ACN Protocol Implementation DMP layer

Draft 080628

Kurt Sterckx

June 28, 2008

Abstract

This document gives information about the implementation of the DMP layer of the ACN protocol. This is NOT a official document.

1 Reference

You can't read this document without having access to the following documents.

[ACN] Entertainment Service and Technology Association [http://www.esta.org/tsp/]. ANSI E1.17-2006, Entertainment Technology – Architecture for Control Networks. 2006-10-19

[Arch] Entertainment Service and Technology Association [http://www.esta.org/tsp/]. ESTA TSP CP/2003-1007R4. Entertainment Technology – Architecture for Control Networks. "ACN" Architecture. 2006-10-19

[DMP] Entertainment Service and Technology Association [http://www.esta.org/tsp/]. ESTA TSP CP/2003-1010R3. Entertainment Technology – Architecture for Control Networks. Device Management Protocol. 2006-10-19

[DDL] Entertainment Service and Technology Association [http://www.esta.org/tsp/]. ESTA TSP CP/2003-1011R4. Entertainment Technology – Architecture for Control Networks. Device Description Language. 2006-10-19

.

2 Introduction

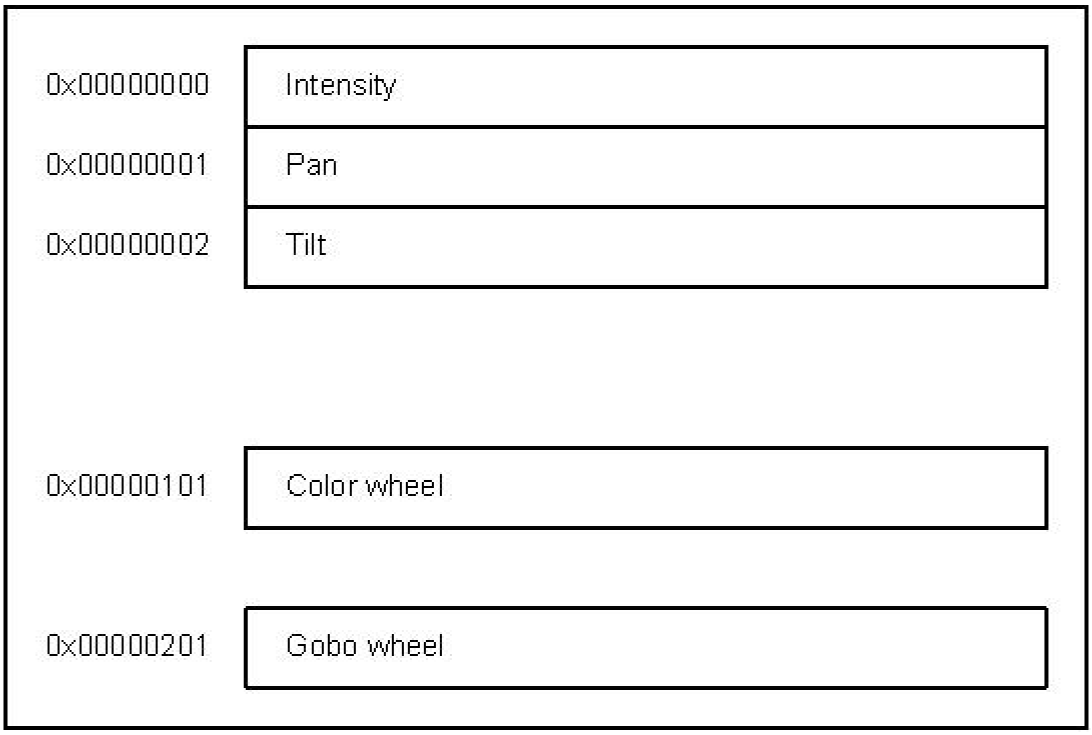
This document gives suggestions and remarks about implementing the DMP layer of the ACN protocol. The reader must be familiar with [ACN], [Arch] and [DMP]. When there is a conflict between this document and the reference documents, the information in the reference documents must be used.

I have personal no connection with the ESTA nor the ACN commission, so all the information in this document is based on reading the [ACN] documentation. It took me a lot of reading to understand the documentation and I hope that this document shortens that process for a reader.

I write this document while I try to implement C++ classes that can be used to create/parse ACN packets. This software will be available as open source (http://ksacn.sourceforge.net/).

3 DMP

DMP is used in the ACN protocol suite to exchange property values. In the ACN world each "device" has a map with properties. Each property has a address in this map.



A property represent a certain value. That value can be as short as one byte and as large as a bitmap. It is very important to known that DMP doesn't carry any information about the value type. This information is in the DDL file. This means that a controller can only know the value type (the number of bytes and the format) of a property by reading the DDL file of the device. A device know the value type because it is the owner of the property, it knows the properties it uses.

The address is a four octet(32-bit) number. Each property has a unique address. In DMP you can set a property by supplying its address and a value.

4 Fixed sized and variable sized values

The value of a property can have a type with a fixed number of bytes. Think of the 8-bit intensity value used for a dimmer. It will be specified as a one octet value in the DDL file(varsize="false"; size="1"). In this case the value can be placed in the DMP data without any extra information. When the DMP packet is parsed by the device it will find that the property at the dimmer address is accessed and it knows that the value is one octet. So it can read one byte from the data. In this case the value length is NOT present in the DMP packet.

The value of a property can also have a variable number of bytes. A example of this is a string that can have a variable number of characters. In this case the DDL file will specify that the property is a variable type. The DDL will also specify the maximum length(varsize="true"; size=maximum size). When the controller wants to set the value of a variable sized property it will first store the real size of the value in the DMP data in two octets, followed by the value octets(see 3.1 Property types in [DMP]). A device that parses the DMP packet will known that the property has a variable sized value. So it will first read the size and then value bytes.

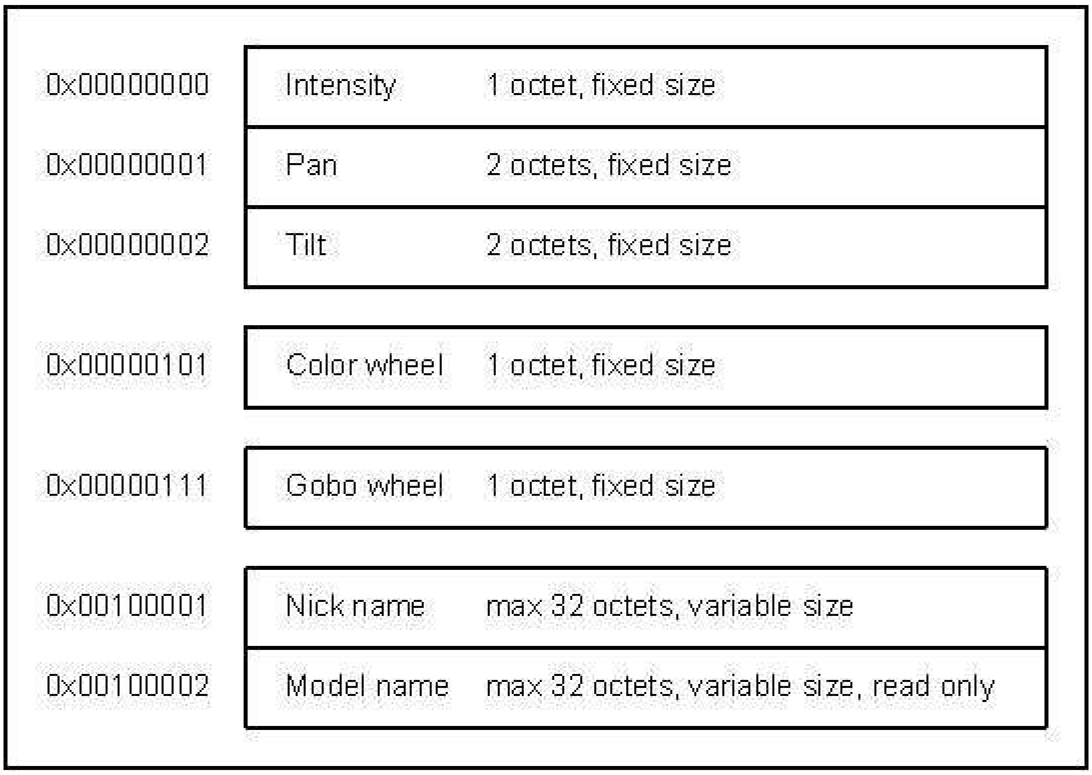
In either case the controller must known the type of a property. It can known this by reading the DDL. The device must also known the type of a property. It is the owner of the properties, so it knows there sizes.

It is **VERY IMPORTANT** to understand that the type/size of the properties is <u>NOT</u> present in the DMP packet. This means that a monitor devices that reads ACN packets, can't parse the DMP data, without reading the DDL file. A example is Wireshark, a Ethernet monitor, it can't parse DMP data. It can however skip the data, so it can parse the other PDU's.

I think that the type information is left out of DMP packet to make them smaller. Because the controller must always have a way to known the types of the properties before it can manipulate them, it also known there size. The device already known the sizes of his properties. So it would only mean more data to transfer and more error checking to perform when the size of the properties would also be present in the DMP data.

5 Example device

For the rest of the text it is better to have a example device, so we can reference to its property map. The device is controlling a very simple moving head. The next pictures shows his property map.

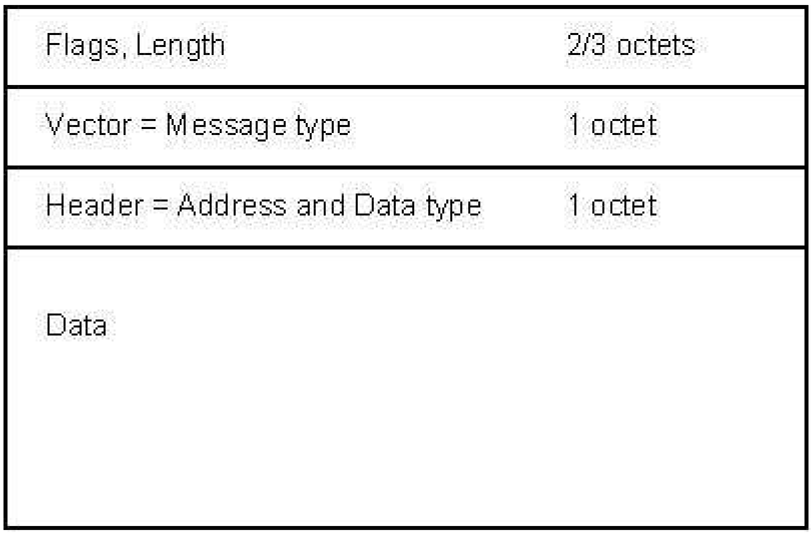


0x00000000	Intensity	1 octet, fixed size
0x00000001	Pan	2 octets, fixed size
0x00000002	Tilt	2 octets, fixed size
0x00000101	Color wheel	1 octet, fixed size
0x00000111	Gobo wheel	1 octet, fixed size
0x00100001	Nick name	max 32 octets, variable size
0x00100002	Model name	max 32 octets, variable size, read only

I think the properties are very well known in the lighting world. Only the Nick name is something new. This is a name that can be given to the head by a user. This can be useful to identify the moving head. The model name is the name given by the manufacture to this moving head. It is a read only property. All the other properties can be read and written.

