



DRAFT FOR WITHDRAWAL REVIEW

**ANSI E1.30-4 – 2010 (R2021)
EPI 26. Device Description Language (DDL)
Extensions for DMX512 and E1.31 Devices**

Document number CP/2008-1007r7

This standard was approved as an American National Standard by ANSI's Board of Standards Review on _____

Copyright © 2021 the Entertainment Services and Technology Association (ESTA)

All rights reserved.

NOTICE and DISCLAIMER

ESTA does not approve, inspect, or certify any installations, procedures, equipment or materials for compliance with codes, recommended practices or standards. Compliance with a ESTA standard or an American National Standard developed by ESTA is the sole and exclusive responsibility of the manufacturer or provider and is entirely within their control and discretion. Any markings, identification or other claims of compliance do not constitute certification or approval of any type or nature whatsoever by ESTA.

ESTA neither guarantees nor warrants the accuracy or completeness of any information published herein and disclaims liability for any personal injury, property or other damage or injury of any nature whatsoever, whether special, indirect, consequential or compensatory, directly or indirectly resulting from the publication, use of, or reliance on this document. In issuing and distributing this document.

In issuing this document, ESTA does not either (a) undertake to render professional or other services for or on behalf of any person or entity, or (b) undertake any duty to any person or entity with respect to this document or its contents. Anyone using this document should rely on his or her own independent judgment or, as appropriate, seek the advice of a competent professional in determining the exercise of reasonable care in any given circumstance.

Published by:

Entertainment Services and Technology Association
P.O. Box 23200
Brooklyn, NY 11202-3200
USA
Phone: 1-212-244-1505
standards@esta.org

The ESTA Technical Standards Program

The ESTA Technical Standards Program was created to serve the ESTA membership and the entertainment industry in technical standards related matters. The goal of the Program is to take a leading role regarding technology within the entertainment industry by creating recommended practices and standards, monitoring standards issues around the world on behalf of our members, and improving communications and safety within the industry. ESTA works closely with the technical standards efforts of other organizations within our industry, including USITT and VPLT, as well as representing the interests of ESTA members to ANSI, UL, and the NFPA. The Technical Standards Program is accredited by the American National Standards Institute.

The Technical Standards Council (TSC) was established to oversee and coordinate the Technical Standards Program. Made up of individuals experienced in standards-making work from throughout our industry, the Council approves all projects undertaken and assigns them to the appropriate working group. The Technical Standards Council employs Technical Standards Management staff to coordinate the work of the Council and its working groups as well as maintain a “Standards Watch” on behalf of members. Working groups include: Control Protocols, Electrical Power, Event Safety, Floors, Fog and Smoke, Followspot Position, Photometrics, Rigging, and Stage Machinery.

ESTA encourages active participation in the Technical Standards Program. There are several ways to become involved. If you would like to become a member of an existing working group, as have over four hundred people, you must complete an application which is available from the ESTA office. Your application is subject to approval by the working group and you will be required to actively participate in the work of the group. This includes responding to letter ballots and attending meetings. Membership in ESTA is not a requirement. You can also become involved by requesting that the TSC develop a standard or a recommended practice in an area of concern to you.

The Control Protocols Working Group, which authored this standard, consists of a cross section of entertainment industry professionals representing a diversity of interests. ESTA is committed to developing consensus-based standards and recommended practices in an open setting.

Investors in Innovation

The Technical Standard Program (TSP) is financially supported by companies and individuals who make undirected donations to the TSP. Contributing companies and individuals who have helped fund the TSP are recognized as “Investors in Innovation.” The Investors in Innovation when this standard was published include these companies and individuals:

[This is a placeholder for the table]

Extraordinary legacy gift: Ken Vannice

All donations to the TSP support the Technical Standards Program in general and are not directed. You can make a donation by visiting https://tsp.esta.org/tsp/inv_in_innovation/sponsor.html. Become an *Investor in Innovation!*

Contact Information**Technical Standards Manager**

Karl G. Ruling
ESTA
PO Box 23200
New York, NY 11202-3200
USA
+1-212-244-1505 x703
karl.ruling@esta.org

