ANSI E1.17-2015 (R2020), Architecture for Control Networks– Device Management Protocol

Part of ANSI E1.17 – 2015 (R2020) approved by the ANSI Board of Standards Review on 23 March 2020.
This part has no substantive changes from the 2010 edition.

ESTA
630 Ninth Avenue, Suite 609
New York, NY 10036
USA.

Abstract

The Device Management protocol (DMP) is part of the E1.17 protocol suite. This high-level protocol is used to control, monitor, and manage devices in an E1.17 system. DMP is carried by a generalized transport layer protocol that provides reliable and ordered delivery of data.

Contents

Figures

Suggestions for improvement of this standard are welcome. They should be sent to ESTA.

1 Introduction

The Device Management Protocol (DMP) exists to provide configuration, monitoring, and control of equipment within an E1.17 system. In DMP, a controller controls devices across the network by getting and setting the values of properties of those devices. DMP defines the addressing structure necessary to identify individual properties and the messages necessary to manipulate them efficiently. DMP also provides a method allowing components to subscribe to property events and asynchronously to publish the values of their properties when those values change.

DMP messages and their responses are sent over connections whose state of connectedness determines, for example, whether message responses are to be expected and property subscriptions maintained. Connection status is an important feature of DMP allowing controllers and devices to exhibit intelligent behavior as they come and go in network systems.

DMP messages are typically very short. DMP uses the standard E1.17 PDU format (see [Arch]) to facilitate efficient packing of multiple short messages. Multiple short DMP command messages for different components are packed together and sent to an entire group of components together.

Protocols that transport DMP shall provide reliable delivery; however, not all DMP messages need be sent using reliable delivery. See Section 12.

2 Scope, Purpose, and Application

2.1 Scope

This standard defines a protocol to be used within E1.17 systems to control devices. This protocol is intended for use within the E1.17 protocol suite. This protocol may find use outside of the E1.17 protocol suite but this is not a primary design goal.

2.2 Purpose

The purpose of this standard is to specify the device control and management protocol to be used within E1.17 systems.

2.3 Application

This protocol will be used within E1.17 entertainment control systems to control devices.

3 Properties

3.1 General

At the center of DMP control are properties. A property is analogous to a variable in a programming language or a cell in a spreadsheet application and typically represents an aspect of the state of a device, for example, the intensity of a light, or the desired or

actual position of a motor. Properties may have different types (integer, string, etc.) and may be read/write or read only or even write only. A property's value may depend purely on what it is set to via the E1.17 network or may represent some external factor, for example, a temperature sensor, or the position of a slider. A property may also be a constant, for example the serial number of the product.

3.2 Property Types

DMP is not concerned with the actual data type of the properties it manipulates. Rather, it is interested in whether a property is of fixed size or variable size and with how many octets are utilized in the transmission of the data values for properties. Note that some DMP properties may have a fixed size of zero. In DMP, variable size property values are always transmitted in a specific format in which the actual value is preceded by the length of the value in octets. Actually, this length value contains the two octets used to encode the length of the value in addition to the variable number of octets used to encode the value. This is illustrated in the figure below. When a variable size property value has a size of zero, the encoding in the PDU still contains two octets specifying that its size is zero. A variable size property value of zero size will have a length of two (for the two octets that specify length).

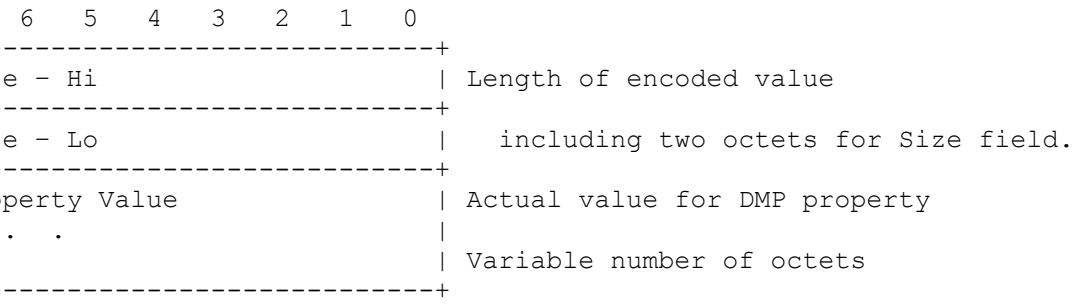

Figure 1: Encoding of Variable Size Data in DMP

4 Connections

Messages in DMP are sent over *connections* that are implemented on top of an underlying transport protocol. It is not possible to send a message in DMP except over a connection. Connections have a single owner and multiple members. Messages may be sent from the owner to multiple members of a connection or from a member to the owner of the connection. Messages may not be sent on a connection from one member to another. Correct implementation of DMP relies on knowing the connection over which messages are sent and on knowing the state of that connection. Each response message, with the exception of *Subscribe Accept*, shall be sent on the same connection that carried the message that generated it. If a connection is lost before its response can be sent, the response message shall not be sent. Subscriptions, which allow changes in property values to be published to interested components, shall be terminated if the connection on which they are published becomes disconnected.

5 Addressing

The definition and format of all PDUs and PDU blocks within DMP, including byte ordering and packing rules shall be as specified in this Standard and in [Arch]. Each data block passed between DMP and its underlying transport protocol shall contain a single PDU block composed of DMP messages.

5.1 Addressing in DMP

Messages in DMP are sent between E1.17 components. A controller sending a message identifies the destination component to the underlying transport layer (e.g. SDT). This underlying transport layer ensures that the message is delivered to the target component with the desired level of reliability.

Once a message is delivered to a desired component, its contents may address specific properties. The particular DMP message determines what is done regarding those properties.

5.1.1 Property addresses

Within a component, properties are addressed in a one dimensional property array. Property addresses shall be in the range 0 to 4294967295 (32 bits). The device manufacturer determines the indices where a component's properties are mapped within the component's property array. There is no requirement for property addresses to be contiguous.

This addressing scheme gives great flexibility to manufacturers to define the layout of properties to allow efficient use of DMP's range and relative addressing (see below). DDL definitions (see [DDL]) specify how properties of devices map into property addresses used within DMP messages.

5.1.2 Relative Addresses

Relative addresses are used to specify an address as an offset from the most recently used address within the same PDU Block. Addresses generated through the use of relative addressing that exceed the maximum property address shall wrap around to the beginning of the address space. Since addresses are always interpreted as 32-bit unsigned integers regardless of their encoding on the wire, the result of a relative address calculation is always modulus 2^32.

Devices shall maintain a record of the last address used (irrespective of the message type). If a command addresses a range of properties then a subsequent relative address shall be relative to the last property in that range.

Since relative addresses specify offsets to properties within the same DMP PDU Block, controllers shall ensure correct processing by using an absolute address for the first property access in any PDU Block.

5.1.3 Addressing Absolute and Relative, Single and Range

5.1.3.1 General

Most DMP commands take a single property address or an address range. These addresses may be absolute (their full value is given in the address field), or relative (an offset from a previous address is given in the address field). These addresses may be in 1, 2, or 4 byte format. Ranged addresses are used to specify multiple properties starting with the address of a first property, an increment between each address in the range, and the number of addresses in the range.

Range addresses in which some or all of the addresses in the range identify non-existent properties shall not be used.

5.1.3.2 Addressing Notation

Within this Standard and elsewhere that addresses are identified in text, DMP property addresses are written using a specific notation. Single addresses are given as a single number specifying the address. Range addresses are written as three numbers specifying the first property address, the increment between each address in the range, and the number of addresses in the range respectively. In both cases a '+' preceding the address indicates that it is a relative address - otherwise it is an absolute address.

