ANSI E1.17-2015 (R2020), Architecture for Control Networks
Device Description Language (DDL)

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.

TSP document CP/2009-1024r1

Revision History	
R5-draft-435	2009-07-22
R5pre3	06/17/09
R5pre2	2009-03
R5pre1	2008-03-14
R4	2006-03-14
R4pre2	2006-03-08
R4pre1	2006-03-03
R3	2005-10-14
R3pre2	2005-05-20
R3pre1	2005-04-25
R2	2004-11-09
R1	2004-10-07
Pre1	2003

Abstract

DDL is a language for describing devices controllable by getting and setting of properties (this can apply to almost anything!). this Standard describes a generalized control model for such devices. It specifies an XML based syntax for declaring the model for particular devices and defines how to use the language in conjunction with specific control protocols.

Table of Contents

1 Introduction

Device Description Language is a language for describing controllable devices in a way that is machine readable and enables controllers to automate the process of interfacing to them.

1.1 Origin and Goals

Device Description Language (DDL) was developed as a an adjunct to the Device Management Protocol [DMP] that is a part of the E1.17 Network suite developed by the Entertainment Services and Technology Association (ESTA) [ACN].

The design goals for DDL are:

1. DDL should describe devices rather than dictate their functionality and modality.
2. The designer of controlled equipment must be able to describe their device and all its features in as much depth of detail as possible.
3. Any process interpreting a description need not “understand” all of it but must be able to access and handle as much or as little as its designer chooses.
4. DDL must keep the human centric view of what equipment does separate from the algorithmic view of how to control it.
5. DDL should provide a generic language capable of describing devices not yet invented.
6. A clear path for extensibility should be provided.
7. Device descriptions should be static so they can be passed around with or without the equipment they describe and so that identical devices have identical descriptions.
8. DDL should leverage whatever technologies are already available.
9. The burden of policing conformance and providing extensions to the specification should be minimized.

A consequence of 2 and 3 is that interoperability between controller and controlled equipment should reach the highest level of control achievable by the less capable of the two. It is important to bear in mind when reading this specification that controllers are not required to support or act on all the information contained within a device description. Simple controllers will look for a few basic elements while complex controllers will go to greater depth.

1.2 Modularity

To encourage re-use and commonality of descriptions and to extend its flexibility, DDL is a modular language. This means that a single piece of equipment may be described by multiple linked description modules and allows modules describing common functional blocks to be re-used directly in other equipment.

1.3 Parts of DDL

There are three major parts to DDL:

1. The device model – an abstract model used for describing devices.
2. The XML format for DDL – a syntax for rendering device descriptions in a machine and human readable serial format.
3. The Protocol interface – how device descriptions marry up to a particular protocol.

2 Scope Purpose and Application

2.1 Scope

This standard defines a generalized description language for devices that may be controlled remotely via a

data link or networking protocol. DDL does not, for example, describe controller functions although control equipment frequently contains parts that are remotely controllable or monitorable and so some aspects of that equipment might have DDL descriptions.

2.2 Purpose

DDL has been developed together with a protocol suite aimed at control of entertainment technology equipment – motion control, lighting and sound level control, timed sequences etc. – The ESTA ACN Protocol Suite [ACN]. The DDL standard was developed to facilitate a high degree of automation in the configuration of control networks, and in particular in the adaptation of controllers to the controllable devices that are discovered in an arbitrary network.

2.3 Application

While DDL was developed as part of the ACN protocol suite, it has potential application in many control networks. It is directly suited to devices that may be controlled by reading and writing to registers or locations within the device using few message types. Protocols in which devices are controlled by complex messages to few destinations may require some transformation to fit with the DDL model.

3 The Device Model

3.1 Device Model Introduction

The device model provides the underlying structure to DDL. It provides a way to think about how devices are broken down step by step into their fundamental control elements. Having analyzed the device in this way, the hierarchy is laid out in a file or files in a text format (the syntax).