Assistant Technical Standards Manager

Richard J. Nix
ESTA
PO Box 23200
New York, NY 11202-3200
USA
+1-212-244-1505 x649
richard.nix@esta.org

Technical Standards Council Chairpersons

Mike Garl
Mike Garl Consulting LLC
+1-865-389-4371
mike@mikegarlconsulting.com

Mike Wood
Mike Wood Consulting LLC
+1-512-288-4916
mike@mikewoodconsulting.com

Control Protocols Working Group Co-chairpersons

Milton Davis
Doug Fleenor Design, Inc.
+1-805-481-9599
milton@dfd.com

Michael Lay
Candela Controls, Inc
+1-352-433-2479
michael_m_lay@yahoo.com

Acknowledgments

The Control Protocols Working Group members when this document was approved by the working group on 20 April 2021 are shown below.

Voting members:**Observer (non-voting) members:****Interest category codes:**

CP = Custom-market Producer
DE = Designer
DR = Dealer or Rental company
G = General interest
MP = Mass-market Producer
U = User

Table of Contents

NOTICE and DISCLAIMER.....	i
Investors in Innovation.....	iii
Acknowledgments.....	vii
Table of Contents.....	x
Abstract.....	1
Foreword – ACN EPIs.....	1
1 Introductory Discussion.....	1
1.1 DMX512.....	1
1.2 Versions of DMX512, E1.31 and DMX512 over Ethernet Protocols.....	1
1.3 DDL.....	1
1.4 Multiple Interfaces.....	1
1.5 Input vs Output.....	2
1.6 Characteristics of DMX512 Access.....	2
2 DDL Properties and DMX512 Slots.....	2
2.1 Interlocks and Exclusions.....	3
2.2 Examples.....	3
3 DMX512 Slot Addressing.....	3
4 Syntax for DMX512 Addressing.....	4
4.1 Identification of This Protocol.....	4
4.2 Property Map Elements.....	4
4.3 Statements.....	5
4.3.1 Compound Statements.....	5
4.3.2 Conditional Statements.....	5
4.4 Procedures.....	5
4.4.1 Setslot().....	5
4.4.2 next().....	7
4.4.3 wait().....	7
4.4.4 Order of Execution.....	8
4.5 Expressions.....	8
4.5.1 Variables.....	8
4.5.2 Literals.....	8
4.5.3 Operators.....	9
4.6 Examples.....	9
Annex A. Normative References.....	13

Abstract

This EPI defines protocol specific extensions to Device Description Language for describing DMX devices.

Foreword – ACN EPIs

E1.17-200x is the “ESTA Architecture for Control Networks” standard [ACN]. It specifies an architecture – including a suite of protocols and languages that may be configured and combined with other standard protocols in a number of ways to form flexible networked control systems.

E1.17 Profiles for Interoperability (EPIs) are standards documents that specify how conforming implementations are to operate in a particular environment or situation in order to guarantee interoperability. They may specify a single technique, set of parameters or requirement for the various ACN components. They may also specify how other standards (including other EPIs) either defined within ACN or externally are to be used to ensure interoperability.

1 Introductory Discussion

1.1 DMX512

In the years since its introduction in 1986, USITT DMX512 (often referred to as DMX512 or just DMX) has spread from its initial remit of providing simple level control to dimmers, to control of a huge range of devices. The most recent standard specification is DMX512-A [DMX512]. Because the semantics of controlling anything except dimmers are outside the scope of any of the DMX512 specifications, there is no standard way to arrange control of other items. However, other DMX512 devices are prevalent throughout the lighting industry and are characterized by a wide range of control styles and methods.

1.2 Versions of DMX512, E1.31 and DMX512 over Ethernet Protocols

The first standard for DMX512 was published in 1986. Subsequent standards have updated this and ESTA has developed a protocol for transport of DMX512 data over IP networks contained in the E1.31 standard [E1.31]. There are also a number of proprietary protocols in use that perform a similar function of transporting DMX512 data over Ethernet or other media. All these standards share the same NULL START Code data format and the syntax described here is intended to work with the latest standard DMX512-A, and also with earlier versions of DMX512, with E1.31 and with any other protocol which uses DMX512 NULL START Code data format. For this reason, the term DMX512 is used within this document to refer to any such protocol as applicable. When specific versions of protocols are referenced then the terms *DMX512-A*, *DMX512/1990*, *DMX512-1986*, *E1.31* are used ([DMX512-A], [DMX512/1990], [DMX512-1986], [E1.31]).