Here are some examples: '275' is a single absolute address. '100, 2, 3' addresses properties 100, 102 and 104. Order is important in the following relative address examples: '100' (absolute address establishes start location), '+12' is a single relative address identifying property 112, '+10, 2, 3' is a relative range addresses identifying properties 122, 124 and 126, and '+10' is a single relative address identifying property 136 (not 132).

5.1.4 Address and Data Types

Addresses may be specified in several formats. The address and data format used in each PDU is specified in its header.

The header also contains the size of the address elements (address, increment, or count), encoded in the lowest two bits. The address size is used together with bits in the higher four flags to decode the packing of the address mode and data format as shown below.

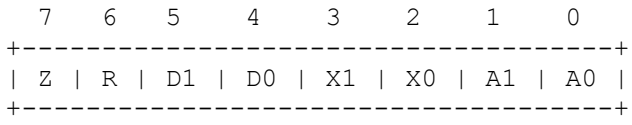

Figure 2: Address and Data Type Encoding

Fields:

Z = Reserved for future use. Shall be 0.

R = Specifies whether address is relative to last valid address in PDU Block or not.
1 = Relative address
0 = Absolute address

D1, D0 = Specify non-range or range address, single data, equal size or mixed size data array
For messages that contain both addresses and data items:
00 = Non-range address, Single data item
01 = Range address, Single data item
10 = Range address, Array of equal size data items
11 = Range address, Series of mixed size data items
For message types that have no data items, e.g., *Get Property,*
00 = Non-range address
01 = Range address
10 = Not permitted
11 = Not permitted
For message types that have no address and no data:
00 = No address and no data
01 = Not permitted
10 = Not permitted
11 = Not permitted

X1, X0 = These bits are reserved and their values shall be set to 0 when encoded. Their values shall be ignored when decoding.

A1, A0 = Size of Address elements
00 = One octet address, (range: one octet address, increment, and count).
01 = Two octet address, (range: two octet address, increment, and count).
10 = Four octet address, (range: four octet address, increment, and count).
11 = Reserved.
See Section 5.1.5, Address Formats, below for the format of addresses.

Single data item = Property data is provided as a single data value.

Array of equal size data items = Property data is provided as a number of equal size data values packed together. The number of data items is equal to the number of addresses specified in the range address. If this format is used in repeating address-data pairs the size of data items must be known by other means.

Series of mixed size data items = Property data is provided as a number of mixed sized data values packed together. There is no guarantee that the data values are the same size. The number of data items in each data series is equal to the number of addresses specified in its paired address. The data sizes of the items in this data field may not be calculated and must be known by other means.

When an array or series of data values is provided, each value corresponds to a property addressed in the message. When a single data value is provided, it corresponds to all properties addressed in the message.

Note that it is possible to have properties with zero sized data. In this case the address and data formats shall indicate that there are data items even though their sizes are zero.

5.1.5 Address Specifier Formats

The format of DMP address fields are shown below. Each address field contains an address specifier that may specify a single property address or a range of property addresses.

5.1.5.1 One Byte Addresses (1) or Offsets (+1)

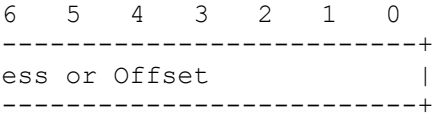

Figure 3: 1 Byte Address or Address Offset Format

Field Definitions:

Address or Offset = An unsigned value between 0 and 255 specifying a single property address or an offset from a property address.

5.1.5.2 Two Byte Addresses (2) or Offsets (+2)

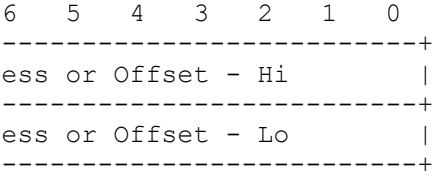

Figure 4: 2 Byte Address or Address Offset Format

Field Definitions:

Address or Offset = An unsigned value between 0 and 65535 specifying a single property address or an offset from a property address.

5.1.5.3 Four Byte Addresses (4) or Offsets (+4)

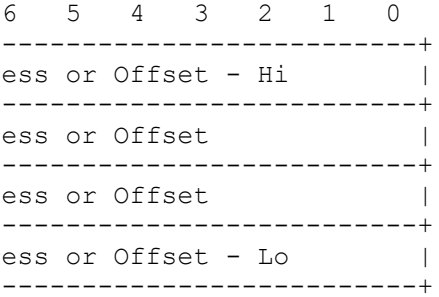

Figure 5: 4 Byte Address or Address Offset Format

Field Definitions:

Address or Offset = An unsigned value between 0 and 4294967295 specifying a single property address or an offset from a property address.

5.1.5.4 One Byte Range (R1/R+1) Addresses

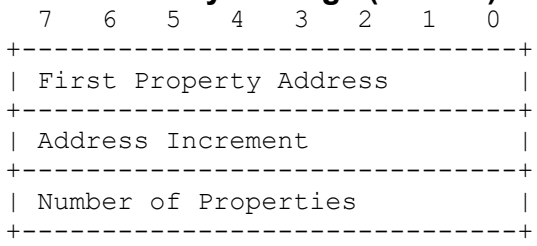

Figure 6: Range 1 Byte Address Format

All properties addressed in a range are of the same size.

Field Definitions:

First Property Address = An unsigned value between 0 and 255 specifying the first property address in a range of addresses.

Address Increment = An unsigned value between 0 and 255 specifying the amount to add to the previous property address in the range to obtain the next property address in the range.

Number of Properties = An unsigned value between 0 and 255 specifying the number of properties in the address range.

5.1.5.5 Two Byte Range (R2/R+2) Addresses

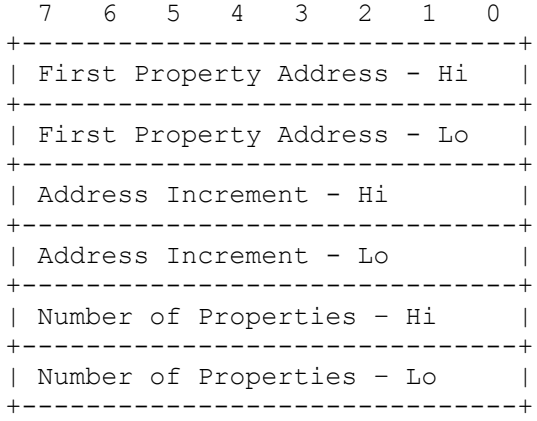

Figure 7: Range 2 Byte Address Format

Field Definitions:

First Property Address = An unsigned value between 0 and 65535 specifying the first property address in a range of addresses.

Address Increment = An unsigned value between 0 and 65535 specifying the amount to add to the previous property address in the range to obtain the next property address in the range.

Number of Properties = An unsigned value between 0 and 65535 specifying the number of properties in the address range.

5.1.5.6 Four Byte Range (R4/R+4) Addresses

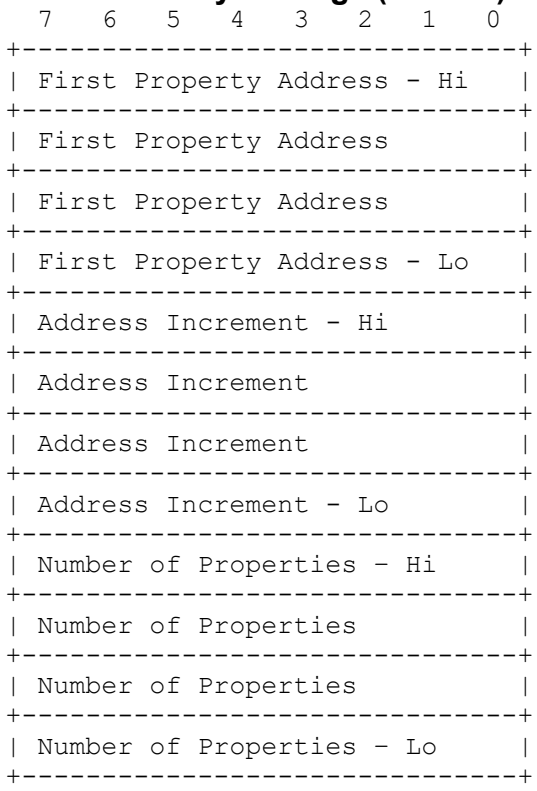

Figure 8: Range 4 Byte Address Format

Field Definitions:

First Property Address = An unsigned value between 0 and 4294967295 specifying the first property address in a range of addresses.