DDL is a hierarchical modular language and in DDL the term device means the piece of equipment described by a single module including all the modules contained within it. So if multiple modules (each describing a device) are combined into a larger module, then that too describes a device.

3.2 Descriptions, Classes, Instances and Identifiers

Each device description describes a class of devices. Any physical device that conforms to that description is then called an instance of that class. DDL uses a UUID [UUID] to identify device classes and by extension their DDL descriptions. Any UUID that identifies a device class is called a Device Class Identifier or DCID.

3.3 Modules, Devices, and Appliances

DDL devices are modules that may describe an entire piece of equipment or may just describe a part of it. DDL allows any device to contain sub-devices (also called child devices) and there is no distinction between devices and sub-devices in the way that DDL describes them – any device may be included as a child device of some parent.

Any instance of a device that has no parent device is called a root device. The entire device structure formed by a root device and all its children and descendants is called an appliance. An appliance typically corresponds to a single piece of equipment and is frequently a single network entity although this is dependent on the protocol used. (In ACN systems using DMP an appliance corresponds to a component that exposes properties. Confusingly, DMP calls this a device (see: Relationship Between DDL Devices and DMP Components)

While a DDL device is a module, DDL syntax also defines other modules that are not devices (see: DDL Modules – Devices, Behaviorsets and Languagesets).

3.4 Hierarchy of Properties

3.4.1 Modified Tree Structure

3.4.1.1 General

The hierarchy of properties representing a device in DDL forms a set of modified tree structures. In the DDL device model each node of the tree is called a property and one or more trees of properties represents the structure of a device (the model is a set rather than a single tree, because there can be multiple root-level properties in a device).

A tree is a structure very familiar in computing and consists of a set of nodes connected such that every node is the child of a parent node. The exception is a single node in each tree called the root node that has no parent itself but is the parent, or more distant ancestor of all other nodes within the tree. Any node may have any number (including zero) of child nodes and a node with no children is called a leaf node.

3.4.1.2 Branch Re-combination and Shared Properties

The structure of DDL is modified because separate branches of the tree may re-combine – in which case the node at a re-combination point has multiple parents. The re-combination is restricted in DDL: it is a requirement that there shall be no property that is both ancestor and descendent of any other property or of itself. In mathematical terms this is a directed acyclic graph or DAG [http://en.wikipedia.org/wiki/Directed_acyclic_graph].

For example, consider an automated lighting fixture with an initialization operation that is performed by setting the Boolean value of an initializer property. Structurally the initializer is a sub property of the property representing the function that it initializes (e.g. an initializer for the pan function would be a sub property of the pan property). But it is common practice in real devices for a single initializer to cause initialization of multiple functions of the device (one initializer for both pan and tilt). This means that the one initializer is shared between pan and tilt. In DDL this situation is allowed – the initializer property has both pan and tilt as its parents so the pan and tilt branches of the tree re-combine at the initializer property.

In the XML syntax defined below, there is no direct way to represent re-combination of the branches of the tree, so this is achieved by declaring the same property within each of its parents and indicating that it is shared. The shared property is fully declared in one place only – this is the defining declaration – and in all other places that the same property needs to appear, a simple reference to the defining declaration is used – this is called a propertypointer (see: Shared Properties).

3.4.2 Property Values

3.4.2.1 General

As well as zero or more child properties (also called sub properties or properties of properties) each property shall have zero or one values.

A property value is a single item that may be a number, a text string or sometimes a more specialized object such as an Ethernet MAC address. Typically the value may be read or written via the network, although it may also be implied or provided within the DDL itself.

3.4.2.2 Complex Values Harm Interoperability

Any structure within a property value is not visible to the device model. Creation of devices with properties whose values have complex structure is strongly discouraged – the philosophy of creating devices that can be described using a common language like DDL conflicts with attempts to bury complex structure and behavior within the values themselves and the chances of a generic controller being able to handle such values is greatly reduced.