1.3 DDL

Device Description Language [DDL] provides a rich and extensible way to describe a wide range of controllable devices, and maintains a separation between the device model which it describes as a structure of properties, and the access protocol which is used to access some of those properties remotely. It provides the <protocol> element that allows any necessary extension for describing how the properties of the device model are accessed using any given protocol. This EPI defines the syntax of the protocol element used to describe DMX512 access to properties.

1.4 Multiple Interfaces

Historically, a DMX512 device (as opposed to a controller) usually had a single DMX512 interface for control. The advent of E1.31 [E1.31] means that it is much more common for devices to have multiple sources of DMX512 data – these may correspond with physical interfaces (e.g. one DMX512-A and one Ethernet interface) or may be multiple logical streams received on a single physical interface (as is common with E1.31), or both.

This EPI recognizes and expects that many devices will receive multiple concurrent sources of DMX512 data.

1.5 Input vs Output

DMX512 is a unidirectional protocol, however devices may have both DMX512 inputs and DMX512 outputs. This EPI only specifies a means to declare the way DMX512 inputs access device properties, because these are a

means to access and control a DDL device. DMX512 outputs can be described using *foreign* or *xeno-binding* behaviors. Properties specific to configuration or status of the DMX512 interface itself whether input or output, can be described using *netinterface* derived behaviors.

1.6 Characteristics of DMX512 Access

The extensions defined in this EPI only attempt to describe devices controlled using NULL START Code data as defined in [DMX512].

Because DMX512 is a unidirectional protocol, any property that is accessed by it is implicitly write-only. NULL START Code data provides an array of up to 512 eight-bit bytes that are called “Slots” in DMX512-A terminology. This array is repeatedly updated at a rate that is variable but must be faster than once per second and is typically of the order of 20 to 40 updates per second.

Usually a DMX512 device is configured with an “address” (sometimes called “start address” or “base address”), that is an index into the array. It then uses the value of that and some number of consecutive slots as its control data values. Other devices on the link will be configured with different addresses and will therefore take their control values from different slots in the array.

Because the number of available slots is quite limited and the eight-bit format is very rigid, many devices either pack multiple functions into a single slot, or use multiple slots for a single function. There is no standardized way to do this and to complicate things further, many more complex DMX512 devices implement control channels or similar mechanisms where special values in one slot change the meaning of other slots, or where one or more slots needs to take a sequence of values – often with some timing constraints – in order to trigger or access a function.

2 DDL Properties and DMX512 Slots

Because the functionality of DMX512 devices often does not correspond to full slots in DMX512 and may need sequences of values, a direct mapping between DMX512 slots and DDL properties is not sufficient. Furthermore, attempting to model devices by focusing on the DMX512 slot values very rapidly gives rise to hugely complex descriptions.

The strategy for DDL is therefore to model a device in DDL properties. These are virtual properties that correspond in a direct way to functions within a device but often do not correspond directly to DMX512 slot values. There is a `<propmap_DMx>` element that then explains how to map the required value for the DDL property into DMX512 slots in order to set that value in the device.

For example, a 16-bit pan function on an automated luminaire is controlled by two separate slot values in DMX512 (often called “pan coarse” and “pan fine”) – these give the desired or target position for the pan function. Within the description, this pan target position is described as a single 16-bit virtual property. In order to control pan, a controller therefore needs to set a value into this property. The `propmap_DMx` element then describes how DMX512 slot values must be manipulated in order to load a value into the pan position property – in this case, by splitting the value into two and placing the most significant byte in one slot and the least significant byte in another.

The syntax for describing the functionality of DMX512 devices relies on `<propmap_DMx>` elements. These specify the size of the property value and a mapping that can take any legal value for that property and map it into one or more values or sequences of values in DMX512 slots. This mapping may involve combination with other DDL property values and may involve sequential operations with delays. For example, many devices have configuration operations such as “lamp strike,” which require control or other channels to hold values stable for some delay before an action can occur.