Address Increment = An unsigned value between 0 and 4294967295 specifying the amount to add to the previous property address in the range to obtain the next property address in the range.

Number of Properties = An unsigned value between 0 and 4294967295 specifying the number of properties in the address range.

5.1.6 Optimization of Addressing

The use of a range address followed by an array or series of values with a *Set Property* message allows large numbers of evenly spaced properties in a device to be updated in a very efficient way with very low addressing overhead. The use of relative addressing also allows a series of values to be updated with reduced address and message header overhead.

Enabling this efficiency is the single most important criterion in choosing the address map for the device properties within a component. Properties should be placed so that those properties that frequently need to be updated together can be accessed in a

single range. This is done by placing the properties right next to one another in the address space, or by placing them at fixed address intervals from one another.

Consider a strip light with separate properties for red, green, and blue for each of 5 lights. Assume that the properties are laid out as follows, starting with red for the first light at address 0, green for the first light at address 1, and blue for the first light at address 2, then red for the second light at address 3, and so forth. To update red, green, and blue properties for the first light use the following address range: 0, 1, 3. That is, start with property 0, update 3 properties, each one index apart. Now, to update all green properties, use the following address range: 1, 3, 5. That is, start with property 1 (green for the first light), update 5 properties, each 3 index numbers apart (1, 4, 7, 10, and 13).

It must be emphasized that convenience and efficiency of addressing is the only criterion designers should consider in choosing how equipment properties should be addressed and that the device model used in DDL (See [DDL]) is independent of the DMP property address map.

When a range address is used in conjunction with a *Get Property* message, the component responds with those properties in the same sequence as they would follow for a *Set Property* message. The replying component may break the response up into multiple sub-range replies.

6 Primary Messages

DMP uses 4 primary messages and 7 response messages to do its work.

The primary messages are:
- *Get Property*
- *Set Property*
- *Subscribe*
- *Unsubscribe*

6.1 Getting and Setting Property Values

Get Property and *Set Property* are sent by a controller to monitor and manipulate properties in a remote device.

6.1.1 Get Property

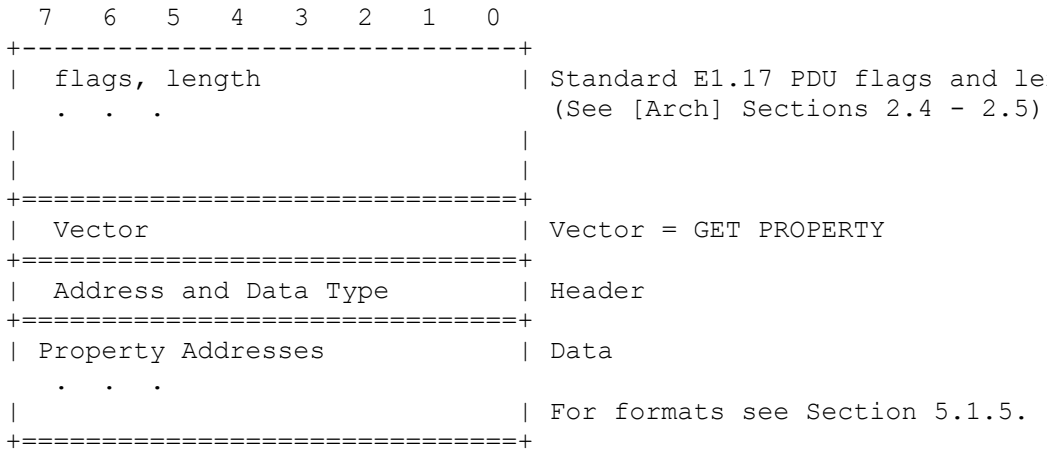

Figure 9: *Get Property* Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, GET PROPERTY.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4. The D1 bit shall be 0.

Data:

Property Addresses = *One or more absolute or relative property* address specifiers in range or non-range, 1 to 4 byte property address format.

Rules:

The D1 bit in the Address and Data Type field shall be 0.

6.1.2 Set Property

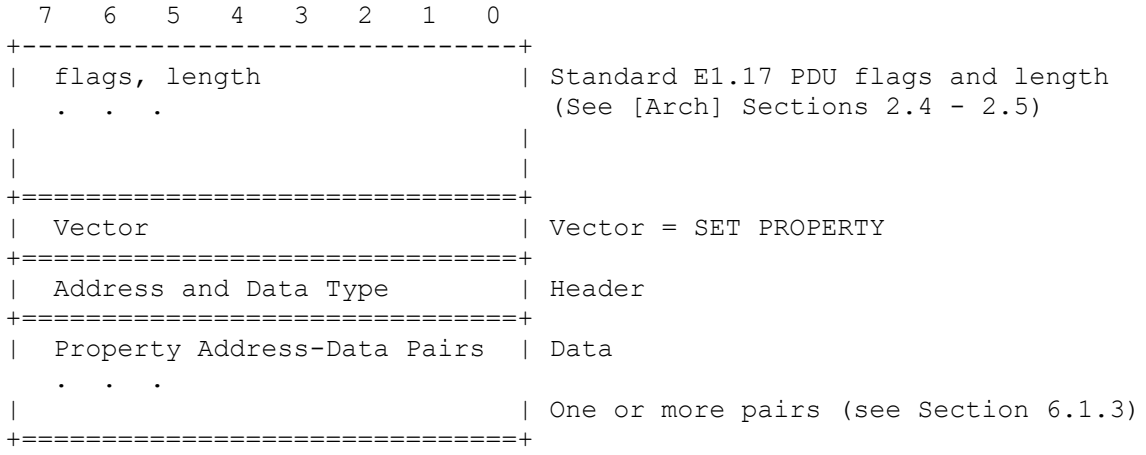

Figure 10: *Set Property* Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SET PROPERTY.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Property Address-Data Pairs = One or more Property Address-Data Pairs as specified in 6.1.3. These pairs shall all have the address type and data format specified in the Address and Data Type field.

6.1.3 Property Address-Data Pair

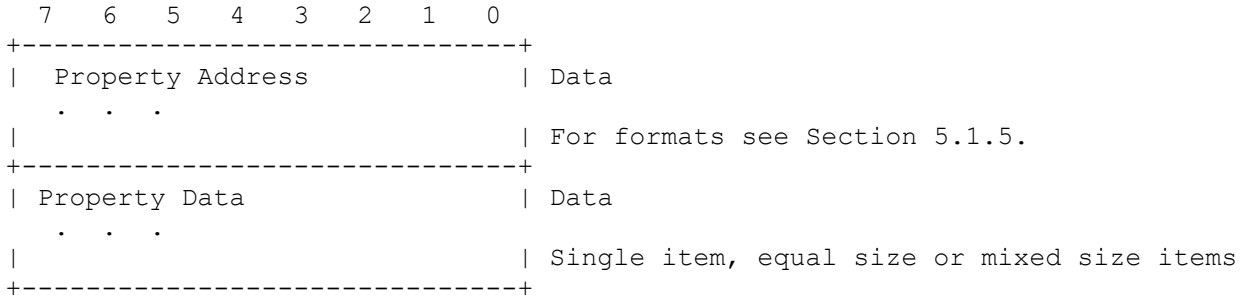

Figure 11 Property Address-Data Pair

Fields:

Property Address = Absolute or relative property address specifier in range or non-range, 1 to 4 byte property address format as specified in the Address and Type field in the message containing this block.