3.4.3 Examples

For example consider a device that is a remote controlled robotic camera. This might have a property for exposure control, one for an X,Y,Z positioner and one for the pan-tilt-rotate direction:

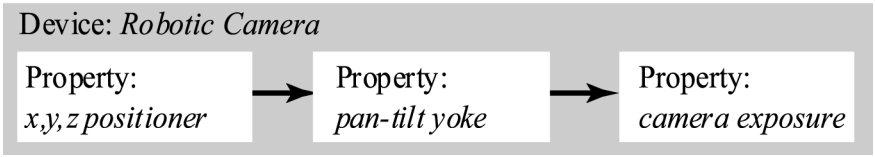
XML and DMP examples

Snippets of XML description are included in examples throughout this section of this Standard. These examples are often simplified for readability and may not conform strictly to the DDL specification.

An ellipsis, "..." is used in many examples to indicate which portions not relevant to the example have been omitted for brevity. This is not part of the XML syntax.

Examples also use ESTA DMP [DMP] extensively. The reader should be aware that DDL can be used with other protocols and in fact that the device model is not tightly coupled to XML and could be represented using other syntax (although the current project has no plans to do so).

Figure 1:Robotic Camera Property Layout



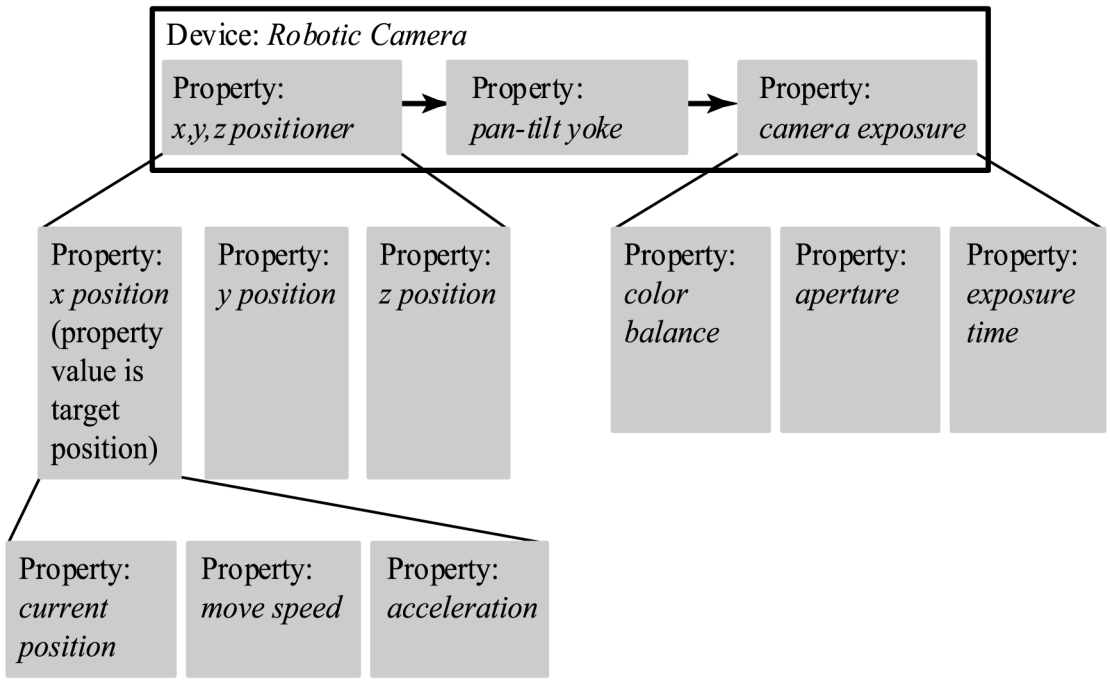
Example 1: Robotic Camera Sample DDL

```
<!-- description of a robotic camera -->
<property ...>
  <label>Robotic Camera</label>...
  <property ...>
    <label>x,y,z positioner</label>...
  </property>
  <property ...>
    <label>pan-tilt yoke</label>...
  </property>

  <property ...>
    <label>exposure control</label>...
  </property>
</property>
```

In their turn X, Y and Z are separate child properties of the positioner that have values representing the position for X, Y and Z. Each of these motor driven axes in turn might have a child property representing desired or target position, maximum speed, acceleration or other parameters used to control movement. The speed property would probably have child properties that define its upper and lower limits. Other properties of X, Y and Z might be the units they operate in (meters, millimeters etc.). The exposure control meanwhile could have a property for color balance and one for exposure time/aperture.