This means that there is one Protocol element representing a DMX512 mapping for each writable DDL property rather than one for each DMX512 slot. In some cases there will be fewer DMX512 Protocol elements, but often there will be more.

2.1 Interlocks and Exclusions

The way multiple functions are packed into few DMX512 slots often means that there are restrictions or exclusions. For example, a DMX512 slot representing intensity may have another meaning while a control

channel is set to some special value. These restrictions must be expressed in the DDL in a generic way (e.g. using driver/driven relationships), as there is no mechanism for doing so within the DMX512 packing syntax.

There are already many mechanisms to do this as DDL devices frequently have interlocks or exclusions on control functions for physical reasons as well as protocol restrictions.

2.2 Examples

A pan axis is described in DDL in the same ways as it would be for any other protocol and is driven by a single 16-bit writable property representing desired position. In order to set this property, the sixteen-bit value is split in two and inserted into two adjacent DMX512 slots.

A lamp strike function is represented in DDL with a single bit value representing desired condition (on/off). In order to set this boolean value, the controller is required to set the value of a control slot to 30%, then take the value of the slot that normally represents intensity from 0 to 100%, wait 1 second, then restore the intensity slot to 0. To clear the boolean value, the controller must repeat the process, but with the control channel at 25%.

3 DMX512 Slot Addressing

Within a device, a DMX512 slot is specified by its fixed position (the *slot offset*) within a block of slots. This block is in turn identified by its *base address*. In many common devices, the base address corresponds to the DMX512 “Start address” configured on the front panel, and each slot used is then a fixed offset from that address.

Some devices may have multiple base addresses allowing different functions to be addressed independently. Any number of base addresses are allowed (for example a dimmer pack with an input patch has one base address per dimmer). In other devices, the base address may be fixed (for example in any device that consumes all 512 channels).

Some devices may have multiple DMX512 inputs – for example a physical interface and an E1.31 interface. In this case, a slot address also needs to identify the interface on which it is received.

Some interfaces (for example ESTA E1.31) allow multiple universes to be received on a single interface. In this case the universe must be identified as well.

The interface, the universe identifier (if required), and the base address are all (potentially) configurable items. They cannot be set from within the DMX512 protocol itself, but may be changed by local means or often by other remote methods such as DMP, SNMP, Web based configuration or RDM. These are represented in DDL as property values since this allows the full range of DDL description to be used to describe constraints on their values and how they may be set. It also allows them to be set using alternative access protocols if present.

Each block of slots used in any DMX512 device has an associated property with *baseAddressDMX512* behavior that specifies the base address and where necessary, the interface and universe for that block.

The fixed slot offset is an inherent part of the way a DMX512 device is designed and can be expressed as a literal constant in the DDL.

Slot addressing in this EPI relies on a number of specific behaviors that are defined in the Core DDL behaviorset [acnbaseDDL], [CoreDDL].

4 Syntax for DMX512 Addressing

4.1 Identification of This Protocol

Within device descriptions, protocol elements (and use protocol elements) that use the syntax and rules defined within this specification shall use the Organization Name “ESTA” and the Protocol Name “EPI26”. These names are defined in accordance with ACN EPI16 [ESTA-IDs], and give the combined textual identifier “ESTA.EPI26”.

4.2 Property Map Elements

Each mapping of a DDL property to operations on one or more DMX512 slots shall be specified within a `<propmap_DMX>` element. This shall be a child of a `<protocol>` element with the protocol attribute “ESTA.EPI26”.

The size of each DDL property that can be accessed via DMX512 shall be specified using a *size* attribute on the `propmap_DMX` element. This is the size of the DDL “virtual” property and is given in *bits*.

```

<!ELEMENT protocol (propmap_DMX) >
<!ATTLIST protocol
  name NMTOKEN #FIXED "ESTA.EPI26"
>

<!ELEMENT propmap_DMX (#PCDATA)>
<!ATTLIST propmap_DMX
  size CDATA #REQUIRED
>

```

The content of this element describes the necessary mapping using a simple scripting language consisting of a list of operations.

The script describes the logical operations, both combinatorial and sequential, required to “write” the value required for the DDL property. Within the script the values of other properties within the description can be referenced by their property ids in a manner similar to variables.