Property Data = As specified by the Address and Type field in the message, either a single data item or a block of either equal size or mixed size data items to be set into the property or properties addressed in the Property Address field.

6.2 Subscriptions

Subscribe and *Unsubscribe* are sent by controllers to indicate their interest in events from a device for the indicated properties. *Event* messages are published (sent) to subscribers.

6.2.1 Subscribe

A controller sends the *Subscribe* message to a device to request that the device begin sending the controller events for the specified properties.

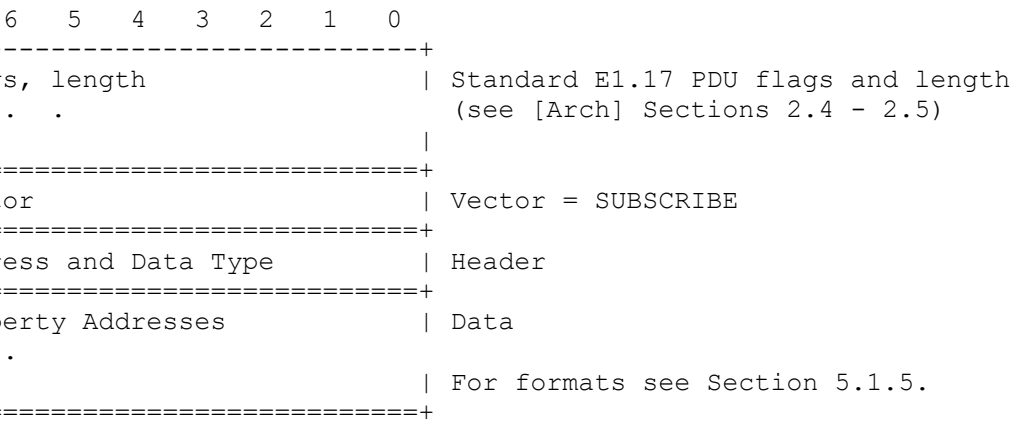

Figure 12: *Subscribe* Property Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SUBSCRIBE.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4. The D1 bit shall be 0.

Data:

Property Addresses = One or more absolute or relative property address specifiers to be subscribed to in range or non-range, 1 to 4 byte address format {1, 2, 4, +1, +2, +4, R1, R2, R4, R+1, R+2, R+4} as specified in Address and Data Type.

Rules:

Subscribe shall be sent reliably.

The D1 bit of the Address and Data Type field shall be 0.

6.2.2 Unsubscribe

A controller sends the *Unsubscribe* message to a device to request that the device cease sending the controller events for the specified properties.

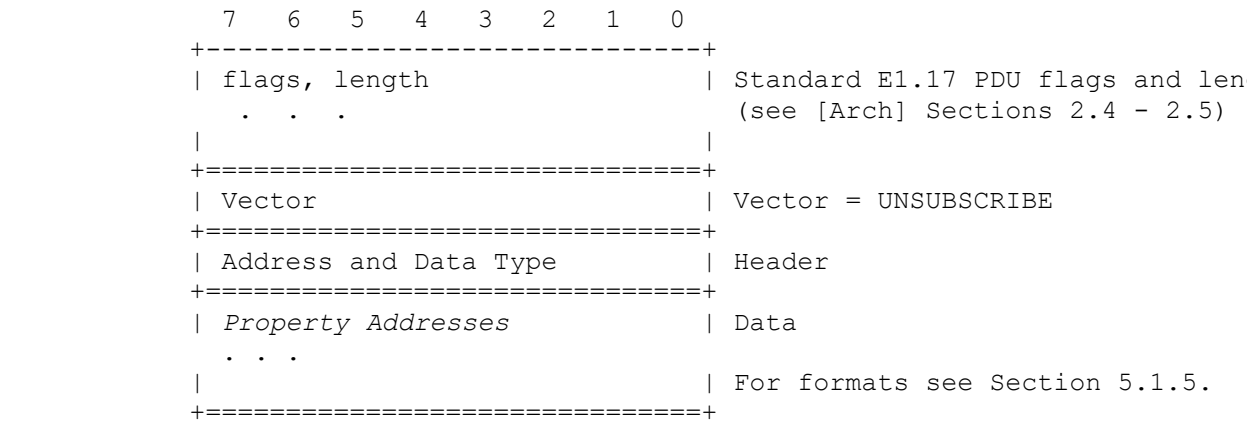

Figure 13: *Unsubscribe* Property Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, UNSUBSCRIBE.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4. The D1 bit shall be 0.

Property Address = One or more absolute or relative property address specifiers in non-range or range, 1 to 4 byte address format {1, 2, 4, +1, +2, +4, R1, R2, R4, R+1, R+2, R+4}, as specified in Address and Data Type field, to unsubscribe from.

Rules:

Unsubscribe shall be sent reliably.

The D1 bit of the Address and Data Type field shall be 0.

7 Response Messages

Messages that are sent in response to DMP primary messages are called Response Messages. Response messages are sent back to the sender of the primary message on the same connection that carried the primary message (except *Subscribe Accept* see Section 7.3.1). Within a connection, DMP devices shall process messages in the order they are received and shall send responses in the order they are processed. A compliant implementation of DMP shall ensure that the order of responses is known by the receiver of those responses, for example, by the use of sequence numbers in the underlying transport.

The response messages are:

- *Get Property Reply*
- *Get Property Fail*
- *Set Property Fail*
- *Subscribe Accept*
- *Subscribe Reject*
- *Event*
- *Sync Event*

7.1 Responses to *Get Property*

Either *Get Property Reply,* or *Get Property Fail* shall be sent in response to *Get Property*. *Get Property Reply* messages contain the value of the property that was requested. *Get Property Fail* indicates that the *Get Property* request could not be met and indicates the reason (e.g. no such property). In the case where the original *Get Property* message addressed multiple properties, multiple responses of different types may be generated.

7.1.1 Get Property Reply

```
  7   6   5   4   3   2   1   0
+-------------------------------+
| flags, length                 | Standard E1.17 PDU flags and length
   .  .  .                      |  (see [Arch] Sections 2.4 - 2.5)
|                               |
+===============================+
| Vector                        | Vector = GET PROPERTY REPLY
+===============================+
| Address and Data Type         | Header
+===============================+
| Property Address-Data Pairs   | Data
   .  .  .
|                               | One or more pairs (see Section 6.1.3)
+===============================+
```

Figure 14: *Get Property Reply* Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, GET PROPERTY REPLY.

Header:

Address and Data Type = An octet specifying the header size, the address type, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Property Address-Data Pairs = One or more Property Address-Data Pairs as specified in 6.1.3. These pairs shall all have the address type and data format specified in the Address and Data Type field.