6 DMP messages

A DMP packet contains messages. Each message is place in a PDU. The vector of the PDU contains the message type. This vector is always one octet long. The header of the PDU contains the **Address and Data type**. This octet represents the format of the addresses(and **NOT** the values) used in the DMP data. The DMP data is placed in the PDU data. The format of the data depends on the message type.



Flags, Length	2/3 octets
Vector = Message type	1 octet
Header = Address and Data type	1 octet
Data	

DMP has in total 16 messages. The messages are place in two categories: Primary messages and Response message. Each category has 8 messages. The primary message are send by a controller. The response messages are the response on a primary message, and are send by the device.

7 Address and Data type

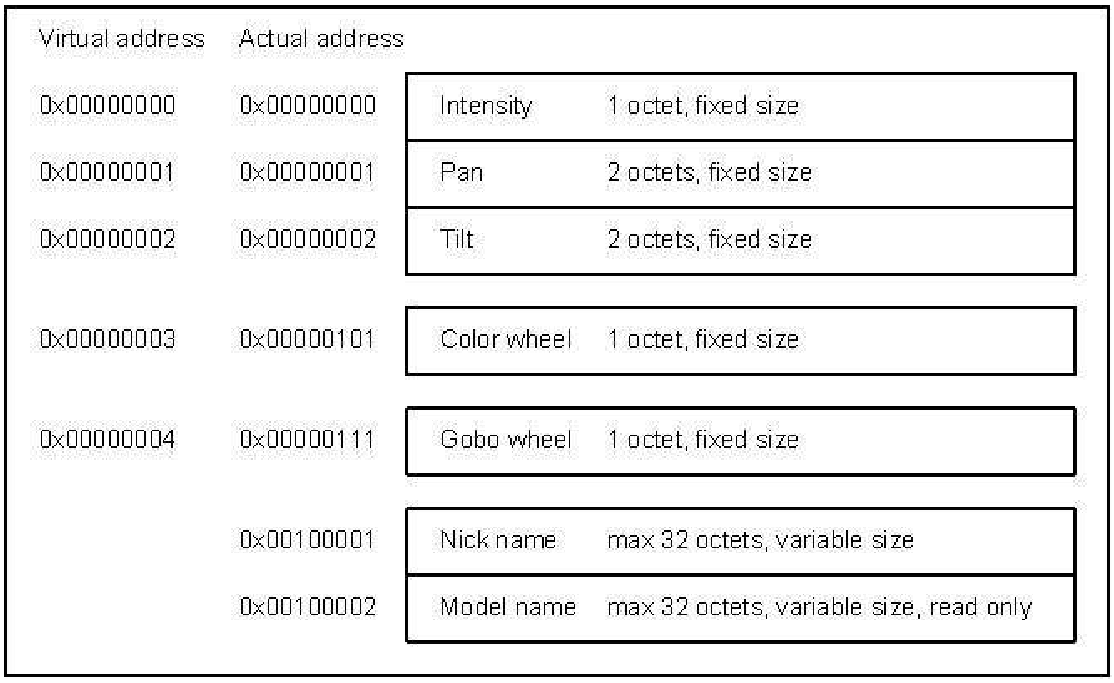
I find the name of this octet a bit confusing. In most cases the octet doesn't define the data type. So the name should better have been Address type. The octet gives information about the addresses used in the DMP data.

The octet has five fields (see 5.1.5 Address and Data types in [DMP]), but only four of them are defined.

7.1 Virtual/actual field

A address can be actual or virtual. The actual address is the address like specified in the DDL file. A controller can however create a virtual address map in the device. This virtual address map is like a patching system. In stead of using DMX address 1, you patch it to channel 5 of your desk, from that moment you can use channel 5 to access DMX address 1. The same thing can be done with the virtual address map. The controller supplies a virtual address that is connected to a actual address. Whenever the device must handle a virtual address, it uses the virtual map to lookup the actual address. This allows a controller to create a more efficient map, allowing to access the properties with shorter DMP messages. It allows also the linking of multiple properties to the same virtual address. This can be used to change multiple properties with just a write to one virtual address (for example when a dimmer has a fade time for each of it channels, the fade properties can be connected to one virtual address. Setting the property by using the virtual address, will change all the fade properties, they will all get the same value).

The next pictures shows a example of a virtual address map for our example device.



Virtual address	Actual address		
0x00000000	0x00000000	Intensity	1 octet, fixed size
0x00000001	0x00000001	Pan	2 octets, fixed size
0x00000002	0x00000002	Tilt	2 octets, fixed size
0x00000003	0x00000101	Color wheel	1 octet, fixed size
0x00000004	0x00000111	Gobo wheel	1 octet, fixed size
	0x00100001	Nick name	max 32 octets, variable size
	0x00100002	Model name	max 32 octets, variable size, read only

The controller must known the actual addresses that are connected to the virtual address. This is needed so it knows the value type of the property.

The device has a virtual map for each controller. The virtual map is never shared amongst controllers. A controller may only have one virtual map. The device can however decide how many maps may be created, meaning how many controllers may create a map. It can take a lot of resources(memory) to implement a virtual map. A device must not support the virtual mapping.

For implementing a virtual map it is very important to known that there can't be more virtual addresses then there are actual addresses. The DDL must also specify which actual addresses can be connected to a virtual address. This can be used to limit the number the resources needed by the device to create a virtual map. A controller may only map a single actual address(single property) to ONE virtual address. When the controller allocates a virtual map, the device will allocate memory for a table that can hold the maximum number of virtual addresses(the same, or less, than the number of actual addresses).

Example implementation:

When the controller allocates a virtual map, the device will create a array. Each item of the array has three items: in use flag, virtual address, actual address. The in use flag indicates of the item is empty or in use.

When the controller uses the Map Property message to map a virtual address to a actual address, the device will perform the following steps:

1. Look up the actual address in the array. When it is already present, [DMP] specified that the actual address must be remapped to the new virtual address.
2. When the actual address is not found, a empty item must be found. This item must be filled with the virtual address and the actual address. The in use flag is also set.
3. I think it is also best to check of the virtual address is connected to actual addresses with the same value type. When this is not the case, it is best to refuse the mapping. There is however no reason code for this. The reason for this, is the fact that one virtual address can be mapped to multiple actual address. When now the virtual address is used to set the value, the value is given to each actual address. This means that the value expected by each actual address must be of the same type. When this is not the case, it is impossible to handle the value. I think it is better to check that the value can be handled, during the mapping, and refuse the mapping when this is not the case. The [DMP] document has no information about this situation.

When the controller send a message with a virtual address, the device will use the array to lookup the virtual address. When the virtual address is found, the actual address is also known and from that the property(and is value type). The virtual address can be present more than once in the array. This means that each actual address connected with the virtual address must be set to the given property value.

The array can be implemented in a more efficient (trees, maps, …), so that the time it takes to lookup a virtual address can be short. It will depend on the number of virtual addresses that are allowed and the resources available to the device, what is the best implementation.

The virtual address map is a complete map that in theory holds all the possible virtual addresses, meaning a 4Giga byte space. However certain virtual address can be unconnected. In that case the virtual address is just ignored.

7.2 Relative/Absolute field

A address can be relative or absolute. In case of absolute address the given address value is the real virtual/actual address value. The value can immediate be used to find the property.

In case of a relative address, the given value must be added to a address stored in the device. This address is called the last address in THIS document.

The last address will be set to the last absolute address that is accessed in the same DMP SDT Client block. When a absolute address is used to access a property, the last address will be set to the absolute address. When a relative address is used to access a property, the absolute address is calculated by adding the relative address to the last address, the result is store in the last address and used to access the property. The addition is done with role over, for example adding 0x00000003 to 0xFFFFFFFF gives 0x00000002.

It is not obvious from [DMP] what must be done when virtual and actual addresses are intermixed in the same DMP SDT Client block and both use relative addressing. It think it is best to have a last virtual address and a last actual address. They are two separated address spaces.

A DMP SDT Client block must always start with a absolute address. This means that the last address is set to the value UNKNOWN when the parser starts reading the DMP SDT Client block. Any relative address that is given when the last address is set to unknown, is a error. Any actual address given will be stored in the last address and remove the UNKNOWN value.

The use of relative addresses allows for very compact DMP packets.

7.3 Non-range/Range

To reduce the size of a DMP packet even further the ACN protocol allows the controller to access a range of addresses. There are four different formats:

7.3.1 Non-Range, single data item

In this case there is only one address given and that address is used to access a data item. It depends on the message type if virtual and/or relative addresses can be used (there is a nice table at the end of the [DMP] that shows the different addressing methods for each message type. This document also has a table but in a little bit different format, see 7.5 Address and Data type naming).

The size of the data item is NOT given in the DMP packet. You only know there is only one item. The size of the data depends on the DMP message, and in case it is a property value, the value type of the property.

The given address can be 1, 2 or 4 octets long (see 7.4 Size of address).

7.3.2 Range, single data item

In this case there is start address, a increment and a count given. The range contains the given count of addresses. This is the number of addresses that must be accessed. The range starts at the given start address(that can be virtual and/or relative). To get the next address, the address must be incremented with the given increment.

Each property in the range will be set to given data. This means that all data items must be of the same type(or at least size) !!

The size of the data item is NOT given in the DMP packet. You only know there is only one item. The size of the data depends on the DMP message, and in case it a property value, the value type of the property.

Example:

Start address	Actual/Absolute 0x00000001
Increment	1
Count	2

Data	0x0123

When this is used as data for the Set Property message type and send to the example device, the pan and tilt will both be set to 0x0123.

The given start address/increment/count can be 1, 2 or 4 octets long (see 7.4 Size of address).

It may not be obvious when reading the standard document, but the start address may be virtual/actual and relative/absolute. It depends however on the message type what address types are supported. In case of virtual addresses the address calculation(increment) is performed on the virtual addresses. When there is a non existing virtual address in the range, it is ignored, it is not a error.

7.3.3 Range, array of equal size data items

This uses the same principles as Range, Single Data item.

In this case however there are multiple data items. Each data item must be of the same size. There must be the same number of data items as addresses present in the range. This means that there must be count data items.

The first data item will be given to the start address. The next data item to the start address + increment, and so.

The size of the data item is NOT given in the DMP packet. You only know there are multiple items of the same size. The size of the data depends on the DMP message, and in case it a property value, the value type of the property.

Example:

Start address	Actual/Absolute 0x00000001
Increment	1
Count	2
Data	0x0123
Data	0x5678

When this is used as data for the Set Property message type and send to the example device, the pan will be set to 0x0123 and the tilt to 0x5678

The given start address/increment/count can be 1, 2 or 4 octets long (see 7.4 Size of address).

In case of virtual addresses, the size of the data items can be calculated based on the DMP packet. This is done by dividing the size of the data array by count. There may also be only one address, data pair(see later) in the DMP PDU. This is needed to known the size of the data array.

Size of data array = PDU Length –2/3(Length) -1(Vector) –1(Header) –address length

The address length depends on the number of octets used to represent the start address, increment and count, resulting in a address length of 3, 6 or 12.

When this range type is used to transfer variable sized properties using virtual addresses, all values must have the same size, so the two octets that indicate the length, must be the same (for example when strings are transferred).

7.3.4 Range, array of mixed size data items

This uses the same principles as Range, Single Data item.

In this case however there are multiple data items. Each data item can be of a different size. There must be the same number of data items as addresses present in the range. This means that there must be count data items.

This range can’t be used with virtual addresses.

The first data item will be given to the start address. The next data item to the start address + increment, and so.

The size of the data item is NOT given in the DMP packet. You only know there are multiple items of the same size. The size of the data depends on the DMP message, and in case it a property value, the value type of the property.