In ABNF [ABNF] the content of propmap_DMX shall match the statements production:

```

statements = 1*( *S statement *S ) ;a list of statements

S          = 1*( CR / LF / TAB / SPACE ) ;Same as XML
TAB        = %x09
LF         = %x0A
CR         = %x0D
SPACE     = %x20

```

Wherever numbers are required they may be hexadecimal, decimal or binary:

```

number     = hexnum / decnum / binnum
hexnum     = "0x" 1* hexdigit
decnum     = 1* decdigit
binnum     = "0b" 1* bindigit
bindigit   = "0" / "1"
decdigit   = bindigit
           / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
hexdigit   = decdigit
           / "a" / "b" / "c" / "d" / "e" / "f"
           / "A" / "B" / "C" / "D" / "E" / "F"

```

4.3 Statements

A statement is a procedure statement, a conditional statement, or a compound statement

```
statement = procedure / conditional / compound
```

4.3.1 Compound Statements

A compound statement consists of a sequence of statements enclosed in curly braces { }.

```
compound = "{" statements "}"
```

4.3.2 Conditional Statements

A conditional statement takes the form:

```
conditional = ifstatement [ elifstatement ] [ elsestatement ]
ifstatement = "if" S test S statement
elifstatement = "elif" S test S statement
elsestatement = "else" S statement
test = "(" boolexpr ")"
boolexpr = expression ; evaluates to a boolean
```

Test is an expression. If the value is non-zero, the associated statement is executed. If zero, it is not.

As with the C language, where conditionals are nested, any else or elif statement is defined to be associated with the innermost (most recent) conditional. In most cases ambiguity should be avoided and compound statements should be used for clarification. For example, in the statement:

```
if (A) if (B) setslot(dmxAddr, 5, 1) else setslot(dmxAddr, 5, 0)
```

The else statement is associated with the test (B) so if the test (A) evaluates to zero then no assignment is made.

4.4 Procedures

A number of fixed procedure statements are available including the all important setslot(). A procedure keyword always has an argument list (may be empty) in brackets.

```
procedure = next / wait / setslot
```

4.4.1 Setslot()

The setslot procedure describes how to pack a value (based on virtual property values) into a DMX512 slot.

```
setslot = "setslot" [ indexnames ]
          "(" baseref "," offsetexpr "," valueexpr ")"
```

For example:

```
setslot(dmxAddr, 5, #);
```

4.4.1.1 Array indexes

When a property is within an array, the DMX512 addressing script is always interpreted as accessing a specific array element and must be applicable to any chosen element of the array. This is achieved by introducing a named variable whose value takes the index of the specific property being written when evaluating any of the expressions in the procedure. For each dimension of the array, the setslot procedure shall declare a symbolic index name.

```
indexnames      = "[" *S indexname *S "]" [ indexnames ]
indexname       = name          ; declares a name for an array index variable
```

Index names are ordered with the nearest to the root of the property tree (outermost) first and the nearest to the identified property (innermost) last.

For example, in a processor controlling an array of 4 racks, each containing 18 dimmers, access to individual dimmer level properties might be written thus:

```
setslot[rack][dimmer](dmxAddr, 4 + rack * 18 + dimmer, #);
```

The index names can be used in any of the expressions in arguments to the setslot procedure. The scope of each name is the arguments of the setslot procedure on which they are declared.

4.4.1.2 Base Address

A slot address is based on a reference to a base-address property. This shall be the name of a property that has baseAddressDMX512 behavior [acnbaseDDL]. The reference shall obey the default scoping rules of the DDL specification for element identifiers and references.

```
baseref         = *S propertyref *S
                 ;N is number of dimensions in array propertyname
propertyref     = propertyname N(arrayindex)
propertyname    = name          ; shall match the xml:id of a property (IDREF)
arrayindex      = "[" indexexpr "]"
indexexpr       = expression    ; evaluates to an integer
                 ; within the bounds of the array
```

4.4.1.2.1 Propertyref to an Array

Note that any propertyref, including baseref, that identifies an array property shall have as many arrayindex specifiers as there are dimensions in the array. Array indexes are ordered with the nearest to the root of the property tree (outermost) first and the nearest to the identified property (innermost) last.