Figure 2:Robotic Camera Extended Property Layout



Example 2: Robotic Camera Extended Sample DDL

```
<DDL>
  <device>
    <!-- description of a robotic camera -->
    <property ...>
      <label>Robotic Camera</label>...
      <property ...>
        <label>x,y,z positioner</label>...
        <property ...>
          <label>X position</label>...
          <protocol name="example">...</protocol>
          <property ...>
            <label>target position</label>...
            <protocol name="example">...</protocol>
          </property>
          <property ...>
            <label>speed limit</label>...
            <protocol name="example">...</protocol>
          </property>
          <property ...>
            <label>acceleration</label>...
            <protocol name="example">...</protocol>
          </property>
        </property>
        <property ...>
          <label>Y position</label>...
          <!-- expands similarly to X axis -->
        </property>
        <property ...>
```

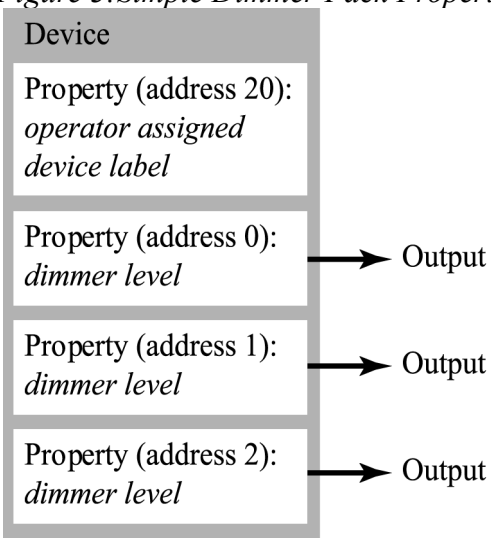
```
          <label>Z position</label>...
          <!-- expands similarly to X axis -->
        </property>
      </property>
      <property ...>
        <label>pan-tilt yoke</label>...
      </property>
      <property ...>
        <label>exposure control</label>...
        <property ...>
          <label>color balance</label>...
        </property>
        <property ...>
          <label>aperture</label>...
        </property>
        <property ...>
          <label>exposure time</label>...
        </property>
      </property>
    </property>
  </device>
</DDL>
```

At the higher levels in the tree hierarchy properties represent major functional blocks of the device (such as the X,Y,Z positioner in the example above) and at the very top of each tree there is always just one root property (logical trees are usually viewed upside down with the root at the top), in this case representing the camera itself (a DDL device may however contain more than one tree). At the highest levels properties often do not have a value but are simply containers for other properties. As the hierarchy descends properties may represent smaller items (e.g. the desired X position) and at the lowest level they are points of detail such as the upper limit on the speed setting for a motion control (which would probably be a fixed value present for informational purposes).

The control structure of the device is represented by the structure and attributes of properties and the behavior they provide. The state of the device is represented by the values associated with the properties. To control or operate the device, the values of its properties must be changed.

Figure 3: shows an example of a very simple three-way dimmer pack. There is just one property for each dimmer that represents the dimmer output level. There is also an additional property that is a non-volatile string that the operator may use to store a label for this particular pack. The label attribute is a simple text label that could be presented to the user to tell them what purpose the property serves.

