6. Conventions

6.1 General

This clause defines various conventions and notation used in the standard, i.e., naming conventions, service specification method and notation, and data type definitions.

6.2 Service specification method and notation

The method and notation for specifying service interfaces is described in Clause 7 of IEEE Std 802.1AC-2016.

6.3 Lexical form syntax

A lexical form refers to the following:

— A name
— A data type

The conventions illustrated in the following list regarding lexical forms are used in this standard:

a) Type names: e.g., ClockQuality (no word separation, initial letter of each word capitalized).
b) Enumeration members and global constants: e.g., ATOMIC_CLOCK (word separation underscored, all letters capitalized).
c) Fields within PTP messages, instances of structures, and variables: e.g., secondsField, clockQuality, clockIdentity (two-word field names at a minimum, no word separation, initial word not capitalized, initial letter capitalization on subsequent words).
d) Members of a structure: e.g., clockQuality.clockClass (no word separation, structure name followed by a period followed by the member name).
e) Data set names: e.g., defaultDS, parentDS, portDS, currentDS, timePropertiesDS (no word separation, initial word not capitalized, initial letter capitalization on subsequent words, end marked by the letters DS).
f) Data set members: e.g., defaultDS.clockQuality.clockClass (data set name followed by a period followed by a member name followed by a period followed by a member name).
g) PTP message names: e.g., Sync, Pdelay_Req (word separation underscored, initial letter of each word capitalized).

When a lexical form appears in text, as opposed to in a type, or a format definition, the form is to be interpreted as singular, plural, or possessive as appropriate to the context of the text.

6.4 Data types and on-the-wire formats

6.4.1 General

The data types specified for the various variables and message fields define logical properties that are necessary for correct operation of the protocol or interpretation of IEEE 1588 precision time protocol (PTP) or IEEE 802.11 message content.

NOTE—Implementations are free to use any internal representation of data types if the internal representation does not change the semantics of any quantity visible via communications using the IEEE 802.1AS protocol or in the specified operations of the protocol.

Copyright © 2020 IEEE. All rights reserved.
Authorized licensed use limited to: Linkoping University Library. Downloaded on November 17,2021 at 03:48:56 UTC from IEEE Xplore. Restrictions apply.

6.4.2 Primitive data types specifications

All non-primitive data types are derived from the primitive types in Table 6-1. Signed integers are represented in twos complement form.

Table 6-1—Primitive data types

Data type	Definition
Boolean	TRUE or FALSE
EnumerationN	N-bit enumerated value
UIntegerN	N-bit unsigned integer
IntegerN	N-bit signed integer
Nibble	4-bit field not interpreted as a number
Octet	8-bit field not interpreted as a number
OctetN	N-octet field not interpreted as a number, with N > 1
Float64 (see NOTE)	IEEE 754™ binary64 (64-bit double-precision floating-point format)
NOTE—The Float64 data type was called Double in the 2011 edition of this standard. The semantics of the data type has not changed.	

6.4.3 Derived data type specifications

6.4.3.1 ScaledNs

The ScaledNs type represents signed values of time and time interval in units of 2^{-16} ns.

typedef Integer96 ScaledNs;

For example: –2.5 ns is expressed as:

0xFFFF FFFF FFFF FFFF FFFD 8000

Positive or negative values of time or time interval outside the maximum range of this data type are encoded as the largest positive or negative value of the data type, respectively.

6.4.3.2 UScaledNs

The UScaledNs type represents unsigned values of time and time interval in units of 2^{-16} ns.

typedef UInteger96 UScaledNs;

For example: 2.5 ns is expressed as:

0x0000 0000 0000 0000 0002 8000

Values of time or time interval greater than the maximum value of this data type are encoded as the largest positive value of the data type.

Copyright © 2020 IEEE. All rights reserved.
Authorized licensed use limited to: Linkoping University Library. Downloaded on November 17,2021 at 03:48:56 UTC from IEEE Xplore. Restrictions apply.

6.4.3.3 TimeInterval

The TimeInterval type represents time intervals, in units of 2^{-16} ns.

typedef Integer64 TimeInterval;

For example: 2.5 ns is expressed as:

0x0000 0000 0002 8000

Positive or negative time intervals outside the maximum range of this data type are encoded as the largest positive and negative values of the data type, respectively.

6.4.3.4 Timestamp

The Timestamp type represents a positive time with respect to the epoch.

```
struct Timestamp
{
        UInteger48 seconds;
        UInteger32 nanoseconds;
};
```

The seconds member is the integer portion of the timestamp in units of seconds.