4.4.1.2.2 Multiple Universes or Interfaces

In any device where multiple interfaces and/or universes of DMX512 may be present, these shall be identified as a part of the base address property declaration. See baseAddressDMX512 behavior specification for details.

4.4.1.3 Slot offset

The baseref argument identifies a DMX512 base address. The offset expression evaluates to an integer and is added to the base address to identify a specific DMX512 slot. As per slotAddressDMX512 behavior [acnbaseDDL] it is an error if the combination of baseref and offsetexpr falls outside the range 1..512.

```
offsetspec      = "[" offsetexpr "]"
offsetexpr      = expression    ; an offset from the baseaddress
```

4.4.1.4 Slot value

The value to put into the DMX512 slot is evaluated by an expression that must be in the range 0..255.

```
valueexpr      = expression
```

4.4.1.4.1 Sizes, truncation and padding

The size *in bits* of each DDL property is found in the size attribute on the propmap_DMx element.

Where the calculated unsigned value is greater than 255 (the maximum value that can be stored in the 8-bit slot), then the slot value is the least significant 8 bits of the calculated value given. There is no need for explicit masking.

Where values to go in a slot evaluate to 127 or less, the most significant bits must be set to zero when copied to the slot.

4.4.2 next()

The next procedure indicates that at least one DMX512 packet shall be output before proceeding.

```
next          = "next (" *S ") "
```

For example, the next() procedure below ensures the the value 99 is transmitted in at least one DMX512 packet before the value 0 is transmitted in the same slot. Without it, the output would be ambiguous and only one value (0 or 99) would actually be transmitted. See 4.4.4 Order of Execution.

```
setslot(DMXaddr, 2, 99)
next( )
setslot(DMXaddr, 2, 0)
```

4.4.3 wait()

It is not uncommon in DMX512 devices to find configuration and control sequences that rely on slots holding a value for a certain length of time. The wait function allows such sequences to be easily encoded.

```
wait          = "wait(" timeexpr ") "
timeexpr      = expression      ; evaluates to a time in milliseconds
```

For example, an automated fixture has a recalibrate command activated by setting a control channel (at offset 0) to a nominal 60% value for between 3 and 5 seconds and then returning it to zero. In DDL the recalibrate property would be a trigger property that would be set to force recalibration. Its propmapDMX could be described:

```
setslot(DMXadd, 0, 255 * 6 / 10)
wait(3200)
setslot(DMXadd, 0, 0)
```

4.4.3.1 Interpretation of wait time

The value of the time expression in a wait statement is given in milliseconds. However, the timing of DMX512 packets allows nothing like 1 millisecond resolution. The start of the wait time shall be delayed until at least one packet has been transmitted (e.g. the result of preceding setslot operations). The actual transmission of any subsequent operations may then be further delayed because they must wait until the next packet after the expiry of the wait time (which may be up to a second later!).

This logic means that wait(0) is exactly equivalent to next().

4.4.4 Order of Execution

The procedures may be interpreted in any order that does not change the sequence of transmitted data.

In practice this means that consecutive procedures of the same name may be reordered, but that procedures of different names may not be reordered. For example:

```
setslot(dmxAddr, 5, # >> 8);
setslot(dmxAddr, 6, #);
```

This is saying that the high order byte of the desired value must go in slot 5 and the low order byte in slot 6. It does not imply any sequence since both slots will be transmitted together in the same packet in slot order.

4.5 Expressions

Expressions are composed of variables, literals and operators. Brackets () may be used to impose priority. Because variable names may contain certain operator characters, they shall be separated from operators by whitespace.

```

expression      = unaryexpr / binaryexpr / subexpr
unaryexpr       = unaryop rterm
binaryexpr      = lterm binaryop rterm
subexpr         = *S "(" mterm ")" *S
rterm           = ( *S literal *S ) / ( S variable *S ) / expression
lterm           = ( *S literal *S ) / ( *S variable S ) / expression
mterm           = ( *S literal *S ) / ( *S variable *S ) / expression

```

4.5.1 Variables

Variables are either property values, or array index names.

```

variable        = propertyval / indexname
propertyval     = propertyref / "#"
propertyref     ; defined above
indexname       ; defined above

```

Property values are referenced by name using their id attributes. The reference shall obey the default scoping rules of the DDL specification for element identifiers and references. The value of the variable is the value of the named property.

