profile number
- 32-bit tag number

Profile-specific tags are defined either by Nest or by external vendors. Additionally the Weave Common Profile includes a set of predefined profile-specific tags that can be used across organizations.

Context-Specific Tags

Context-specific tags identify elements within the context of a containing structure element. A context-specific tag consists of a single 8-bit tag number. The meaning of a context specific tag derives from the structure it resides in, implying that the same tag number may have different meanings in the context of different structures. Effectively, the interpretation of a context specific tag depends on the tag attached to the containing element. Because structures themselves can

be assigned context-specific tags, the interpretation of a context-specific tag may ultimately depend on a nested chain of such tags.

Context-specific tags can only be assigned to elements that are immediately within a structure. This implies that an element with a context-specific tag cannot appear as the outermost element of a TLV encoding.

Lengths

Depending on its type, a TLV element may contain a length field that gives the length, in bytes, of the element's value field. A length field is only present for string types (character and byte strings). Other element types either have a predetermined length or are encoded with a marker that identifies their end.

Primitive Types

The Weave TLV format supports the following primitive types:

- Signed integers
- Unsigned integers
- UTF-8 Strings
- Byte Strings
- Single or double-precision floating point numbers (IEEE 754-1985 format)
- Booleans
- Nulls

Of the primitive types, integers, floating point numbers, booleans and nulls have a predetermined length specified by their type. Byte strings and UTF-8 strings include a length field that gives their lengths in bytes.

Container Types

The Weave TLV format supports the following container types:

- Structures
- Arrays
- Paths

Each of the container types is a form of element collection that can contain primitive types and/or other container types. The elements appearing immediately within a container type are called its *members*. A container type can contain any number of member elements, including none. Container types can be nested to any depth and in any combination. The end of a container type is denoted by a special element called the 'end-of-container' element. Although encoded as a member, conceptually the end-of-container element is not included in the members of the containing type.

Structures

A structure is a collection of member elements that each have a distinct meaning. All member elements within a structure must have a unique tag as compared to the other members of the structure. Member elements without tags (anonymous elements) are not allowed in structures. The encoded ordering of members in a structure may or may not be important depending on the intent of the sender or the expectations of the receiver. For example, in some situations, senders and receivers may agree on a particular ordering of elements to make encoding and decoding easier.

Arrays

An array is an ordered collection of member elements that either do not have distinct meanings, or whose meanings are implied by their encoded positions in the array. All member elements of an array must be anonymous elements.

Paths

A path is an ordered collection of member elements that describes how to traverse a tree of TLV elements to arrive at particular element or set of elements. Thus a path forms a kind of name for a TLV element. A path can contain any type of element, including other paths. All member elements of a path must have a tag, however unlike structures these tags needn't be unique with respect to the other members of the path.

Element Encoding

A TLV element is encoded a single control byte, followed by a sequence of tag, length and value bytes. Depending on the nature of the element, any of the tag, length or value fields may be omitted.

Control Byte	Tag	Length	Value
1 byte	0 to 8 bytes	0 to 8 bytes	*Variable*

Control Byte Encoding

The control byte specifies the type of a TLV element and how its tag, length and value fields are encoded. The control byte consists of two subfields: an *element type field* which occupies the lower 5 bits, and a *tag control field* which occupies the upper 3 bits.

Element Type Field

The element type field encodes the element's type as well as how the corresponding length and value fields are encoded. In the case of Booleans and the null value, the element type field also

encodes the value itself.

			Element Type					
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
			0	0	0	0	0	Signed Integer, 1-byte value
			0	0	0	0	1	Signed Integer, 2-byte value
			0	0	0	1	0	Signed Integer, 4-byte value
			0	0	0	1	1	Signed Integer, 8-byte value
			0	0	1	0	0	Unsigned Integer, 1-byte value
			0	0	1	0	1	Unsigned Integer, 2-byte value
			0	0	1	1	0	Unsigned Integer, 4-byte value
			0	0	1	1	1	Unsigned Integer, 8-byte value
			0	1	0	0	0	Boolean False
			0	1	0	0	1	Boolean True
			0	1	0	1	0	Floating Point Number, 4-byte value
			0	1	0	1	1	Floating Point Number, 8-byte value
			0	1	1	0	0	UTF-8 String, 1-byte length
			0	1	1	0	1	UTF-8 String, 2-byte length
			0	1	1	1	0	UTF-8 String, 4-byte length
			0	1	1	1	1	UTF-8 String, 8-byte length
			1	0	0	0	0	Byte String, 1-byte length
			1	0	0	0	1	Byte String, 2-byte length
			1	0	0	1	0	Byte String, 4-byte length
			1	0	0	1	1	Byte String, 8-byte length
			1	0	1	0	0	Null
			1	0	1	0	1	Structure

			1	0	1	1	0	Array
			1	0	1	1	1	Path
			1	1	0	0	0	End of Container
			1	1	0	0	1	*Reserved*
			1	1	0	1	0	*Reserved*
			1	1	0	1	1	*Reserved*
			1	1	1	0	0	*Reserved*
			1	1	1	0	1	*Reserved*
			1	1	1	1	0	*Reserved*
			1	1	1	1	1	*Reserved*

For types that have varying length or value fields, the bottom two bits of the element type field signal the width of the corresponding field as follows:

00 -- 1 byte
01 -- 2 bytes
10 -- 4 bytes
11 -- 8 bytes

Tag Control Field

The tag control field identifies the form of tag assigned to the element (including none) as well as the encoding of the tag bytes.