7.1.2 Get Property Fail

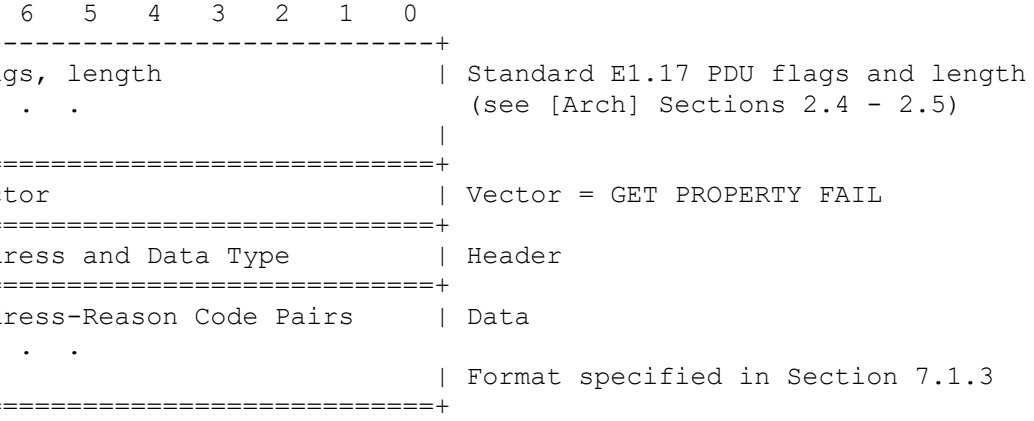

Figure 15: *Get Property Fail* Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, GET PROPERTY FAIL.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Address-Reason Code Pairs Address = One or more pairs of address-reason code pairs. The reason codes shall be one of the reason codes specified in Figure 21, *Get Property Fail* Reason Codes.

Reason	Value	Definition
Nonspecific	1	Non-specific or non-DMP reason.
Not a Property	2	The address does not correspond to a property.
Write Only	3	The property's value may not be read.
Unavailable	13	The property's value is not available due to restrictions imposed by device specific functionality (e.g., access permission mechanisms).

Figure 16: *Get Property Fail* Reason Codes

7.1.3 Property Address-Reason Code Pair

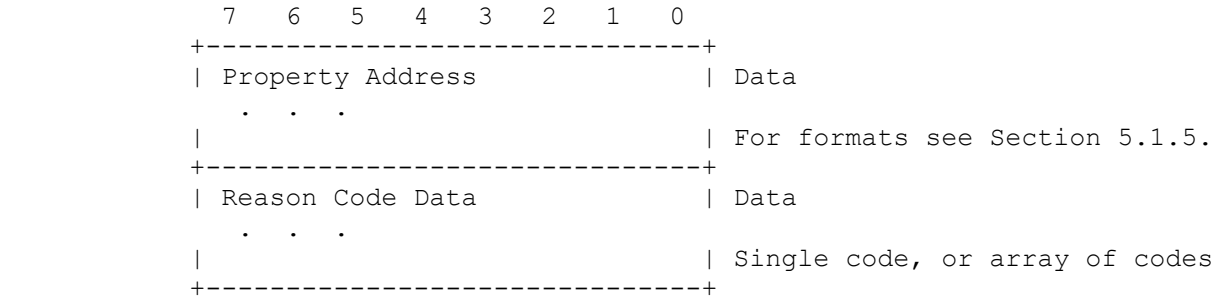

Figure 17 Property Address-Reason Code Pair

Property Address = One absolute or relative property address specifier in non-range or range, 1 to 4 byte property address format.

Reason Code Data = An unsigned single byte integer code indicating the reason for the failure of the corresponding *message* on one or more addresses specified in Property Address. The reason code shall be one of the reason codes valid for the message.

Rules:

The Address and Data Type field of the message containing this pair of fields specifies the type of address and whether there is a single reason code or an array of reason codes for each pair.

7.2 Set Property Fail Response to *Set Property*

Set Property Fail shall be sent in response to a *Set Property* that was not completely successful. The message provides the property address or addresses at which the *Set Property* failed and indicates the failure reason for each property. In the case where the *Set Property* addressed multiple properties, multiple responses may be generated.

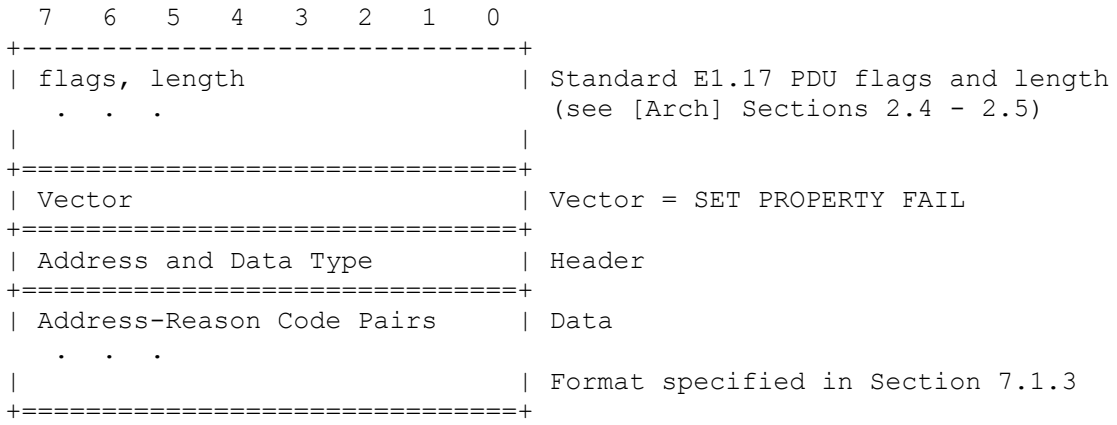

Figure 18: *Set Property Fail* Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SET PROPERTY FAIL.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Address-Reason Code Pairs Address = One or more pairs of address-reason code pairs. The reason codes shall be one of the reason codes specified in Figure 24, *Set Property Fail* Reason Codes.

Reason	Value	Definition
Nonspecific	1	Non-specific or non-DMP reason.
Not a Property	2	The address does not correspond to a property.
Not Writable	4	The property's value may not be written.
Data Error	5	The data does not correspond to the property.
Unavailable	13	The property's value cannot be set due to restrictions imposed by device specific functionality (e.g., access permission mechanisms).

Figure 19: *Set Property Fail* Reason Codes

7.3 Responses to Subscribe and Unsubscribe

Subscribe Accept or *Subscribe Reject* shall be sent in response to a *Subscribe.* No response shall be sent in reply to an *Unsubscribe message.*

7.3.1 Subscribe Accept

A device sends the *Subscribe Accept* message to a controller, in response to a *Subscribe* message, to acknowledge that the device will now begin sending the controller events for the specified properties.

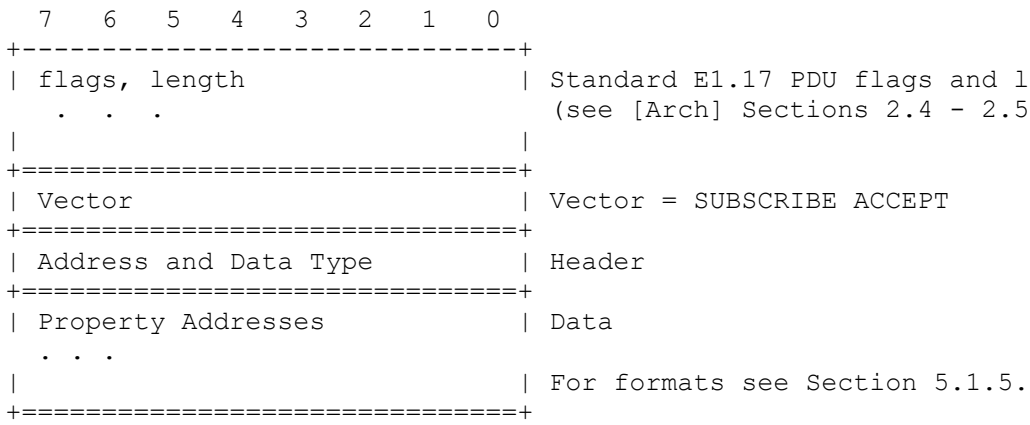

Figure 20: *Subscribe Accept* Property Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SUBSCRIBE ACCEPT.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Property Addresses = One or more absolute or relative property address specifiers in range or non-range, 1 to 4 byte address format as specified in Address and Data Type, to be subscribed to.