See 4.4.1.1 Array indexes4.4.1.1 Array indexes for explanation of indexname.

The special name “#” (U+0023, crosshatch) refers to the value of the property containing the expression (the property whose access is being described). It shall never be indexed because the expression containing it is always evaluated with reference to an individual element of the array.

4.5.2 Literals

Literals are numeric constants.

```

literal        = number

```

4.5.3 Operators

Operators have the same meaning and priority as in the C language.

```

unaryop      = "~" / "!" / "+" / "-"
binaryop     = "*" / "/" / "%"           ; multiplicative
              / "+" / "-"             ; additive
              / "<<" / ">>"           ; shift
              / "<" / ">" / "<=" / ">=" ; relational
              / "==" / "!="          ; equality/inequality
              / "&" / "|"            ; bitwise
              / "&&" / "||"          ; logical condition

```

4.6 Examples

Throughout these examples, the following UUIDname is assumed:

```

<UUIDname
  UUID="71576eac-e94a-11dc-b664-0017316c497d"
  name="acnbase.bset"/>

```

Firstly a DMX512 interface property is declared – this represents a single DMX512 universe. The first child property declares this to be an input interface and the second is a DMX512 address property with id="DMXADDR":

```

<property valuetype="NULL" id="DMXinterface">
  <label>The DMX512 Input Port</label>
  <behavior set="acnbase.bset" name="netDMX512-XLRpri"/>
  <property valuetype="immediate">
    <behavior set="acnbase2.bset" name="netInterfaceDirection"/>
    <value type="uint">0</value>
  </property>
  <property valuetype="network" id="DMXADDR">
    <behavior set="acnbase.bset" name="baseAddressDMX512"/>
    <behavior set="acnbase.bset" name="persistent"/>
    <behavior set="acnbase.bset" name="volatile"/>
    <protocol protocol="ESTA.DMP"/>
    <propref_DMP loc="x" read="true" size="2" write="true"/>
  </protocol>
  </property>
</property>

```

Now we will declare a one byte input example that is accessed at DMX512 address Start + 3. The “#” refers to the desired value of the current DDL property. This value is simply placed into the array at the base address plus three:

```
<property valuetype="network">
  <label>One byte input example</label>
  <behavior .../>
  <protocol protocol="ESTA.EPI26">
    ...
    <propmap_DMX size="8">
      setslot(DMXADDR, 3, #)
    </propmap_DMX>
  </protocol>
</property>
```

Now we will declare a single 16-bit property that is split between two DMX512 slots at offset 6 (high byte) and 7 (low byte):

```
<property valuetype="network">
  <label>16-bit input example</label>
  <behavior .../>
  <protocol protocol="ESTA.EPI26">
    ...
    <propmap_DMX size="16">
      setslot(DMXADDR, 6, # >> 8)
      setslot(DMXADDR, 7, #)
    </propmap_DMX>
  </protocol>
</property>
```

In this example, two input properties of four bits each, are packed into a single byte value at address offset 1. Note that the requirement to pack additional bits with zeros means that masking is not required:

```
<property valuetype="network" id="inputA">
  <label>High nibble input</label>
  <behavior .../>
  <protocol protocol="ESTA.EPI26">
    <propmap_DMX size="4">
      setslot(DMXADDR, 1, (# << 4) | inputB)
    </propmap_DMX>
  </protocol>
</property>
...
<property valuetype="network" id="inputB">
  <label>Low nibble input</label>
  <behavior .../>
  <protocol protocol="ESTA.EPI26">
    <propmap_DMX size="4">
      setslot(DMXADDR, 1, (inputA << 4) | #)
    </propmap_DMX>
  </protocol>
</property>
```