The nanoseconds member is the fractional portion of the timestamp in units of nanoseconds.

The nanoseconds member is always less than 10^9.

For example:

+2.000000001 seconds is represented by seconds = 0x0000 0000 0002 and nanoseconds= 0x0000 0001

6.4.3.5 ExtendedTimestamp

The ExtendedTimestamp type represents a positive time with respect to the epoch.

```
struct ExtendedTimestamp
{
        UInteger48 seconds;
        UInteger48 fractionalNanoseconds;
};
```

The seconds member is the integer portion of the timestamp in units of seconds.

The fractionalNanoseconds member is the fractional portion of the timestamp in units of 2^{-16} ns.

The fractionalNanoseconds member is always less than $(2^{16})(10^9)$.

For example:

+2.000000001 seconds is represented by seconds = 0x0000 0000 0002 and fractionalNanoseconds = 0x0000 0001 0000

Copyright © 2020 IEEE. All rights reserved.
Authorized licensed use limited to: Linkoping University Library. Downloaded on November 17,2021 at 03:48:56 UTC from IEEE Xplore. Restrictions apply.

6.4.3.6 ClockIdentity

The ClockIdentity type identifies a PTP Instance.

typedef Octet8 ClockIdentity;

6.4.3.7 PortIdentity

The PortIdentity type identifies a port of a PTP Instance.

```
struct PortIdentity
{
        ClockIdentity clockIdentity;
        UInteger16 portNumber;
};
```

6.4.3.8 ClockQuality

The ClockQuality represents the quality of a clock.

```
struct ClockQuality
{
        UInteger8 clockClass;
        Enumeration8 clockAccuracy;
        UInteger16 offsetScaledLogVariance;
};
```

6.4.4 Protocol data unit (PDU) formats

6.4.4.1 General

The data types defined in 6.4.2 and 6.4.3 shall be mapped onto the wire according to the mapping rules for the respective medium, e.g., IEEE Std 802.3-2018 and IEEE Std 802.11-2016, and the terms of 6.4.4.

IEEE 802.1AS PDUs consist of the messages defined or referenced in Clause 10, Clause 11, Clause 12, and Clause 13, based on the data types defined in 6.4.2 and 6.4.3. The internal ordering of the fields of the IEEE 802.1AS PDUs is specified in 6.4.4.3 to 6.4.4.5.

6.4.4.2 Numbering of bits within an octet

Bits are numbered with the most significant bit being 7 and the least significant bit being 0.

NOTE—The numbering and ordering of bits within an octet of a PDU, described here, is independent of and unrelated to the order of transmission of the bits on the underlying physical layer.

6.4.4.3 Primitive data types

Numeric primitive data types defined in 6.4.2 shall be formatted with the most significant octet nearest to the beginning of the PDU followed in order by octets of decreasing significance.

The Boolean data type TRUE shall be formatted as a single bit equal to 1 and FALSE as a single bit equal to 0.

Copyright © 2020 IEEE. All rights reserved.
Authorized licensed use limited to: Linkoping University Library. Downloaded on November 17,2021 at 03:48:56 UTC from IEEE Xplore. Restrictions apply.

Enumerations of whatever length shall be formatted as though the assigned values are unsigned integers of the same length, e.g., Enumeration16 shall be formatted as though the value had type UInteger16.

6.4.4.4 Arrays of primitive types

All arrays shall be formatted with the member having the lowest numerical index nearest to the beginning of the PDU followed by successively higher numbered members, without any padding. In octet arrays, the octet with the lowest numerical index is termed the most significant octet.

When a field containing more than one octet is used to represent a numeric value, the most significant octet shall be nearest to the beginning of the PDU, followed by successively less significant octets.

When a single octet contains multiple fields of primitive data types, the bit positions within the octet of each of the primitive types as defined in the message field specification shall be preserved. For example, the first field of the header of PTP messages is a single octet composed of two fields, one of type Nibble bit 4 through bit 7, and one of type Enumeration4 bit 0 through bit 3 (see 11.4.2 and 10.6.2).

6.4.4.5 Derived data types

Derived data types defined as structs shall be formatted with the first member of the struct nearest to the beginning of the PDU followed by each succeeding member, without any padding. Each member shall be formatted according to its data type.

Derived data types defined as typedefs shall be formatted according to its referenced data type.

Copyright © 2020 IEEE. All rights reserved.
Authorized licensed use limited to: Linkoping University Library. Downloaded on November 17,2021 at 03:48:56 UTC from IEEE Xplore. Restrictions apply.