Example:

Start address	Actual/Absolute 0x00000000
Increment	1

Count	3
Data	0x81
Data	0x0123
Data	0x5678

When this is used as data for the Set Property message type and send to the example device, the intensity will be set to 0x81, the pan will be set to 0x0123 and the tilt to 0x5678

The given start address/increment/count can be 1, 2 or 4 octets long (see 7.4 Size of address).

7.4 Size of address

The size of the address/increment/count can be 1, 2 or 4 octets. The size of the fields is however the same. So there are only three combinations possible in case of a range.

	Start address Total for non range	Increment	Count	Total length in case of range
1 octet	1 octet	1 octet	1 octet	3 octets
2 octets	2 octets	2 octets	2 octets	6 octets
4 octets	4 octets	4 octets	4 octets	12 octets

7.5 Address and Data type naming

I have named each possibility so I can use the same name in the software. The name has three parts separated by a underscore.

The first part indicates the Virtual/Actual and Absolute/Relative fields:

VA	Virtual + Absolute
VR	Virtual + Relative
AA	Actual + Absolute
AR	Actual + Relative

The second part indicates the range type:

NR	Non range, single data item
RS	Range, single data item
RAeS	Range, multiple data items, equal size (don't ask where

	the A comes from)
RSmS	Range, multiple data items, mixed size

The third part indicates the number of octets used by the address, increment and count fields:

ONE	One octet
TWO	Two octets
FOUR	Four octets

The next table gives all the possible names and the address type

Name	Virtual	Relative	Address size	Address type
VA_NR_ONE	X		1	Non Range, single data item
VR_NR_ONE	X	X	1	
AA_NR_ONE			1	
AR_NR_ONE		X	1	
VA_NR_TWO	X		2	Non Range, single data item
VR_NR_TWO	X	X	2	
AA_NR_TWO			2	
AR_NR_TWO		X	2	
VA_NR_FOUR	X		4	Non Range, single data item
VR_NR_FOUR	X	X	4	
AA_NR_FOUR			4	
AR_NR_FOUR		X	4	
VA_RS_ONE	X		1	Range, single data item
VR_RS_ONE	X	X	1	
AA_RS_ONE			1	
AR_RS_ONE		X	1	
VA_RS_TWO	X		2	Range, single data item
VR_RS_TWO	X	X	2	
AA_RS_TWO			2	
AR_RS_TWO		X	2	
VA_RS_FOUR	X		4	Range, single data item
VR_RS_FOUR	X	X	4	
AA_RS_FOUR			4	
AR_RS_FOUR		X	4	
VA_RAeS_ONE	X		1	Range, multiple data item, same size
VR_RAeS_ONE	X	X	1	
AA_RAeS_ONE			1	
AR_RAeS_ONE		X	1	
VA_RAeS_TWO	X		2	Range, multiple data item, same size
VR_RAeS_TWO	X	X	2	
AA_RAeS_TWO			2	
AR_RAeS_TWO		X	2	
VA_RAeS_FOUR	X		4	Range, multiple data item, same size
VR_RAeS_FOUR	X	X	4	

AA_RAeS_FOUR		4	
AR_RAeS_FOUR	X	4	
AA_RSmS_ONE		1	Range, multiple data item, mixed size
AR_RSmS_ONE	X	1	
AA_RSmS_TWO		2	Range, multiple data item, mixed size
AR_RSmS_TWO	X	2	
AA_RSmS_FOUR		4	Range, multiple data item, mixed size
AR_RSmS_FOUR	X	4	

The next table shows the different DMP messages with the possible address types. I use the names defined in previous table. The size is however omitted (_xxx can be _ONE, _TWO, _FOUR).

Message type	Data item	Address type
Set property	Property Address-Data pairs	VA_NR_xxx (only one pair)
		VR_NR_xxx (only one pair)
		AA_NR_xxx
		AR_NR_xxx
		VA_RS_xxx (only one pair)
		VR_RS_xxx (only one pair)
		AA_RS_xxx
		AR_RS_xxx
		VA_RAeS_xxx (only one pair)
		VR_RAeS_xxx (only one pair)
		AA_RAeS_xxx
		AR_RAeS_xxx
		AA_RSmS_xxx
		AR_RSmS_xxx
Set property fail	Address-Reason code pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx
		AA_RSmS_xxx
		AR_RSmS_xxx
Get property	Property addresses	VA_NR_xxx
		VR_NR_xxx
		AA_NR_xxx
		AR_NR_xxx
		VA_RS_xxx
		VR_RS_xxx
		AA_RS_xxx
		AR_RS_xxx
Get property reply	Property Address-Data pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx

		AA_RSmS_xxx
		AR_RSmS_xxx
Get property fail	Address-Reason code pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx
		AA_RSmS_xxx
		AR_RSmS_xxx
Event	Property Address-Data pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx
		AA_RSmS_xxx
		AR_RSmS_xxx
Map property	Virtual address type + Actual-Virtual address pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx
Unmap property	Actual property addresses	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
Map property fail	Address-Reason code pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx
		AA_RSmS_xxx
		AR_RSmS_xxx
Subscribe	Actual property addresses	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
Unsubscribe	Actual property addresses	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
Subscribe Accept	Property addreses	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx

Subscribe Reject	Address-Reason code pairs	AA_NR_xxx
		AR_NR_xxx
		AA_RS_xxx
		AR_RS_xxx
		AA_RAeS_xxx
		AR_RAeS_xxx
		AA_RSmS_xxx
		AR_RSmS_xxx
Allocate Map	No data	
Deallocate Map	No data	
Allocate Map Reply	One reason code	

The next figure shows the Property address – Data pairs layout for each name. The property value can be removed in case there is no value, or changed to a reason code or virtual address when that is used in the message.

Non-range, single data item
One octet address

VA_NR_ONE
VR_NR_ONE
AA_NR_ONE
AR_NR_ONE

1 Vir 0 Act	1 Rel 0 Abs	0	0	0	0	0	0

Address/Offset
One property value

Non-range, single data item
Two octet address

VA_NR_TWO
VR_NR_TWO
AA_NR_TWO
AR_NR_TWO

1 Vir 0 Act	1 Rel 0 Abs	0	0	0	0	0	1

Address/Offset, High byte
Address/Offset, Low byte
One property value

Non-range, single data item
Four octet address

VA_NR_FOUR
VR_NR_FOUR
AA_NR_FOUR
AR_NR_FOUR

1 Vir 0 Act	1 Rel 0 Abs	0	0	0	0	1	0

Address/Offset, High byte
Address/Offset
Address/Offset
Address/Offset, Low byte
One property value

Range, single data item
One octet address

VA_RS_ONE
VR_RS_ONE
AA_RS_ONE
AR_RS_ONE

1 Vir 0 Act	1 Rel 0 Abs	0	1	0	0	0	0

First property address
Address increment
Number of properties
One property value

Range, single data item
Two octet address

VA_RS_TWO
VR_RS_TWO
AA_RS_TWO
AR_RS_TWO

1 Vir 0 Act	1 Rel 0 Abs	0	1	0	0	0	1

First property address, high byte
First property address, low byte
Address increment, high byte
Address increment, low byte
Number of properties, high byte
Number of properties, low byte
One property value

Range, single data item
Four octet address

VA_RS_FOUR
VR_RS_FOUR
AA_RS_FOUR
AR_RS_FOUR

1 Vir 0 Act	1 Rel 0 Abs	0	1	0	0	1	0

First property address, high byte
First property address
First property address
First property address, low byte
Address increment, high byte
Address increment
Address increment
Address increment, low byte
Number of properties, high byte
Number of properties
Number of properties
Number of properties, low byte
One property value

Range, multipe data item, same size
One octet address

VA_RAeS_ONE
VR_RAeS_ONE
AA_RAeS_ONE
AR_RAeS_ONE

1 Vir 0 Act	1 Rel 0 Abs	1	0	0	0	0	0

First property address
Address increment
Number of properties
Multiple property values, equal size

Range, multipe data item, same size
Two octet address

VA_RAeS_TWO
VR_RAeS_TWO
AA_RAeS_TWO
AR_RAeS_TWO

1 Vir 0 Act	1 Rel 0 Abs	1	0	0	0	0	1

First property address, high byte
First property address, low byte
Address increment, high byte
Address increment, low byte
Number of properties, high byte
Number of properties, low byte
Multiple property values, equal size

Range, multipe data item, same size
Four octet address

VA_RAeS_FOUR
VR_RAeS_FOUR
AA_RAeS_FOUR
AR_RAeS_FOUR

1 Vir 0 Act	1 Rel 0 Abs	1	0	0	0	1	0

First property address, high byte
First property address
First property address
First property address, low byte
Address increment, high byte
Address increment
Address increment
Address increment, low byte
Number of properties, high byte
Number of properties
Number of properties
Number of properties, low byte
Multiple property values, equal size

Range, multipe data item, mixed size
One octet address

AA_RSmS_ONE
AR_RSmS_ONE

0 Act	1 Rel 0 Abs	1	1	0	0	0	0

First property address
Address increment
Number of properties
Multiple property values, mixed size

Range, multipe data item, mixed size
Two octet address

AA_RSmS_TWO
AR_RSmS_TWO

0 Act	1 Rel 0 Abs	1	1	0	0	0	1

First property address, high byte
First property address, low byte
Address increment, high byte
Address increment, low byte
Number of properties, high byte
Number of properties, low byte
Multiple property values, mixed size

Range, multipe data item, mixed size
Four octet address

AA_RSmS_FOUR
AR_RSmS_FOUR

0 Act	1 Rel 0 Abs	1	1	0	0	1	0

First property address, high byte
First property address
First property address
First property address, low byte
Address increment, high byte
Address increment
Address increment
Address increment, low byte
Number of properties, high byte
Number of properties
Number of properties
Number of properties, low byte
Multiple property values, mixed size

7.6 Property Address-Data Pair

In the Set Property, Get Property Reply and Event messages the PDU data can contain <u>multiple</u> address/data pairs. Each pair has the format given by the Address and Data type in the header of the PDU. This means that we can set multiple properties on total different addresses and with total different sizes using only one PDU. We can also set multiple ranges using one PDU. This features is very useful to create small DMP packets.

Example:

The following DMP PDU shows how one PDU can be used to set both the dimmer, pan and tilt property of the example moving head (the Address and data type is AA_NR_ONE=0x00).