Figure 3:Simple Dimmer Pack Property Layout



Example 3: Simple Dimmer Pack Sample DDL

```
<DDL>
  <device>
    <property ...><label>Dimmer pack</label>...
      <property ...>
        <label>Device supervisory stuff</label>...
        ...
        <property ...>
          <label>Device Label</label>...
          <protocol name="example">
            <address persistent="true">20</address>
          </protocol>
        </property>
      </property>
      <property ...>
        <label>Dimmer 1</label>...
        <property ...>
          <label>Dimmer 1 Intensity</label>...
          <protocol name="example"><address>0</address></protocol>
        </property>
      </property>
      <property ...>
        <label>Dimmer 2</label>...
        <property ...>
          <label>Dimmer 2 Intensity</label>...
          <protocol name="example"><address>1</address></protocol>
        </property>
      </property>
      <property ...>
        <label>Dimmer 3</label>...
        <property ...>
```

```
          <label>Dimmer 3 Intensity</label>...
          <protocol name="example"><address>2</address></protocol>
        </property>
      </property>
    </property>
  </device>
</DDL>
```

3.4.4 Meaning of Property Values – Driven Properties

For many device features, the simplest control model, would appear to be to represent the state of that feature by a writable property such that when that property is changed, the software and hardware within the device adjust the state of the feature to match. However, in real devices there are almost invariably times when the state of the feature does not match the value written to the property. This can occur during transient states while the device is changing the feature – for example with a motor driven feature it can take significant time to reach a given target position; it occurs when there are values of the property that represent states that the feature cannot take – for example due to range limits; and it occurs when there are other inputs that may change that feature such as local controls or overrides or physical conditions such as obstructions (the scenery hitting the device).

As control models become more sophisticated, it is often the case that the feature is not controlled by a single property value, but by a combination of values – for example, a motor driven feature (such as pan or tilt for a projector or camera) is typically controlled internally using a combination of acceleration up to some speed limit, a period of movement at that speed and a deceleration to rest at the target. The device may expose all four parameters (acceleration, speed, deceleration, target) that all contribute to the dynamic control of the single movement axis.

In all these cases, where the interface is bidirectional, the designer may well wish to expose a read-only property that represents the actual state of a feature, in addition to the writable properties used to control it – this presents desirable feedback.

In order to present a consistent structure for modeling devices across this wide range of conditions DDL uses a rule that for any feature, there must be an outermost property that represents the actual state of that feature. This property then contains all the writable properties that are available to control that feature. In a bidirectional network where the device exposes the actual state of the feature, this is the outermost property in the description. However, in cases where there is no actual state value on the network, the description still includes an actual state property but must make it an implied property (see below) – indicating that the device model is identical but the actual state is not available.

This principle is extended within descriptions to the general case that one property may be driven by one or more driver properties. In which case, the driven property is always the outermost (closest to the root of the description tree) and the driver properties are child (or descendent) properties of it. As well as the case of driven actual values that differ dynamically from their target driver properties, driven properties are used extensively in DDL to represent cases where multiple properties are used in combination to achieve a result. Examples include cases where multiple network inputs contribute to some algorithmic process to achieve a result (e.g. “Highest Takes Precedence” combination common in lighting controls) and also selectors where a predefined restricted set of values are available and a property selects between them (e.g., an input voltage selector that can be set to select 100V, 115V or 230V).

3.4.4.1 examples

A property represents the desired or target position of a winch. When it is set to a new value, the winch controller must ramp up the motor speed, move for some time at a given speed and then ramp down the speed to come to rest at the desired position. While this is happening, the state of the winch is not that set in

the property but is changing and a property representing the actual winch position would constantly change to represent this. This latter property representing the actual position could not be writable.

A lighting dimmer has 8 inputs each of equal weight, which are combined on a “highest takes precedence” basis (the dimmer adopts the maximum value out of the inputs). In this case, there is not just one target property but several drivers and their parent driven property reflects the highest value of all the drivers.

A manual slider is represented as a single property value that reflects the state of the slider. Because this feature is an input only and there is no way to control it, the actual value property contains no drivers at all.