Rules:

Subscribe Accept shall be sent reliably.

In order to clearly identify where events will be published, *Subscribe Accept* shall be sent on the connection, owned by the publishing component, to which events for the subscription will subsequently be sent. Note that this is the only response message that is not sent on the same connection that initiated the response.

7.3.2 Subscribe Reject

A device sends the *Subscribe Reject* message to a controller, in response to a *Subscribe Request* message, or because it desires to terminate an active subscription, to indicate that the device will not be sending the controller events for the specified

properties. A reason for the termination of the subscription or rejection of the request is provided in the message.

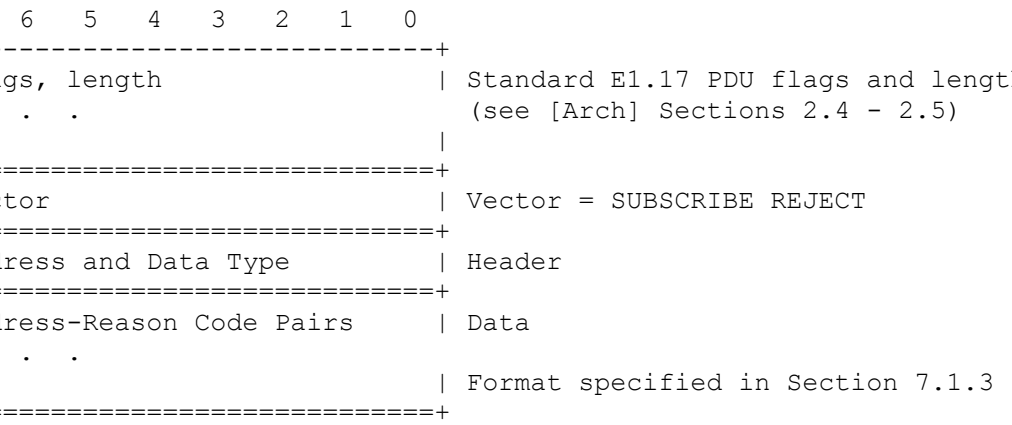

Figure 21: *Subscribe Reject* Property Message Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SUBSCRIBE REJECT.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Address-Reason Code Pairs Address = One or more pairs of address-reason code pairs. The reason codes shall be one of the reason codes specified in Figure 31, *Subscribe Reject* Reason Codes.

Rules:

Subscribe Reject shall be sent reliably.

A device may terminate a subscription by sending this message to the controller on the connection where the events are being sent (published).

A device may reject a subscription request by sending this message on the connection where the *Subscribe* message was received.

Reason	Value	Definition
Nonspecific	1	Non-specific or non-DMP reason.
Not a Property	2	The address does not correspond to a property.
Subscription Not Supported	10	Subscriptions on the specified property are not supported by the device.
No Subscriptions Supported	11	Subscriptions not supported on any property.

```
Insufficient          |  12   | The component cannot support more subscriptions
  Resources           |       |   due to resource limitations
Unavailable           |  13   | The property's value is not available due
                                  to restrictions imposed by device specific
                                  functionality (e.g., access permission
                                  mechanisms).
-------------------------------------------------------------------------
```

Figure 22: *Subscribe Reject* Reason Codes

7.4 Events and Sync Events

A device generates events asynchronously when the value of a property changes or at other times dependent upon the behavior of the property and subject to frequency of event or other limitations that a device may choose to allow for (See Section 11), to let interested controllers track the current value of a property. An event shall not be generated unless the device has been requested by a controller to do so. The generation of events may be controlled by other mechanisms in addition to subscriptions.

A device shall generate a sync event in response to a new subscription to synchronize the subscriber's initial value of the property at the starting point of a stream of events. Sync Events are sent after the *Subscribe Accept* and before any *Events* are sent for that subscription.

7.4.1 Event

The encoding for the *Event* message is the same as that for *Get Property Reply*, except that the vector contains the *Event* message vector value.

```
  7   6   5   4   3   2   1   0
+-------------------------------+
| flags, length                 | Standard E1.17 PDU flags and length
   .   .   .                      (see [Arch] Sections 2.4 - 2.5)
|                               |
+===============================+
| Vector                        | Vector = EVENT
+===============================+
| Address and Data Type         | Header
+===============================+
| Property Address-Data Pairs   | Data
   .   .   .
|                               | One or more pairs. See Section 6.1.3.
+===============================+
```

Figure 23: *Event Message* Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, EVENT.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Property Address-Data Pairs = One or more Property Address-Data Pairs as specified in 6.1.3. These pairs shall all have the address type and data format specified in the Address and Data Type field.

7.4.2 Sync Event

The encoding for the *Sync Event* message is the same as that for *Get Property Reply*, except that the vector contains the *Sync Event* message vector value.

```
  7   6   5   4   3   2   1   0
+-------------------------------+
| flags, length                 | Standard E1.17 PDU flags and length
   .   .   .                      (see [Arch] Sections 2.4 - 2.5)
|                               |
+===============================+
| Vector                        | Vector = SYNC EVENT
+===============================+
| Address and Data Type         | Header
+===============================+
| Property Address-Data Pairs   | Data
   .   .   .
|                               | One or more pairs. See Section 6.1.3.
+===============================+
```

Figure 24: *Sync Event Message* Format

Standard PDU Fields:

Flags = VHDL indicating which of vector, header, data are to be inherited from prior PDU, and indicating whether the extended length format is being used.

Length = The length in octets of the entire PDU including Flags, Length, Vector, Header and Data fields as it occurs in the packet.

Vector = An octet containing the message type, SYNC EVENT.

Header:

Address and Data Type = An octet specifying the address type and size, and the type and count of data in the data section as defined in Section 5.1.4.

Data:

Property Address-Data Pairs = One or more Property Address-Data Pairs as specified in 6.1.3. These pairs shall all have the address type and data format specified in the Address and Data Type field.

8 DMP Components and their Functionalities: Devices and Controllers

All E1.17 messages including DMP messages are sent between components. For more on components see [Arch].

Within a DMP component there can be two distinct functionalities, devices and controllers. A component may contain any combination of these. This means that regardless of which component initiates communication (e.g., regardless of which component connects a session on SDT) either component may issue controller and/or device messages appropriate to their functionality.

Regardless of the functionality outlined here, all DMP components must support all DMP messages, returning the appropriate error results if a message does not make sense for the component functionality. An example of this is a *Set Property* message sent to a component that has no properties. The component shall return an appropriate error result indicating that the property being *set* is not one of its properties.

8.1 Device

All properties contained within a component are part of a device. A device may correspond to a physical piece of equipment but often the equipment is divided into multiple logical devices. For example, a 6-way dimmer pack may have a separate device representing each dimmer and a seventh representing any pack-wide functionality. A DMP component may contain the properties of zero or more devices.

A component containing devices is the recipient of *Get Property* and *Set Property* messages and generates response messages. Although messages are always sent and received by a component, in DMP the message is often loosely referred to as being sent to or from the device. A component containing the properties of devices is said to expose the devices and the properties on the network.

DMP does not recognize devices directly. DMP only addresses the properties within a component. DDL defines how properties within a component are divided among multiple logical devices. (See [DDL].)

Components that contain devices receive these messages:

- *Get Property*
- *Set Property*
- *Subscribe*
- *Unsubscribe*

Components that contain devices generate these messages:

- *Get Property Reply*
- *Get Property Fail*
- *Set Property Fail*

- *Subscribe Accept*
- *Subscribe Reject*
- *Event*
- *Sync Event*

8.2 Controller

A controller issues *Get Property* and *Set Property* messages in order to control the properties of a device. A DMP component may contain zero or one controllers.[1]

Note that while DMP controllers often correspond to equipment that a human operator would recognize as a controller, this is not necessarily so. For example, a DMP controller may simply present status information or may be buried in more complex functionality.