Flags, Length	2 octets	0x700C
Vector = Message type	1 octet	0x02
Header = Address and Data type	1 octet	0x00
Address	1 octet	0x00
Value	1 octet	0x81
Address	1 octet	0x01
Value	2 octets	0x0123
Address	1 octet	0x02
Value	2 octets	0x5678

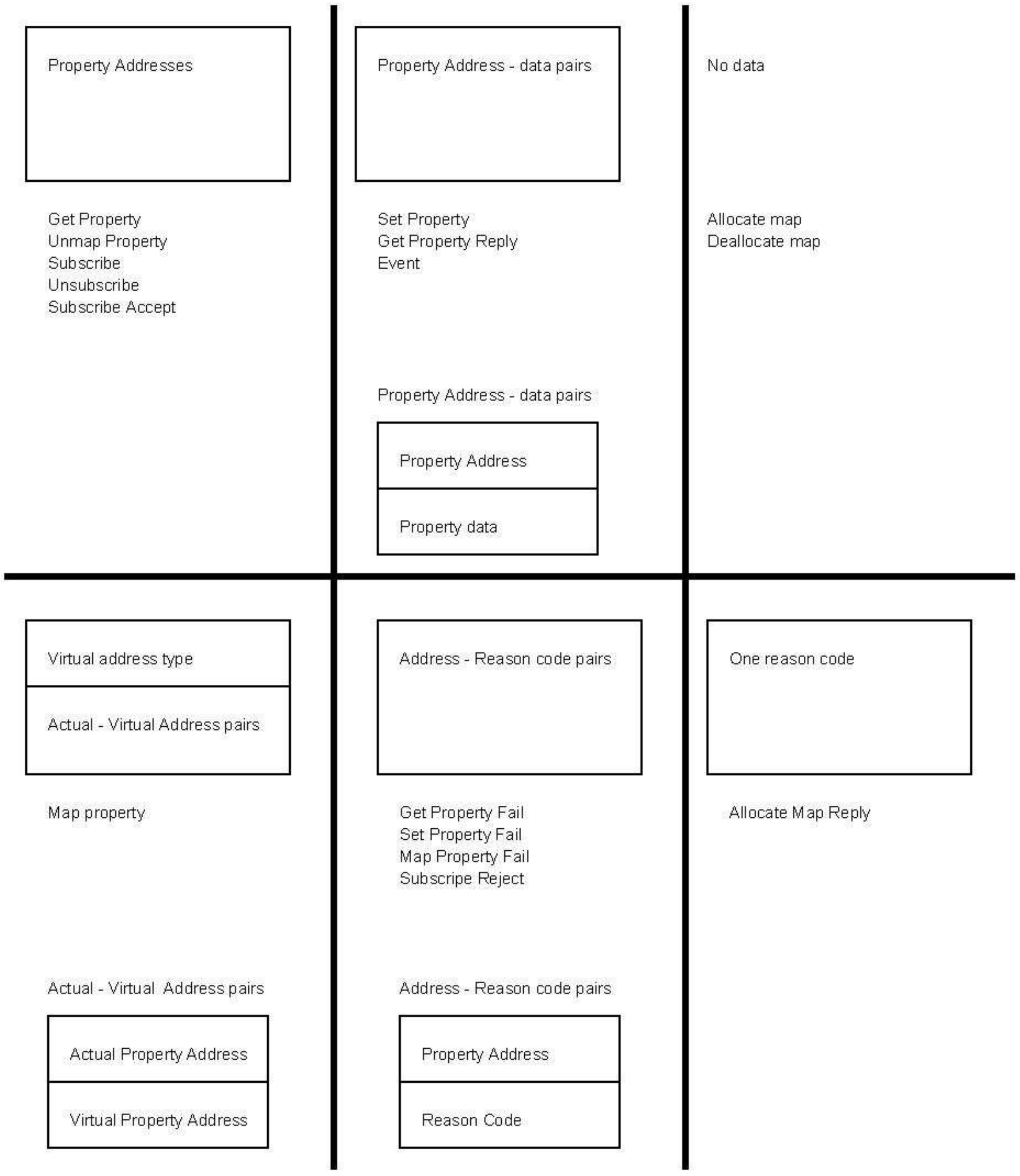
It is important to notice that all addresses have the same size, but the values have different sizes. There is however NO information in the PDU about the value sizes. This means the PDU can only be parsed by software that knows the value sizes (I known, I keep repeating this).

This PDU can't be used to change the other properties by adding a extra address-data pair. The other properties have a address that is bigger then 255. The used address type is actual, absolute, one octet, (AA_NR_ONE) so only addresses from 0 to 255 can be given.

The software that must parse this PDU must use the Length of the PDU to known of there are extra pairs in the data. Each time a pair is found, the remaining length is decrement with the length of the pair. The parsing continues until the remaining length is 0.

7.7 Other data formats

The PDU Data format depends on the DMP message type. There are six different formats possible. The next figure shows the different formats.



7.7.1 Actual – Virtual Address pairs

The data used in a Map property message is rather difficult to understand. The first octet contains a Virtual Address type. This is of the same format as Address and Data type. There are however two things that MUST be the case:

- V bit must be 1, meaning a virtual address
- D1 bit must 0

This means that only the following formats are accepted (with xxx = ONE, TWO or FOUR):

- VA_NR_xxx
- VR_NR_xxx
- VA_RS_xxx
- VR_RS_xxx

The other data in the Map property message contains Actual Address/Virtual Address pairs. The format of these pairs depend on the Actual Address Type placed in the DMP PDU header and the Virtual address type, place in the first octet. The following combinations are possible

Actual Address Type	Virtual Address type
Ax_NR_xxx	Vx_NR_xxx
	Vx_RS_xxx
Ax_RS_xxx	Vx_NR_xxx
	Vx_RS_xxx
Ax_RAeS_xxx	Vx_NR_xxx
	Vx_RS_xxx

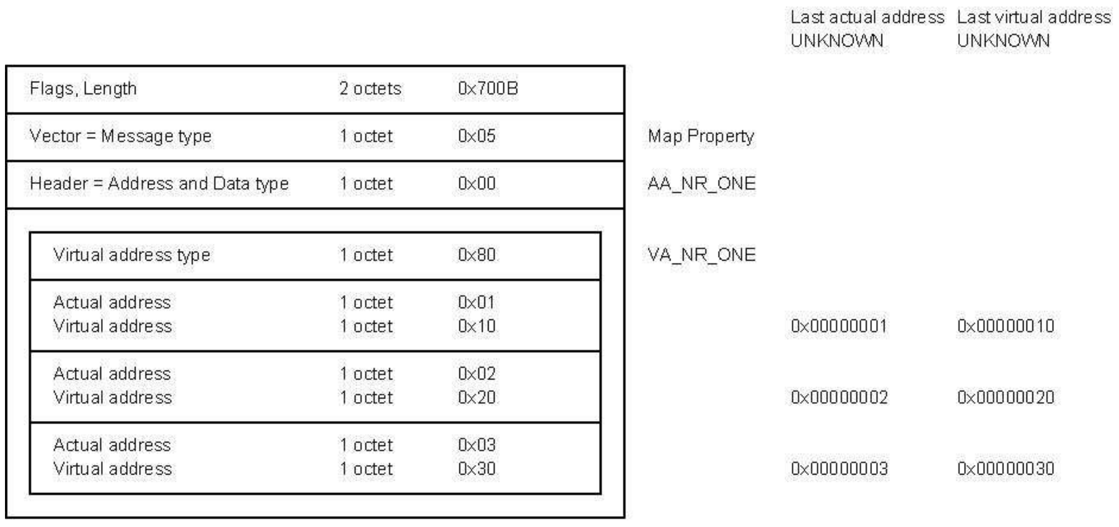
All other combinations are NOT allowed.

This document discusses each combination to see how it must be handled.

7.7.1.1 Ax_NR_xxx + Vx_NR_xxx

This is the simplest combination. There is ONE actual address and ONE virtual address in each pair. The actual address must be connected to the virtual address. There can be multiple pairs.

Example



				Last actual address	Last virtual address
				UNKNOWN	UNKNOWN
Flags, Length	2 octets	0x700B			
Vector = Message type	1 octet	0x05	Map Property		
Header = Address and Data type	1 octet	0x00	AA_NR_ONE		
Virtual address type	1 octet	0x80	VA_NR_ONE		
Actual address	1 octet	0x01			
Virtual address	1 octet	0x10		0x00000001	0x00000010
Actual address	1 octet	0x02			
Virtual address	1 octet	0x20		0x00000002	0x00000020
Actual address	1 octet	0x03			
Virtual address	1 octet	0x30		0x00000003	0x00000030

This DMP DPU contains 3 actual - virtual address pairs

The virtual map will be:

Actual Address	Virtual Address
0x01	0x10
0x02	0x20
0x03	0x30

7.7.1.2 Ax_NR_xxx + Vx_RS_xxx

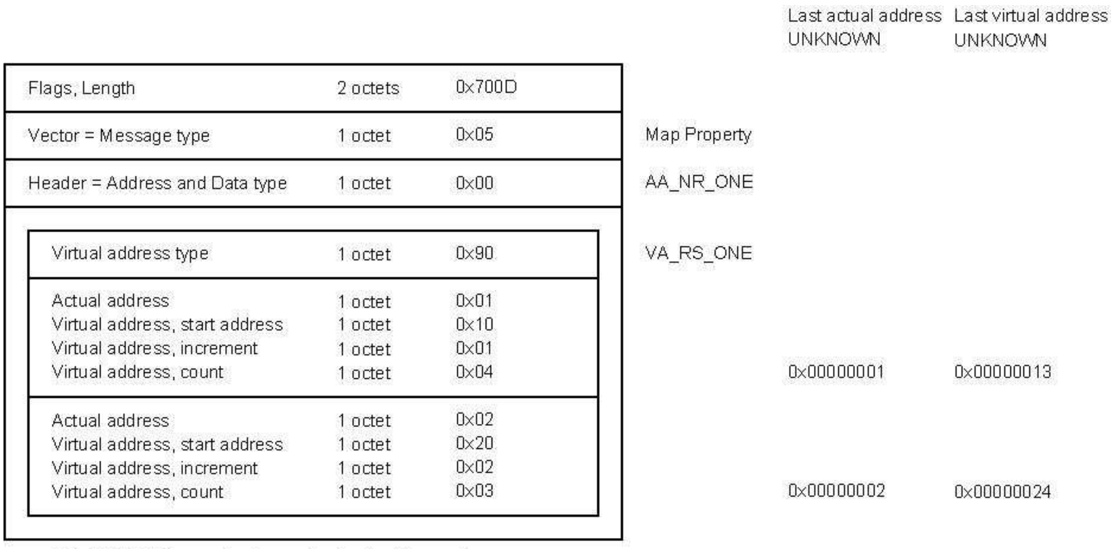
In this case there is ONE actual address and a RANGE of virtual addresses. So there can be more virtual addresses then there are actual addresses. A actual address may however be connected to only one virtual address.

I think this combination is not allowed but it is not obvious from the standard.

I have implemented it so that the code gives back each virtual address in the range with the same actual address. The virtual mapping code will just remap the actual address to the new virtual address. So the actual address will be mapped to the last virtual address in the range.

There can be multiple pairs with each pair containing ONE actual address and a RANGE of virtual addresses.