This example implements the lamp strike function of section 2.2 Examples2.2 Examples, where the “control” slot is at offset 15 and the intensity is at offset 0:

```
<property valuetype="network" id="inputA">
  <label>Lamp on/off</label>
  <behavior nema="type.boolean"/>
  <protocol protocol="ESTA.EPI26">
    <propmap_DMX size="1">
      if (#) {
        setslot(DMXADDR, 15, 77)
      } else {
        setslot(DMXADDR, 15, 64)
      }
      setslot(DMXADDR, 0, 0)
      next()
      setslot(DMXADDR, 0, 255)
      wait(1000)
      setslot(DMXADDR, 0, 0)
      next()
      setslot(DMXADDR, 15, 77)
    </propmap_DMX>
  </protocol>
</property>
```

Now an array of 10 two-byte properties are declared each of which can be patched to an arbitrary place in the DMX512 input. First we declare the array of 10 DMX512 addresses:

```
<property valuetype="NULL" id="DMXinterface">
  <label>The DMX512 Input Port</label>
  <behavior set="acnbase.bset" name="netDMX512-XLRpri"/>
  <property valuetype="immediate">
    <behavior set="acnbase.bset" name="netInterfaceDirection"/>
    <value type="uint">0</value>
  </property>
  <property valuetype="network" id="INPATCH" array="10">
    <behavior set="acnbase.bset" name="baseAddressDMX512"/>
    <behavior set="acnbase.bset" name="persistent"/>
    <behavior set="acnbase.bset" name="volatile"/>
    <protocol protocol="ESTA.DMP"/>
    <propref_DMP loc="x" read="true" size="2" write="true"/>
  </property>
</property>
```

Because the DMX512 address is an array of addresses, when it is used, it needs a subscript. Because the property we are now declaring is also an array, an indexname *i* is declared each time setslot is used:

```
<property valuetype="network" id="inputA" array=10>
  <label>2-byte patchable array example</label>
  <behavior .../>
  <protocol protocol="ESTA.EPI26">
    <propmap_DMx size="16">
      setslot[i](INPATCH[i], 0, # >> 8)
      setslot[i](INPATCH[i], 1, #)
    </propmap_DMx>
  </protocol>
</property>
```

Annex A. Normative References

- [ACN] ESTA <http://tsp.esta.org> E1.17. Entertainment Technology - Architecture for Control Networks. The edition current when this document becomes approved.
- [Arch] ESTA <http://tsp.esta.org> E1.17. Entertainment Technology – Architecture for Control Networks. “ACN” Architecture. The edition current when this document becomes approved.
- [DDL] ESTA <http://tsp.esta.org> E1.17. Entertainment Technology – Architecture for Control Networks. Device Description Language. The edition current when this document becomes approved.
- [CoreDDL] ESTA <http://tsp.esta.org> E1.17. Entertainment Technology – Architecture for Control Networks. EPI 22 DDL Core Modules for ACN Devices. Draft Standard. The edition current when this document becomes approved.
- [acnbaseDDL] ESTA <http://tsp.esta.org> *urn:uuid:71576eace94a-11dc-b664-0017316c497d*. ACN Base Behavior Set. The edition current when this document becomes approved.
- [ESTA-IDs] ESTA <http://tsp.esta.org> E1.17. Entertainment Technology – Architecture for Control Networks. EPI-16 ESTA Registered Names and Identifiers – Format and Procedure for Registration. The edition current when this document becomes approved.
- [DMX512] [DMX512-A] ESTA <http://tsp.esta.org> ANSI E1.11. Entertainment Technology - USITT DMX512-A, Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories. The edition current when this document becomes approved.
- [DMX512/1990] United States institute for Theatre Technology Inc. <http://www.usitt.org> DMX512/1990 - Digital Data Transmission Standard for Dimmers and Controllers. 1990. Superseded by ANSI E1.11-2004 (DMX512-A).
- [DMX512-1986] United States institute for Theatre Technology Inc. <http://www.usitt.org> DMX512 –Digital Data Transmission Standard for Dimmers and Controllers. 1986. Superseded by DMX512/1990.
- [E1.31] ESTA <http://tsp.esta.org> ANSI E1.31-2009, Entertainment Technology – Lightweight streaming protocol for transport of DMX512 using ACN. 2009.
- [ABNF] Internet Engineering Task Force (IETF) <http://ietf.org> RFC 2234 <http://ietf.org/rfc/rfc2234.txt> D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. November 1997.