Components that contain controllers generate these messages:
- *Get Property*
- *Set Property*
- *Subscribe*
- *Unsubscribe*

Components that contain controllers receive these messages:
- *Get Property Reply*
- *Get Property Fail*
- *Set Property Fail*
- *Subscribe Accept*
- *Subscribe Reject*
- *Event*
- *Sync Event*

9 Device Classes and DCID

Devices within components are of a type identified by a device class identifier (DCID). This identifier is common to all devices of a specific type and unique to that type. Identical devices will all have the same DCID. Devices having different external representations will have different DCIDs. Each DCID has a single DDL device description associated with it (see [DDL]). DMP messages deal with properties within components. DDL is used to specify and group together properties and formally describe the capabilities in terms of devices. The discovery mechanisms of E1.17 provide the device class identifier and the device description language filename for each component (see [Discovery-IP]).

Device class IDs are 128 bit unique identifiers. The algorithm used to create DCIDs is specified in [UUID]. The advantage of this specific method of generating DCIDs is that

[1] It is meaningless to have more than one controller in a component since DMP provides no mechanism to distinguish between them. However, a single piece of networked equipment may contain multiple components.

manufacturers can freely generate them as required for new types of device without recourse to any higher authority.

10 Device Description

A device description is a textual description of the makeup of a device. The description is written in DDL (See [DDL]). This description contains the descriptions of any properties of the device. Each device class identifier has one device description. If an event (e.g. a software upgrade for defect correction) occurs such that the functional description of the device changes, then a new device class identifier shall be assigned to that device.

Any component that contains multiple devices must also make available, to the E1.17 system, the device descriptions of all its devices when asked. (See [DDL] for how a component may contain multiple devices.) All components containing a specific device will respond with the same device description for that device. This allows controller components to cache the device descriptions associated with device class identifiers.

The device description must always be available from the component containing the device itself (e.g., via TFTP in IP networks) but may also be available from other sources. Other sources may include, for example: CD-ROMs, floppy disks, or the Internet.

11 Subscriptions and Events

11.1 Controller's Request

When a controller needs to be notified automatically as the value of a property within a device is changing or when it issues events for other reasons, it may send a *Subscribe* request to the device. This request also indicates which properties the controller is interested in receiving change events for.

The device may also expose additional properties that control when events are published. It is also the controller's responsibility to set these additional properties. These additional properties will be described in the device's DDL. (See [DDL])

11.2 Device's Response

The device shall either accept or reject the request, as devices are not required to support subscriptions and events. Devices may also reject the request based on resources, and additional property values.

When a device accepts the request, it shall select an existing connection that it owns, or create a new connection to the requester, over which it will publish events for the subscription. The device shall reliably send the *Subscribe Accept* message on that connection indicating that the events shall be sent there. Once the subscription is placed on a connection, the loss of that connection shall terminate the subscription and a new

Subscribe message will be required to re-establish monitoring of the properties of interest.

No more than one subscription per property may exist for a single component. If a *Subscribe* request is received for a property for which there already exists a subscription from that requester, the device shall simply respond with an appropriate *Subscribe Accept*.

After sending *Subscribe Accept* to any new subscriber to a property, a device shall immediately send a *Sync Event* message declaring the value of the property to ensure that the new subscriber starts with the correct initial value. *Sync Event* has the same format as *Event* (and *Get Property Reply*) but a different message type so that other subscribers to the same property can distinguish messages transmitted because of genuine state changes from those due to new subscribers.

11.3 Events

When the value of a property changes within a device or when a device issues an event on a property for other reasons, the device shall notify the subscribed components by sending an event message. *Event* messages shall be sent when property values change either via DMP commands (e.g. *Set Property*) or via other out-of-band means (e.g. front panel switch). The device shall send events for all properties that it has accepted subscriptions on. Additional control of event publishing may also be based on the value of other properties. A device will expose these properties and their meaning using DDL (see [DDL]).

11.4 Event Management

Devices shall manage their event connections. A device shall only request a component to join a specific connection once even if it sends multiple subscription requests for different properties. A device shall ask a component to leave a connection when that component is no longer subscribed to any properties published within the connection and when the device has no other reason to keep the component on the connection. A device may destroy event connections that no longer have subscriptions active.

Should a device need to terminate an active subscription, it shall send a *Subscribe Reject* message to the controller to indicate that is will no longer be sending events related to that subscription.

To conserve bandwidth, individual events shall not be published for a property that has no subscribers. Events may be published that contain a range of properties, some of which are not subscribed to, when this makes sense for system performance reasons.

11.5 Additional Properties

Devices may implement additional properties to control their event publishing mechanism. The values of the properties can be used to control what events are published and when they are published. A device could, for example, expose a property

that indicates the maximum event publishing frequency (i.e., no more than some number events per second no matter how often a property changes). A device could expose a property that indicates the minimum change in a property value before an event is published. The goal of using additional properties is to minimize event session traffic to only what is required for controller functionality. A device may also expose properties that control the reliability with which events are sent.

These additional properties will be described in the device DDL. (See [DDL].)

12 Reliability

Protocols that transport DMP shall provide reliable delivery; however, not all DMP messages need be sent using reliable delivery.

Consider that if a *Set Property* message fails to reach its destination, the receiving device will be in a state different than that intended by the controller. If there is no continuous retransmission (as for example in [DMX512]), this situation could persist. Continuous transmission of unreliable values is a very efficient method of control when the values are changing at or above the desired control frequency or refresh rate of the system. If the rate of change of values declines, it may make sense to send the values using reliable delivery to avoid excessive retransmission of redundant values. A controller is responsible to utilize the level of reliability it requires to achieve its desired control.

Response messages shall be sent at least as reliably as the messages that initiated them. For example, if a *Get Property* message is sent reliably, the response, *Get Property Reply, or Get Property Fail,* shall be sent back reliably.

13 Protocol Constants

13.1 Protocol Codes

Constant Name	*Value*	*Definition*
DMP_PROTOCOL_ID	2	PDU protocol value

Figure 25: Protocol Codes

13.2 Message Codes

Constant Name	*Value*	*Definition*
GET PROPERTY	1	PDU vector value
SET PROPERTY	2	PDU vector value
GET PROPERTY REPLY	3	PDU vector value
EVENT	4	PDU vector value
reserved	5	PDU vector value
reserved	6	PDU vector value
SUBSCRIBE	7	PDU vector value
UNSUBSCRIBE	8	PDU vector value
GET PROPERTY FAIL	9	PDU vector value
SET PROPERTY FAIL	10	PDU vector value
reserved	11	PDU vector value
SUBSCRIBE ACCEPT	12	PDU vector value
SUBSCRIBE REJECT	13	PDU vector value
reserved	14	PDU vector value
reserved	15	PDU vector value
reserved	16	PDU vector value
SYNC EVENT	17	PDU vector value

Figure 26: Message Codes

14 Allowable Data and Address Combinations

Consult the table below for the allowable message code, data and address type combinations.