Example



				Last actual address	Last virtual address
				UNKNOWN	UNKNOWN
Flags, Length	2 octets	0x700D			
Vector = Message type	1 octet	0x05	Map Property		
Header = Address and Data type	1 octet	0x00	AA_NR_ONE		
Virtual address type	1 octet	0x90	VA_RS_ONE		
Actual address	1 octet	0x01			
Virtual address, start address	1 octet	0x10			
Virtual address, increment	1 octet	0x01			
Virtual address, count	1 octet	0x04		0x00000001	0x00000013
Actual address	1 octet	0x02			
Virtual address, start address	1 octet	0x20			
Virtual address, increment	1 octet	0x02			
Virtual address, count	1 octet	0x03		0x00000002	0x00000024

This DMP DPU contains 2 actual - virtual address pairs

The virtual map will be:

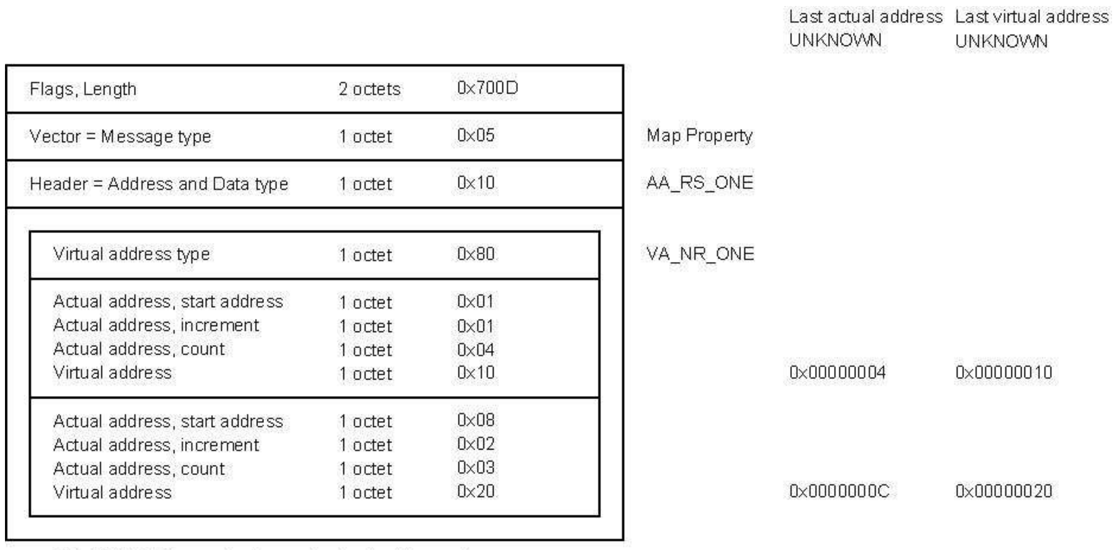
Actual Address	Virtual Address	Remark
0x01	0x10	Overwritten by next combination
0x01	0x11	Overwritten by next combination
0x01	0x12	Overwritten by next combination
0x01	0x13	This will be used
0x02	0x20	Overwritten by next combination
0x02	0x22	Overwritten by next combination
0x02	0x24	This will be used

7.7.1.3 Ax_RS_xxx + Vx_NR_xxx

In this case there is RANGE actual address and ONE virtual addresses. In this case all the actual addresses in the range are connected to the same virtual address.

There can be multiple pairs with each pair containing a RANGE of actual addresses and ONE virtual address.

Example



				Last actual address	Last virtual address
				UNKNOWN	UNKNOWN
Flags, Length	2 octets	0x700D			
Vector = Message type	1 octet	0x05	Map Property		
Header = Address and Data type	1 octet	0x10	AA_RS_ONE		
Virtual address type	1 octet	0x80	VA_NR_ONE		
Actual address, start address	1 octet	0x01			
Actual address, increment	1 octet	0x01			
Actual address, count	1 octet	0x04			
Virtual address	1 octet	0x10		0x00000004	0x00000010
Actual address, start address	1 octet	0x08			
Actual address, increment	1 octet	0x02			
Actual address, count	1 octet	0x03			
Virtual address	1 octet	0x20		0x0000000C	0x00000020

This DMP DPU contains 2 actual - virtual address pairs

The virtual map will be:

Actual Address	Virtual Address
0x01	0x10
0x02	0x10
0x03	0x10
0x04	0x10
0x08	0x20
0x0A	0x20
0x0C	0x20

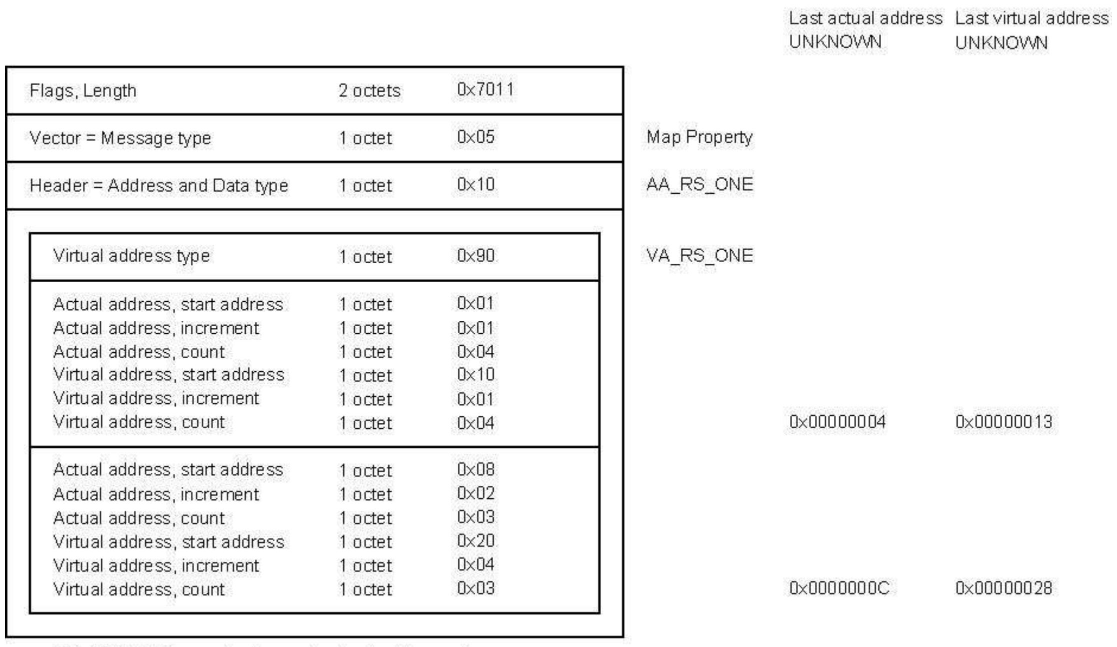
7.7.1.4 Ax_RS_xxx + Vx_RS_xxx

In this case there is RANGE actual address and RANGE virtual addresses. In this case all the actual addresses in the range are connected to corresponding virtual addresses in the range. Both ranges must have the same amount of items (counts must be equal).

So the code must step over both ranges at the same moment and connect the corresponding addresses.

There can be multiple pairs with each pair containing a RANGE of actual addresses and RANGE of virtual address.

Example



				Last actual address	Last virtual address
				UNKNOWN	UNKNOWN
Flags, Length	2 octets	0x7011			
Vector = Message type	1 octet	0x05	Map Property		
Header = Address and Data type	1 octet	0x10	AA_RS_ONE		
Virtual address type	1 octet	0x90	VA_RS_ONE		
Actual address, start address	1 octet	0x01			
Actual address, increment	1 octet	0x01			
Actual address, count	1 octet	0x04			
Virtual address, start address	1 octet	0x10			
Virtual address, increment	1 octet	0x01			
Virtual address, count	1 octet	0x04		0x00000004	0x00000013
Actual address, start address	1 octet	0x08			
Actual address, increment	1 octet	0x02			
Actual address, count	1 octet	0x03			
Virtual address, start address	1 octet	0x20			
Virtual address, increment	1 octet	0x04			
Virtual address, count	1 octet	0x03		0x0000000C	0x00000028

This DMP DPU contains 2 actual - virtual address pairs

The virtual map will be:

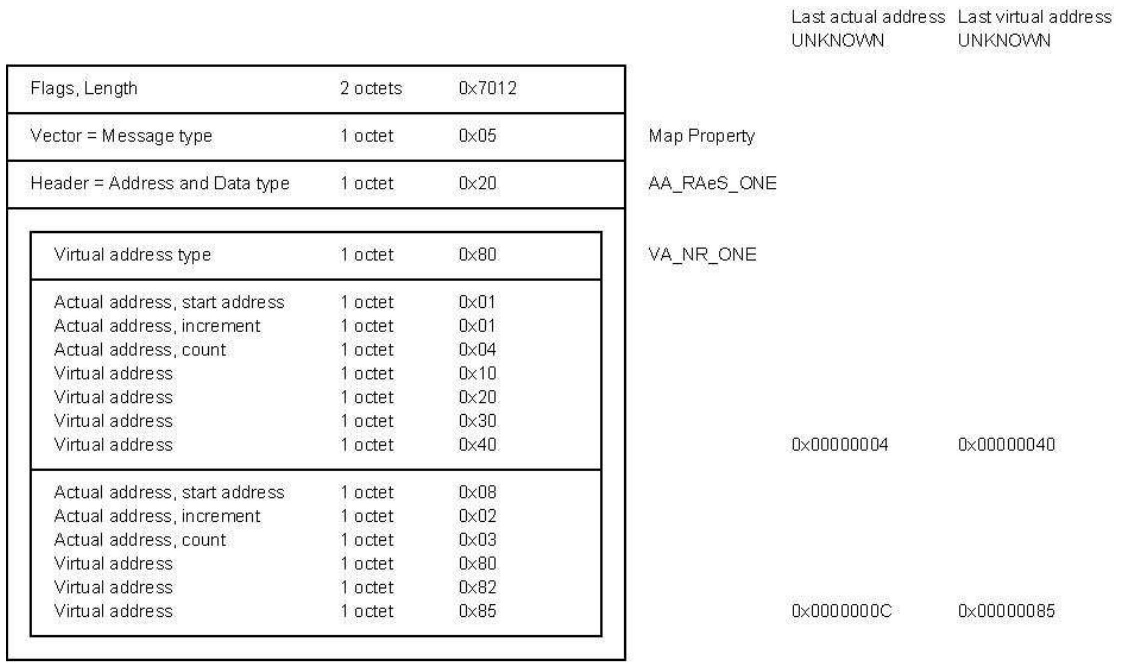
Actual Address	Virtual Address
0x01	0x10
0x02	0x11
0x03	0x12
0x04	0x13
0x08	0x20
0x0A	0x24
0x0C	0x28

7.7.1.5 Ax_RAeS_xxx + Vx_NR_xxx

In this case there is RANGE actual address and multiple virtual address in the same pair. The number of virtual address is equal to the number of addresses in the actual address range(the count). Each actual address in the range is connected to one virtual address.

There can be multiple pairs.