Tag Control	

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0	0	0						Anonymous, 0 bytes
0	0	1						Context-specific Tag, 1 byte
0	1	0						Common Profile Tag, 2 bytes
0	1	1						Common Profile Tag, 4 bytes
1	0	0						Implicit Profile Tag, 2 bytes

1	0	1						Implicit Profile Tag, 4 bytes
1	1	0						Fully-qualified Tag, 6 bytes
1	1	1						Fully-qualified Tag, 8 bytes

Tag Encoding

Tags are encoded in 0, 1, 2, 4, 6 or 8 byte widths as specified by the tag control field. Tags consist of up to three numeric fields: a *vendor id field*, a *profile number field*, and a *tag number field*. All fields are encoded in little-endian order.

Fully-Qualified Form

A profile-specific tag can be encoded in *fully-qualified form*, where the encoding includes all three tag components (vendor id, profile number and tag number). Two variants of this form are supported, one with a 16-bit tag number and one with a 32-bit tag number. The 16-bit variant must be used with tag numbers < 65536, while the 32-bit variant must be used with tag numbers >= 65536.

Tag Control	Vendor Id Size	Profile Number Size	Tag Number Size	
C0h	2 bytes	2 bytes	2 bytes	For tag numbers < 65536
E0h	2 bytes	2 bytes	4 bytes	For tag numbers >= 65535

Implicit Form

A profile-specific tag can also be encoded in *implicit form*, where the encoding includes only the tag number, and the vendor id and profile number are inferred from the protocol context in which the TLV encoding is communicated. This form also has two variants based on the magnitude of the tag number.

Tag Control	Tag Number Size	
80h	2 bytes	For tag numbers < 65536
A0h	4 bytes	For tag numbers >= 65535

Common Profile Form

A special encoding exists for profile-specific tags that are defined by the Weave Common Profile. These are encoded in the same manner as implicit tags except that they are identified as common profile tags, rather than implicit profile tags in the tag control field.

Tag Control	Tag Number Size	
40h	2 bytes	For tag numbers < 65536
60h	4 bytes	For tag numbers >= 65535

Context-Specific Form

Context-specific tags are encoded as a single byte conveying the tag number.

Tag Control	Tag Number Size	
20h	1 bytes	All tag numbers 0 - 255

Anonymous

Anonymous elements do not encode any tag bytes.

Tag Control	Tag Size	
00h	0 bytes	No data encoded.

Length Encoding

Length fields are encoded in 0, 1, 2 or 4 byte widths, as specified by the element type field. Length fields of more than one byte are encoded in little-endian order. The choice of width for the length field is up to the discretion of the sender, implying that a sender can choose to send more length bytes than strictly necessary to encode the value.

End of Container Encoding

The end of a container type is marked with a special element called the end-of-container element. The end-of-container element is encoded as a single control byte with the value 18h. The tag control bits within the control byte must be set to zero, implying that end-of-container element can never have a tag.

Control Byte
1 byte

Value Encodings

Integers

An integer element is encoded as follows:

Control Byte	Tag	Value
1 byte	0 to 8 bytes	1, 2, 4 or 8 bytes

The number of bytes in the value field is indicated by the element type field within the control byte. The choice of value byte count is at the sender's discretion, implying that a sender is free to send more bytes than strictly necessary to encode the value. Within the value bytes, the integer value is encoded in little-endian two's complement format.

UTF-8 and Byte Strings

UTF-8 and byte strings are encoded as follows:

Control Byte	Tag	Length	Value
1 byte	0 to 8 bytes	1 to 4 bytes	0 to 2^{32}-1 bytes

The length field of a UTF-8 or byte string encodes the number of bytes (not characters) present in the value field. The number of bytes in the length field is implied by the type specified in the element type field (within the control byte).

For UTF-8 strings, the value bytes must encode a valid UTF-8 character sequence. Senders **should not** include a terminating null character to mark the end of a string. For byte strings, the value can be any arbitrary sequence of bytes.

Booleans

Boolean elements are encoded as follows:

Control Byte	Tag
1 byte	0 to 8 bytes

The value of a Boolean element (true or false) is implied by the type indicated in the element type field.

Arrays, Structures and Paths

Array, structure and path elements are encoded as follows:

Control Byte	Tag	Value	End-of-Container
1 byte	0 to 8 bytes	*Variable*	1-byte

The value field of an array/structure/path element is a sequence of encoded TLV elements that constitute the members of the element, followed by an end-of-container element. The end-of-container element must always be present, even in cases where the end of the array/structure/path element could be inferred by other means (e.g. the length of the packet containing the TLV encoding).

Floating Point Numbers

A floating point number is encoded as follows:

Control Byte	Tag	Value
1 byte	0 to 8 bytes	4 or 8 bytes

The value field of a floating point element contains an IEEE 754-1985 single or double precision floating point number encoded in little-endian format. The choice of precision is implied by the type specified in the element type field (within the control byte). The sender is free to choose either precision at their discretion.

Nulls

A null value is encoded as follows:

Control Byte	Tag
1 byte	0 to 8 bytes