	Single Address, Single Value		Single Address or Range Address, No Values		Range Address, Single Value		Range Address, Series values	
	Absolute	Relative	Absolute	Relative	Absolute	Relative	Absolute	Relative
Set Property	X	X			X	X	X	X
Set Property Fail	X	X			X	X	X	X
Get Property			X	X				
Get Property Reply	X	X			X	X	X	X
Get Property Fail	X	X			X	X	X	X
Event	X	X			X	X	X	X
Subscribe			X	X				
Unsubscribe			X	X				
Subscribe Accept			X	X				
Subscribe Reject	X	X			X	X	X	X

Figure 27: Allowable Data and Address Types by Message

Annex A: E1.17 Definitions

access protocol	The network or datalink protocol used to access and control the devices described by a Device Description (e.g. DMP for a DMP device). DDL may be adapted to many different access protocols both within and outside of E1.17 and a single device may support multiple access protocols.
ad hoc	Communication between two components that does not occur within a session. Ad hoc messages are passed directly as enclosing layer PDUs and have no reliability or ordering mechanism.
appliance	In DDL an appliance is a piece of equipment described by a root device and all its children and descendents. In DMP systems an appliance corresponds to a component that exposes one or more devices (since the rules require that all devices are descendants of a single root device). See Also root device.
canonical form	Many specifications allows a variety of forms or methods for some items. This is usually useful in the interests of flexibility or ease of processing (for instance by removing the need for extensive checking). However, there may be times when it is required to have exactly one way of representing each case and in this situation a canonical form specifies which of a set of choices must be used such that each possible case has exactly one unambiguous representation. For example, the floating point numbers 12E5 and 1.2E6 are identical. Many specifications dictate that the normalized form with exactly one non-zero digit before the decimal point (1.2E6) be used as the canonical form.
CID	Component Identifier. A UUID [UUID] identifying a particular component.
client protocol	A (higher layer) protocol whose messages are transported by the current one. e.g. DMP is a client protocol of SDT, while SDT is a client protocol of the ACN Root Layer Protocol and the Root Layer Protocol is a client of UDP (or other transport). See also transport protocol.
component	The process, program or application corresponding to a single ACN endpoint. All messages in ACN are sent and received by a component. See [Arch] for a more complete definition.
control session	Any session in DMP's transport protocol (e.g. SDT) in which a DMP controller performs property manipulation on devices (i.e. sends control messages).
controller (DDL)	A machine or program that is used to control devices described by DDL. The controller is connected to the device(s) via one or more data links or networks using one or more access protocols.
controller (DMP)	An ACN component capable of retrieving and setting DMP properties in other components using DMP.
DCID	Device class identifier. This is a unique identifier *UUID* for a Device Class.

	All devices have a DCID that they share with all other devices of their class. The DCID may be used as a key to recognize or retrieve the device description for a device class.
DDL	Device Description Language. See [DDL].
device (DMP)	A device is part of a Component that exposes properties that may be manipulated by a controller using DMP (a component may contain zero or more devices). A device is always of a Device Class and has an associated DCID and device description.
device (DDL)	A device is an entity that may be monitored and controlled by means of a network or datalink and where a DDL description is available describing how to do this. A device includes all those devices (sub-devices) contained within it (device is a recursive term).
device class	The set of devices that are all described by the same device description.
device description	The complete textual definition of a class of devices. Device descriptions are written in E1.17 Device Description Language (DDL). Because of the recursive definition of a device in DDL, a Device Description is also defined recursively and may describe an entire piece of equipment or a part of it.
downstream channel	The channel owned by a session leader and used to communicate to session members.
downstream traffic	Data sent from the session leader to members within the session using the downstream channel.
EPI	ESTA Profile for Interoperability – Synonymous with Interoperability Profile. A brief document that specifies a particular usage or method to be used within ACN systems to ensure interoperability within a particular well defined context. The use of EPIs allows interoperable implementations to be defined for specific environments without restricting the applicability of ACN to other different environments. EPIs can and often do reference other EPIs as part of their specification.
event session	Any session in DMP's transport protocol (e.g. SDT) in which a DMP device sends event messages to report the value of one or more properties.
expose	Features or functionality of a component that may be discovered by the E1.17 discovery mechanisms and are then potentially accessible over the network are said to be exposed by that component. e.g. the term is applied to properties and devices accessible via DMP.
interoperability profile	See EPI.
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6.
message	A valid PDU.
MID	Member Identifier that represents a component within a session. MIDs are unique within a session.

ordering	A mechanism that ensures that all messages within a session are processed by the members in the order intended by the leader.
packet	A PDU block sent as a distinct unit in the underlying protocol (e.g., A PDUBlock received from SDT)
PDU	Protocol Data Unit is a single message (or command). A PDU may contain a PDU block with further embedded PDUs.
PDU block	A contiguous set of PDUs (messages) within a packet or containing PDU.
property (DMP)	A property is a single setting within a device (e.g., intensity) Properties are addressed within a component.
property (DDL)	A property is one aspect or facet describing an entity. e.g. a dimmer might have a property of intensity, while the intensity property could itself have a property of maximum level.
proxy	A proxy is a DMP component that exposes a set of properties that are then mapped to other properties within this or other components. Proxies may be used for many purposes in a DMP system (e.g. they can be used to connect two systems, change the interface to devices, control access to devices, and act as splitters and routers).
reliable message	Any message (PDU) transmitted inside the reliable sequenced wrapper of SDT.
root device	An instance of a device described in DDL that has no parent device. See Also appliance.
root layer PDU	A PDU contained within the PDU block at the packet level.
semantic	A term common in linguistics, computer science, logic and formal languages (e.g. DDL) that identifies the *meaning* of an item or construct as distinct from its syntax (the term *grammar* is loose and is sometimes taken to include semantics but more often excludes it).
session leader	A component that creates, configures, and manages a session. The leader is the only component to send downstream and receive upstream within a session. (With the exception that members may send NAK messages downstream when configured to do so by the leader).
session member	A component that has formally joined a session. Members receive downstream and send upstream (as a special case, members may also send NAK messages downstream).
transport protocol	A (lower layer) protocol the current one uses for transporting its messages (e.g. SDT is the transport protocol for DMP etc.). Also see client protocol.
unreliable message	Any message (PDU) not sent using the SDT reliability mechanism (See reliable message). Unreliable messages sent in a session are still ordered. Reliability shall be provided by other means if it is required for these PDUs (e.g. by timeouts or by separate protocols at other layers).
upstream channel	The channel owned by a session member and used to communicate to the session leader.

wrapper	A PDU containing an embedded PDU block that may contain arbitrary PDUs for other protocols (and/or for SDT itself).

Annex B: DMP References

Normative

[ACN] Entertainment Services and Technology Association [http://tsp.esta.org]. ANSI E1.17 *Entertainment Technology - Architecture for Control Networks - The edition current whenthis Standard becomes approved by the ESTA Board of Directors.*

[Arch] Entertainment Services and Technology Association [http://tsp.esta.org]. *Entertainment Technology – Architecture for Control Networks*. ACN Architecture. - *The edition current when this Standard is approved.*

[DDL] Entertainment Services and Technology Association [http://tsp.esta.org]. *Entertainment Technology - Architecture for Control Networks*. Device Description Language. - *The edition current whenthis Standard becomes approved by the ESTA Board of Directors.*

[Discovery-IP] Entertainment Services and Technology Association [http://tsp.esta.org]. ESTA TSP *Entertainment Technology - Architecture for Control Networks*. EPI 19. Discovery on IP networks. - *The edition current whenthis Standard becomes approved by the ESTA Board of Directors.*

[UUID] Internet Engineering Task Force (IETF) [http://ietf.org/]. RFC 4122 [http://ietf.org/rfc/rfc4122.txt]. P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. July 2005.

Informative

[SDT] Entertainment Services and Technology Association [http://tsp.esta.org]. *Entertainment Technology - Architecture for Control Networks*. Session Data Transport Protocol. - *The edition current whenthis Standard becomes approved by the ESTA Board of Directors.*

[DMX512] Entertainment Services and Technology Association [http://tsp.esta.org]. ANSI E1.11-2008. *USITT DMX512-A - Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories*. 2008.