Example



				Last actual address	Last virtual address
				UNKNOWN	UNKNOWN
Flags, Length	2 octets	0x7012			
Vector = Message type	1 octet	0x05	Map Property		
Header = Address and Data type	1 octet	0x20	AA_RAeS_ONE		
Virtual address type	1 octet	0x80	VA_NR_ONE		
Actual address, start address	1 octet	0x01			
Actual address, increment	1 octet	0x01			
Actual address, count	1 octet	0x04			
Virtual address	1 octet	0x10			
Virtual address	1 octet	0x20			
Virtual address	1 octet	0x30			
Virtual address	1 octet	0x40		0x00000004	0x00000040
Actual address, start address	1 octet	0x08			
Actual address, increment	1 octet	0x02			
Actual address, count	1 octet	0x03			
Virtual address	1 octet	0x80			
Virtual address	1 octet	0x82			
Virtual address	1 octet	0x85		0x0000000C	0x00000085

This DMP DPU contains 2 actual - virtual address pairs

The virtual map will be:

Actual Address	Virtual Address
0x01	0x10
0x02	0x20
0x03	0x30
0x04	0x40
0x08	0x80
0x0A	0x82
0x0C	0x85

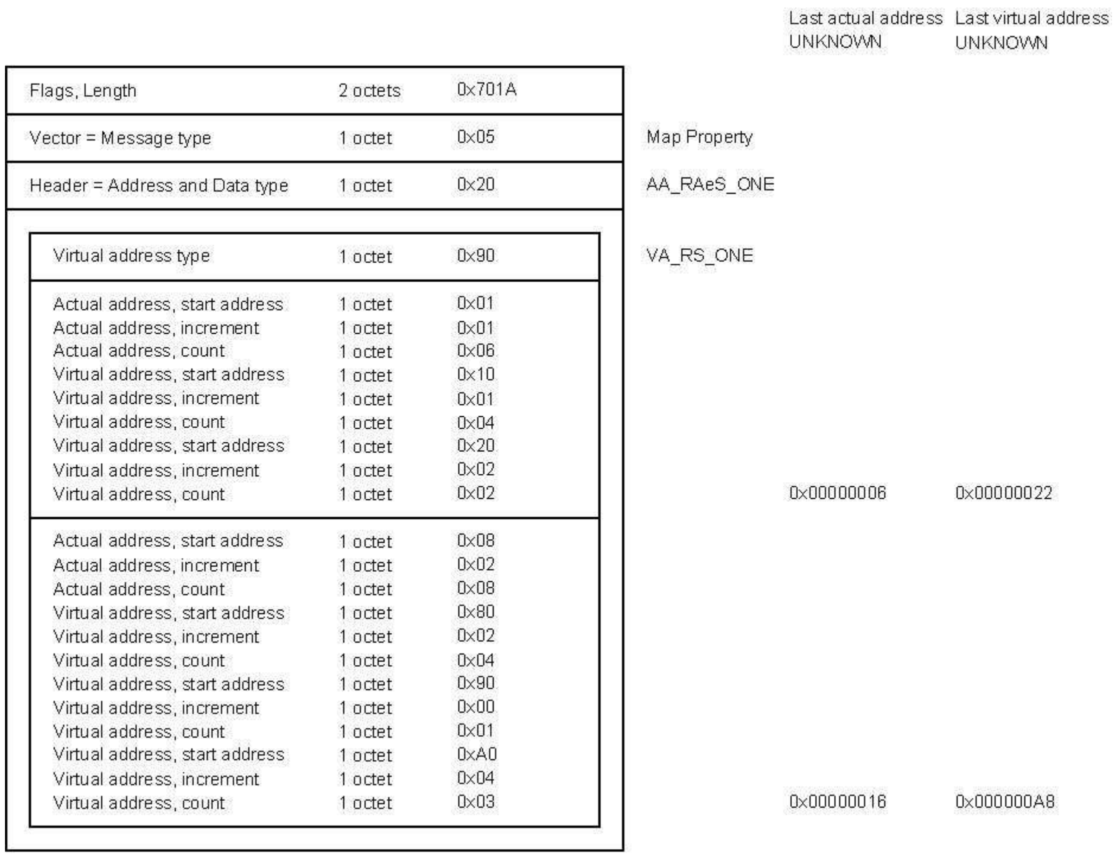
7.7.1.6 Ax_RAeS_xxx + Vx_RS_xxx

In this case there is RANGE actual address and multiple RANGES of virtual address in the same pair. The total number of virtual address must be equal to the number of addresses in the actual address range(the count).

The code must step over both ranges at the same moment and connected the corresponding addresses. When all addresses in a virtual address range are handled, the next range must be used.

There can be multiple pairs.

Example



				Last actual address	Last virtual address
				UNKNOWN	UNKNOWN
Flags, Length	2 octets	0x701A			
Vector = Message type	1 octet	0x05	Map Property		
Header = Address and Data type	1 octet	0x20	AA_RAeS_ONE		
Virtual address type	1 octet	0x90	VA_RS_ONE		
Actual address, start address	1 octet	0x01			
Actual address, increment	1 octet	0x01			
Actual address, count	1 octet	0x06			
Virtual address, start address	1 octet	0x10			
Virtual address, increment	1 octet	0x01			
Virtual address, count	1 octet	0x04			
Virtual address, start address	1 octet	0x20			
Virtual address, increment	1 octet	0x02			
Virtual address, count	1 octet	0x02		0x00000006	0x00000022
Actual address, start address	1 octet	0x08			
Actual address, increment	1 octet	0x02			
Actual address, count	1 octet	0x08			
Virtual address, start address	1 octet	0x80			
Virtual address, increment	1 octet	0x02			
Virtual address, count	1 octet	0x04			
Virtual address, start address	1 octet	0x90			
Virtual address, increment	1 octet	0x00			
Virtual address, count	1 octet	0x01			
Virtual address, start address	1 octet	0xA0			
Virtual address, increment	1 octet	0x04			
Virtual address, count	1 octet	0x03		0x00000016	0x000000A8

This DMP DPU contains 2 actual - virtual address pairs

The virtual map will be:

Actual Address	Virtual Address
0x01	0x10
0x02	0x11
0x03	0x12
0x04	0x13
0x05	0x20
0x06	0x22
0x08	0x80
0x0A	0x82
0x0C	0x84
0x0E	0x86
0x10	0x90
0x12	0xA0
0x14	0xA4
0x16	0xA8

7.7.2 Address – Reason Code pairs

The data used in a Get property fail, Set Proper fail, Map property fail and Subscribe reject messages uses Address-Reason Code pairs.

It is not obvious from [DMP] of the Set Property Fail message may contain virtual addresses. There is no rule that prevents this. However in the table **14 Allowed Data and Address Combinations** only actual addresses area allowed. This would also mean that when a virtual address is used in a Set Property message, and there is some failure, the Set Property Fail message must use the connected actual address(es) to report the reasons.

The Map Property Fail message has also no rule to prevent virtual addresses. It is however must logical to use only actual addresses, because the Map property also supports only actual addresses.

The conclusion is that the address in the Address-Reason code pair, will never be a virtual address. This means that only the following formats are accepted (with xxx = ONE, TWO or FOUR):

- AA_NR_xxx
- AR_NR_xxx
- AA_RS_xxx
- AR_RS_xxx
- AA_RAeS_xxx
- AR_RAeS_xxx
- AA_RSmS_xxx
- AR_RSmS_xxx

The Ax_RAeS_xxx and Ax_RSmS_xxx format must be the same because there can't be mixed sized items. All items have the same size, one octect, the length of a reason code.

The following tables give the different combinations for a Address – Reason pair:

Address	Ax_NR_xxx
Reason	One code

Start address	Ax_RS_xxx
Increment	
Count	
Reason	One code

Start address	Ax_RAeS_xxx or Ax_RSmS_xxx
Increment	
Count	
Reason	Count codes
Reason	
Reason	

8 Range problems

The [DMP] document states that the count and increment of a range address type (xx_Rxxx_xxx) may be 0. It is however not obvious how this must be handled. I define here the way **I think** they must be handled and how it is handled in my code.

8.1 Case 1: Count=0

In this case the range doesn't have any addresses. This means the whole block is ignored. The last address isn't updated. It is like the address range wasn't present. So there may also be NO data(property values, reason code, …) present for the range.

8.2 Case 2: Count !=0, increment=0

In this case the range does have the given number of addresses, but they will all be of the same address value. However there can be multiple data(property value, reason codes, …) items that are present in the data, so they must be handled. They will all be given to the same address. In this case the last address is also updated.

Example:

Start address	Actual/Absolute 0x00000001
Increment	0
Count	2
Data	0x0123
Data	0x5678

When this is used as data for the Set Property message type, the address 0x00000001 will first be set to value 0x0123 and then to value 0x5678. The last actual address will be 0x00000001.

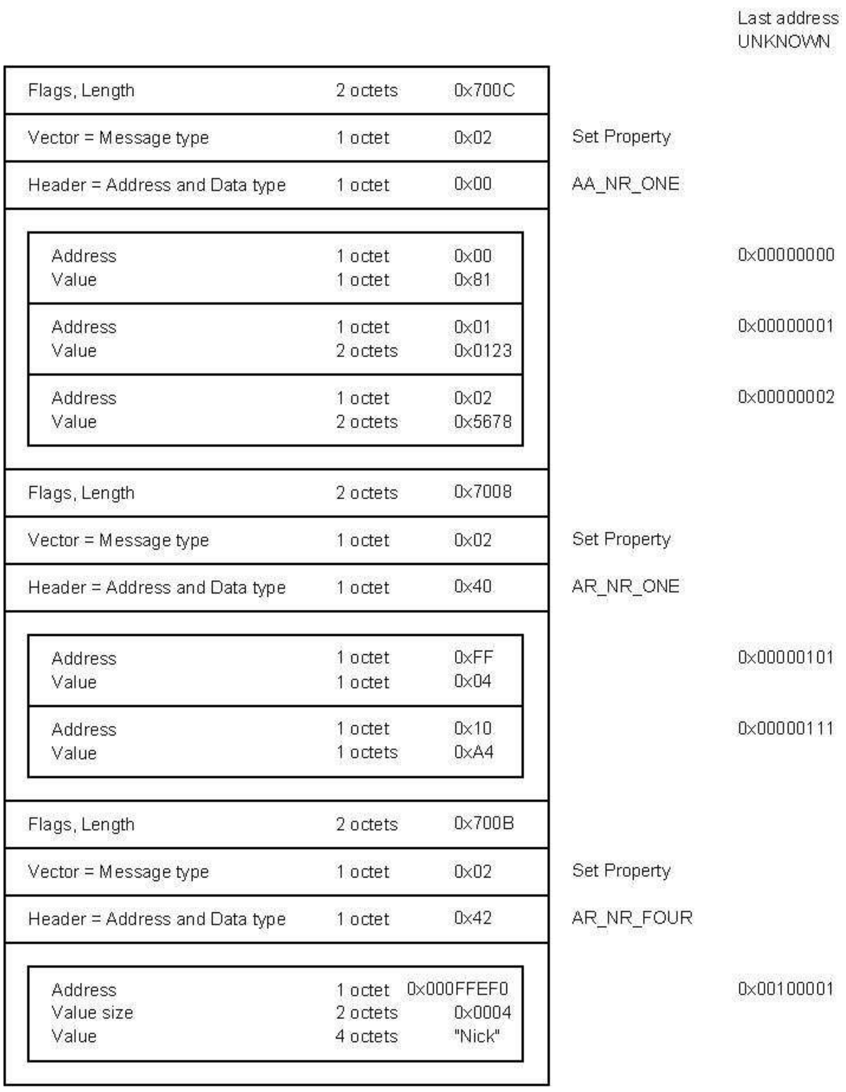
When the address is a relative address and the start address is 0, the last address will be used.

9 Examples on Moving Head

This chapter shows some examples of DMP PDU's that are used for the moving head example.

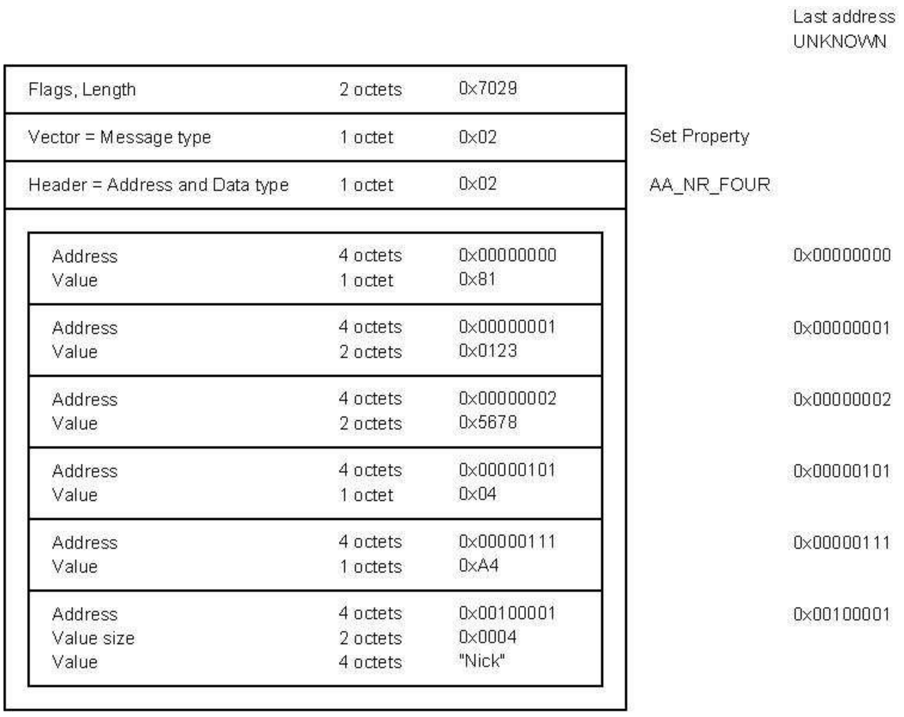
9.1 Absolute + relative

We can extent the example in 7.6 Property Address-Data Pair to also set the other properties. The example uses relative addresses to demonstrate their use..



				Last address UNKNOWN
Flags, Length	2 octets	0x700C		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x00	AA_NR_ONE	
Address Value	1 octet 1 octet	0x00 0x81		0x00000000
Address Value	1 octet 2 octets	0x01 0x0123		0x00000001
Address Value	1 octet 2 octets	0x02 0x5678		0x00000002
Flags, Length	2 octets	0x7008		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x40	AR_NR_ONE	
Address Value	1 octet 1 octet	0xFF 0x04		0x00000101
Address Value	1 octet 1 octets	0x10 0xA4		0x00000111
Flags, Length	2 octets	0x700B		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x42	AR_NR_FOUR	
Address Value size Value	1 octet 2 octets 4 octets	0x000FFEF0 0x0004 "Nick"		0x00100001

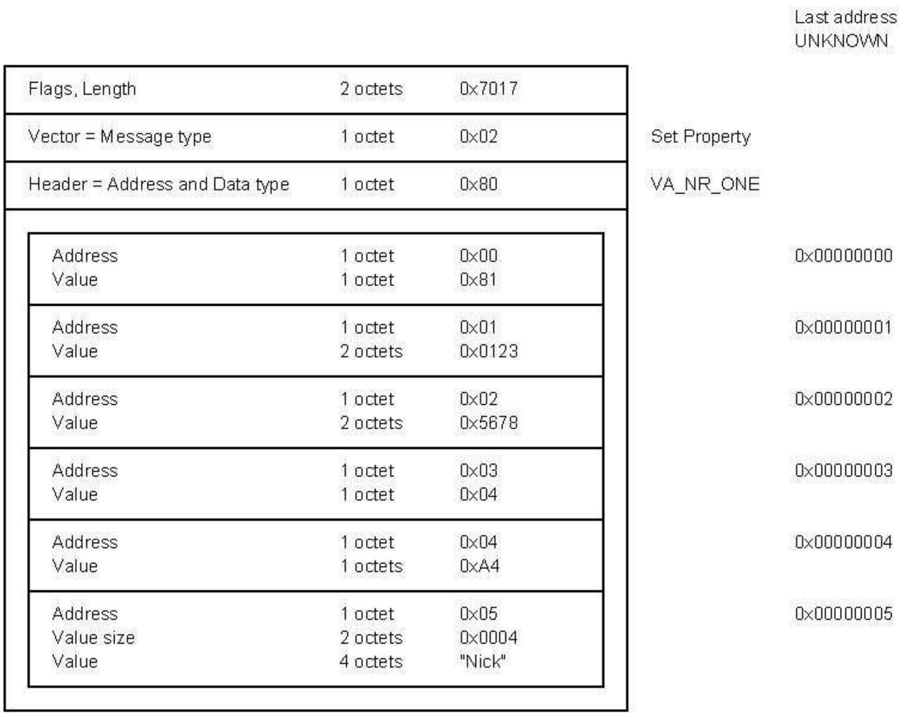
The next DMP PDU performs the same thing. It is much simpler, however it uses 41 bytes while the previous example uses only 34 bytes.



				Last address UNKNOWN
Flags, Length	2 octets	0x7029		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x02	AA_NR_FOUR	
Address	4 octets	0x00000000		0x00000000
Value	1 octet	0x81		
Address	4 octets	0x00000001		0x00000001
Value	2 octets	0x0123		
Address	4 octets	0x00000002		0x00000002
Value	2 octets	0x5678		
Address	4 octets	0x00000101		0x00000101
Value	1 octet	0x04		
Address	4 octets	0x00000111		0x00000111
Value	1 octets	0xA4		
Address	4 octets	0x00100001		0x00100001
Value size	2 octets	0x0004		
Value	4 octets	"Nick"		

This demonstrates the power of DMP to compact messages.

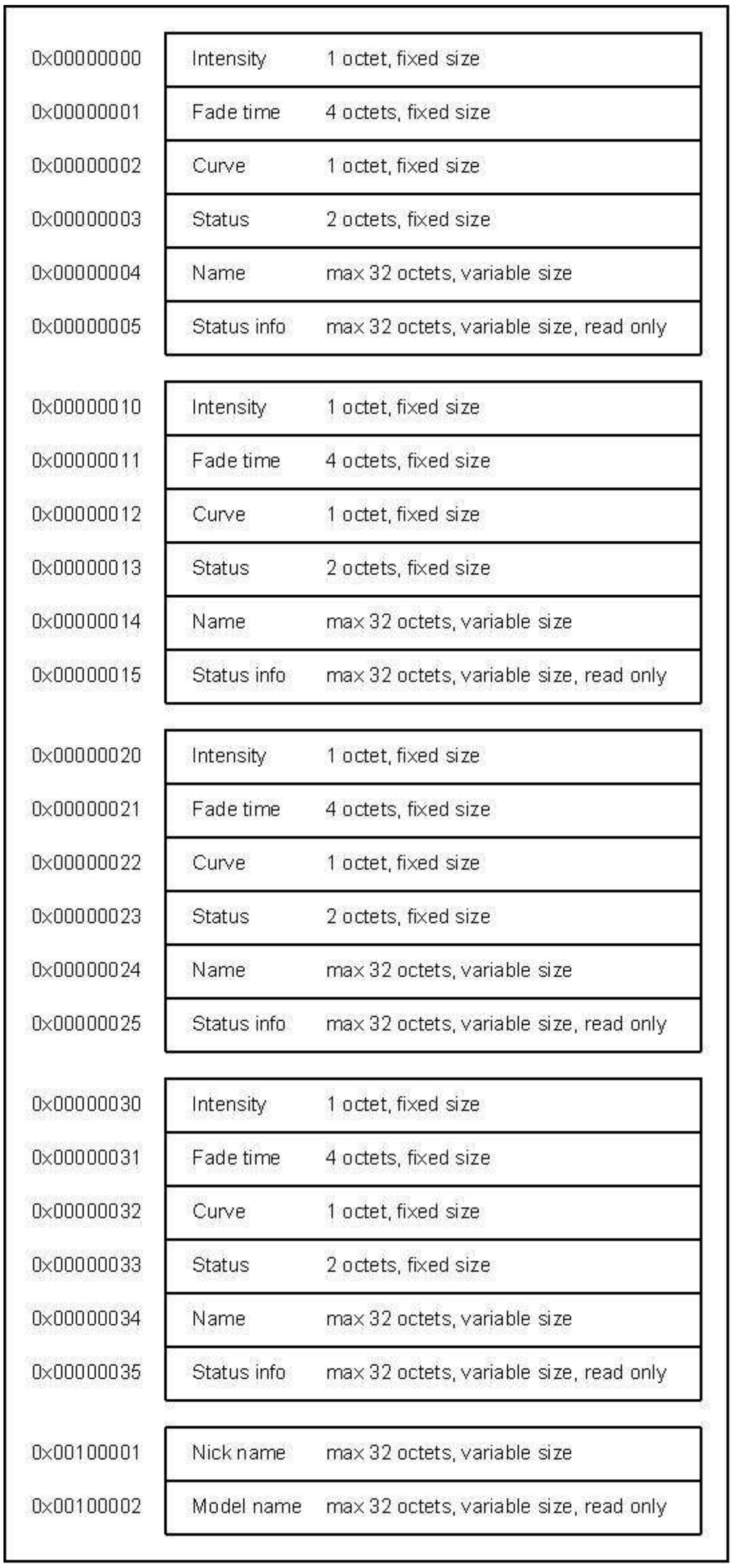
When the device has a virtual map like given in **7.1 Virtual/actual field** the following DMP PDU can be used to set all properties. It uses the virtual addresses.



				Last address UNKNOWN
Flags, Length	2 octets	0x7017		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x80	VA_NR_ONE	
Address	1 octet	0x00		0x00000000
Value	1 octet	0x81		
Address	1 octet	0x01		0x00000001
Value	2 octets	0x0123		
Address	1 octet	0x02		0x00000002
Value	2 octets	0x5678		
Address	1 octet	0x03		0x00000003
Value	1 octet	0x04		
Address	1 octet	0x04		0x00000004
Value	1 octets	0xA4		
Address	1 octet	0x05		0x00000005
Value size	2 octets	0x0004		
Value	4 octets	"Nick"		

10 Examples with dimmer

To demonstrate the other address formats this document uses a example dimmer with the following address map.

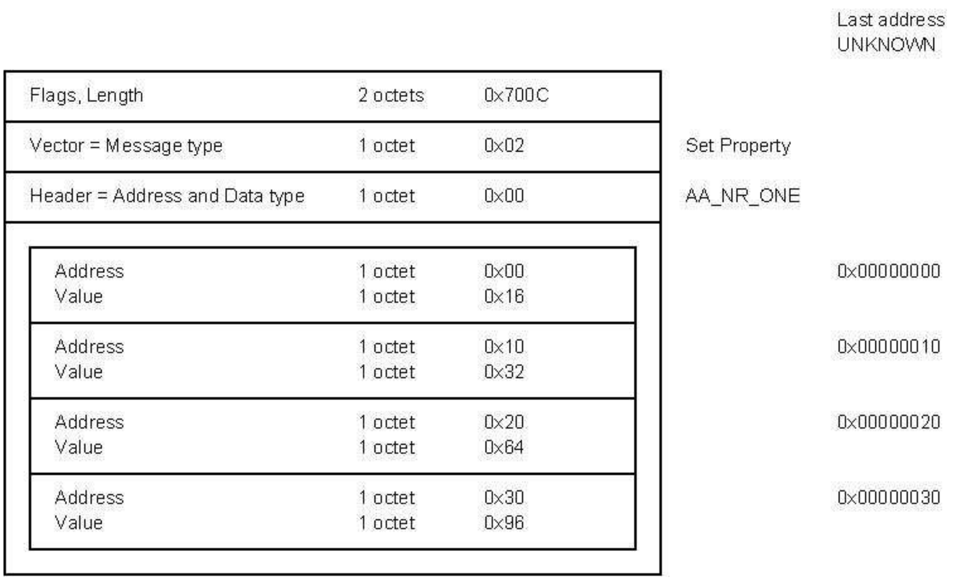


0x00000000	Intensity	1 octet, fixed size
0x00000001	Fade time	4 octets, fixed size
0x00000002	Curve	1 octet, fixed size
0x00000003	Status	2 octets, fixed size
0x00000004	Name	max 32 octets, variable size
0x00000005	Status info	max 32 octets, variable size, read only
0x00000010	Intensity	1 octet, fixed size
0x00000011	Fade time	4 octets, fixed size
0x00000012	Curve	1 octet, fixed size
0x00000013	Status	2 octets, fixed size
0x00000014	Name	max 32 octets, variable size
0x00000015	Status info	max 32 octets, variable size, read only
0x00000020	Intensity	1 octet, fixed size
0x00000021	Fade time	4 octets, fixed size
0x00000022	Curve	1 octet, fixed size
0x00000023	Status	2 octets, fixed size
0x00000024	Name	max 32 octets, variable size
0x00000025	Status info	max 32 octets, variable size, read only
0x00000030	Intensity	1 octet, fixed size
0x00000031	Fade time	4 octets, fixed size
0x00000032	Curve	1 octet, fixed size
0x00000033	Status	2 octets, fixed size
0x00000034	Name	max 32 octets, variable size
0x00000035	Status info	max 32 octets, variable size, read only
0x00100001	Nick name	max 32 octets, variable size
0x00100002	Model name	max 32 octets, variable size, read only

This example shows the extra possibilities that ACN delivers. The dimmer pack has four channels. Each channel has six properties. The dimmer pack has also two general properties.

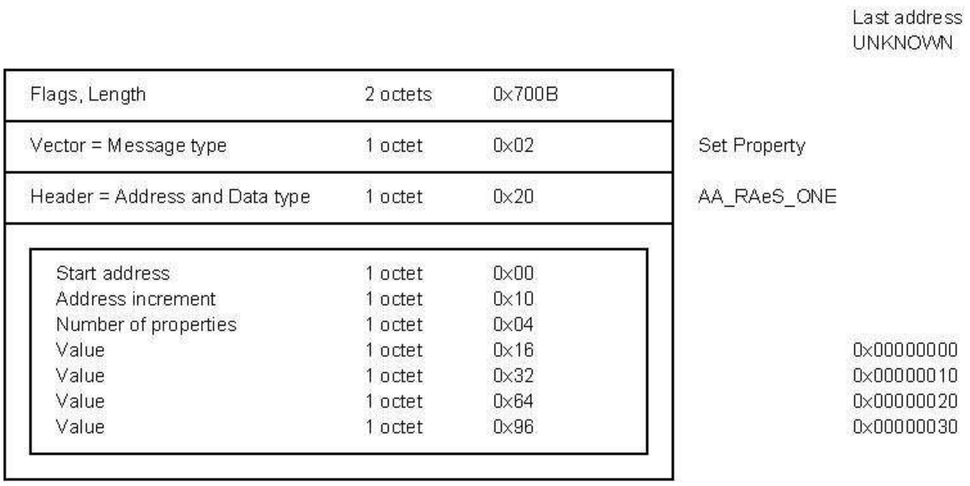
10.1 Setting dimmer values

The most obvious way to set four different intensities on the dimmers is the following DMP PDU.



				Last address UNKNOWN
Flags, Length	2 octets	0x700C		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x00	AA_NR_ONE	
Address	1 octet	0x00		0x00000000
Value	1 octet	0x16		
Address	1 octet	0x10		0x00000010
Value	1 octet	0x32		
Address	1 octet	0x20		0x00000020
Value	1 octet	0x64		
Address	1 octet	0x30		0x00000030
Value	1 octet	0x96		

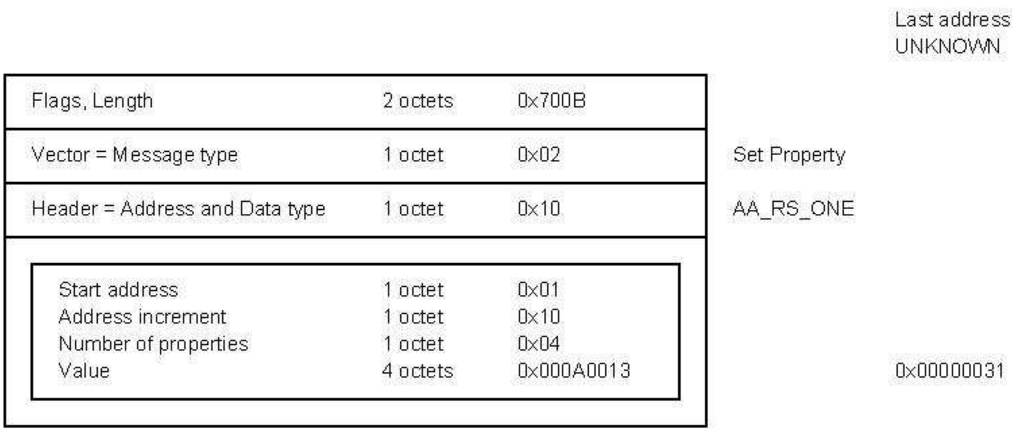
There is however a compacter way when using the Range, multiple data, same size (AA_RAeS_ONE) format.



				Last address UNKNOWN
Flags, Length	2 octets	0x700B		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x20	AA_RAeS_ONE	
Start address	1 octet	0x00		
Address increment	1 octet	0x10		
Number of properties	1 octet	0x04		
Value	1 octet	0x16		0x00000000
Value	1 octet	0x32		0x00000010
Value	1 octet	0x64		0x00000020
Value	1 octet	0x96		0x00000030

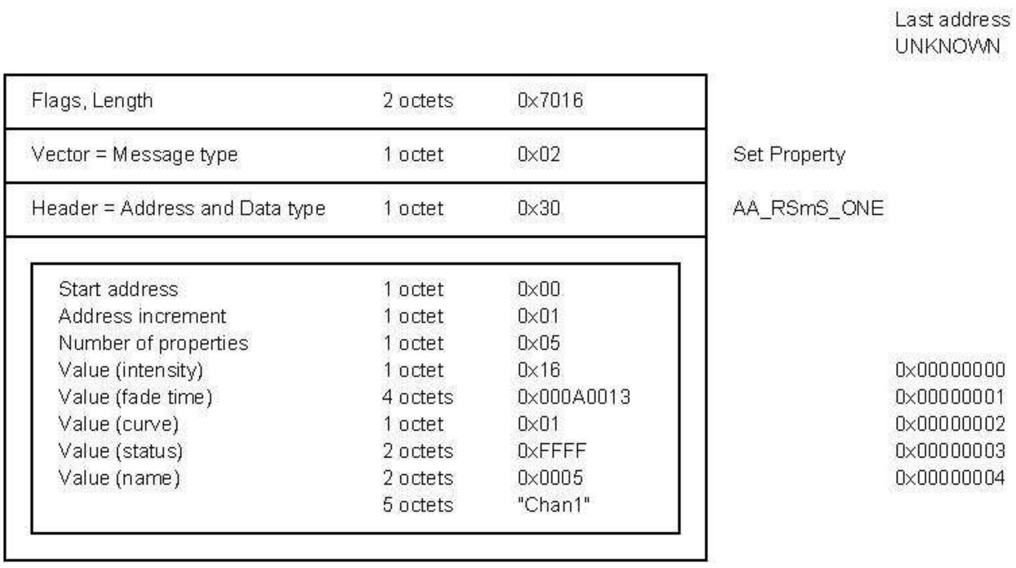
It is only one byte less, but the more channels there are, the more space would be saved.

When all the fade times must be set to the same value, it is possible to use the Range, single data item (AA_RS_ONE) format.



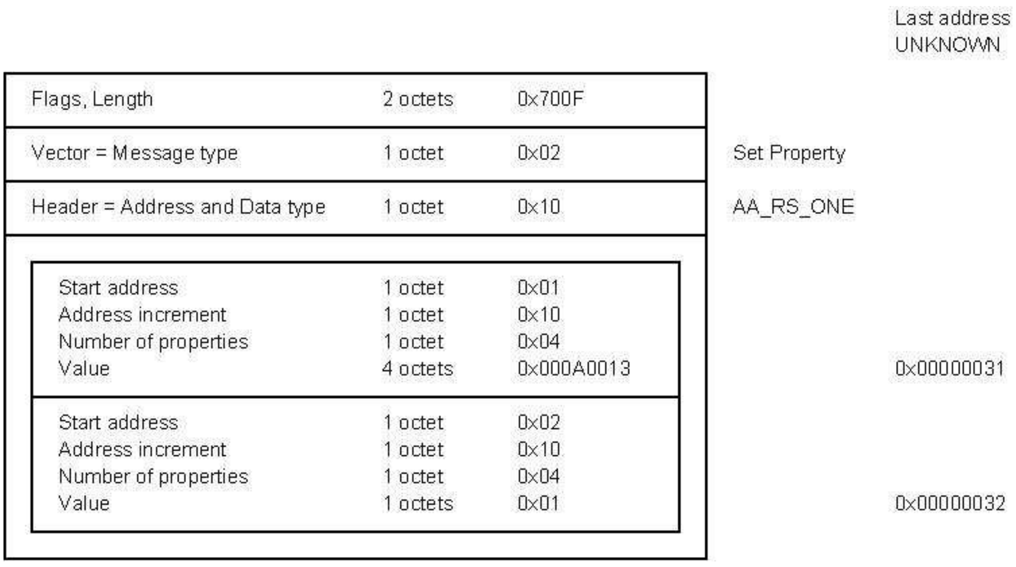
				Last address UNKNOWN
Flags, Length	2 octets	0x700B		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x10	AA_RS_ONE	
Start address	1 octet	0x01		
Address increment	1 octet	0x10		
Number of properties	1 octet	0x04		
Value	4 octets	0x000A0013		0x00000031

It is also possible to set all properties of one channel using the Range, multiple data items, mixed size (AA_RSmS_ONE) format.



				Last address UNKNOWN
Flags, Length	2 octets	0x7016		
Vector = Message type	1 octet	0x02	Set Property	
Header = Address and Data type	1 octet	0x30	AA_RSmS_ONE	
Start address	1 octet	0x00		
Address increment	1 octet	0x01		
Number of properties	1 octet	0x05		
Value (intensity)	1 octet	0x16		0x00000000
Value (fade time)	4 octets	0x000A0013		0x00000001
Value (curve)	1 octet	0x01		0x00000002
Value (status)	2 octets	0xFFFF		0x00000003
Value (name)	2 octets	0x0005		0x00000004
	5 octets	"Chan1"		

It is also possible to have multiple Property address – Data value pairs in ranges, like in the next example that sets all fade times and curves to the same value.



			Last address UNKNOWN
Flags, Length	2 octets	0x700F	
Vector = Message type	1 octet	0x02	Set Property
Header = Address and Data type	1 octet	0x10	AA_RS_ONE
Start address	1 octet	0x01	
Address increment	1 octet	0x10	
Number of properties	1 octet	0x04	
Value	4 octets	0x000A0013	0x00000031
Start address	1 octet	0x02	
Address increment	1 octet	0x10	
Number of properties	1 octet	0x04	
Value	1 octets	0x01	0x00000032

The same can be done with the other range types.

11 Conclusion

This document gives examples and guidelines about the DMP protocol and more specific the way DMP PDU's must be created and parsed. I hoped this information makes it easier to understand the [DMP] document. This document doesn't describe the different message types. That information can be found in the [DMP] document.

I think this document makes clear that it is impossible to create a controller without code to read a DDL file. The DDL file is needed to find the types and sizes of properties. That information is needed to create DMP PDU's.

I am going to use the information in this document and in [DMP] to create a set of C++ classes that can be used to represent and parse DMP PDU's. This classes will be open source so that others can use them to experiment with DMP.