3.4.4.2 Implied Properties

In some cases the value of a driven property may not be available – this can occur simply because the data link is unidirectional or because the device designer cannot or chooses not to expose it. For example, in an analog servo driven system the controller itself may have no feedback of actual position. In these cases the property is neither writable nor readable – this is called an implied property. Implied properties are present in the description so that the description structure is consistent, but is not present on the network. In some cases the value of an implied property may be inferred from the properties driving it. Implied properties must however be declared in the DDL where relevant as this is how DDL builds a consistent description of the device. In devices with a bidirectional interface, implementers should not use implied properties if a network readable value could be provided.

3.4.4.3 Unattainable targets

If a target state is set (by writing a value to a property) that is a legal operation within the protocol, yet that breaks constraints on that property (e.g. a value that exceeds a limit, or that would require movement at an unachievable speed), the device has two possible courses of action. It may reject the command and maintain its previous state, or it may do the best it can by adopting an attainable target that is close to that requested. DDL imposes no requirement on which choice is taken. However, access protocols (The Access Protocol) may define their own rules or may provide messages to handle this situation. Property behaviors may also be defined that allow control or expression of which option is taken or that show the attainable target that has been adopted.

3.4.5 Order of properties – Chains of Control

There are many devices that can be described as a set of operations that are performed in turn on a stream or channel of material. For example, audio channels are routinely subjected to a distinct sequence of operations (input gain, frequency filtering, effects, output gain etc.). Another example is an automated light where a sequence of operations are performed on a light beam (intensity, shutters, gobo, color filter, focus, effects etc.).

In cases where order of processing in such a channel would have an effect on the final outcome or on the value of intermediate properties (e.g. audio metering points), these properties shall be declared in the order in which they are applied to the channel. For more detailed analysis of these situations see stream behaviors in the ACN base behaviorset [acnbaseDDL].

3.5 Hierarchical Device Structure

There are many cases where describing an entire piece of equipment as a single device is not sensible or convenient. This occurs both because common items may repeat in many places (reuse of descriptions is beneficial both in saved space and in allowing controllers to recognise common structures) and because much equipment is modular and parts may be changed, either statically as when different options are present, or dynamically as when different modules are fitted or different configurations selected.

As seen in Modules, Devices, and Appliances any DDL device may contain child devices and may itself be

contained within a parent. The partitioning of appliances into devices is at the choice of the author of the descriptions and DDL devices do not necessarily bear a close relationship to physically identifiable modules of the equipment (although they frequently do so).

Sub devices can occur anywhere within the property tree of a device. Each root property and associated property tree of the sub-device is then considered to be a branch of the parent device tree in exactly the same order as they occur in the sub device. Mechanisms that allow the sub device type to change or be re-specified are very powerful in describing equipment that may change as modules are added or removed or as a result of reconfiguration.

The syntax mechanisms for declaring sub devices are specified in the syntax section.

3.6 The Access Protocol

The device model represents the state of a device by the values of its properties. The network provides the means to access and modify the values of those properties and the access protocol is the link between the control or device application and the network. (In the current implementation of DDL ESTA-DMP [DMP] is the only access protocol defined).

The value of a property is accessed by reading it (or writing where appropriate) via the access protocol. The device description needs to give enough information for a controller to know where and how the value is accessed. This information, which is highly dependent on the access protocol used, is contained in a protocol reference that is part of the property declaration within the description. The protocol reference shall identify the access protocol and then all the protocol-dependent information needed to access the value using that protocol (e.g. the address or message type, the data size and so on).

For any access protocol to be used with DDL an interface must be defined that specifies how the device model is interfaced to the access protocol, how this is represented in the syntax and any extensions needed to make DDL work with it. The requirements of this interface are described in Interface to the Access Protocol.

3.6.1 Equipment Supporting Multiple Access Protocols

When a piece of equipment supports multiple access protocols, the representation of that equipment from a protocol view-point may or may not allow a single common device model to be used for more than one of the protocols. There are also times when two or more protocols must be used in combination to effect control over the equipment. (e.g. a combination of [RDM] and [DMX512]).

Where a common device model is used for multiple protocols, then each property of the model may contain a separate protocol reference for each protocol by which it is accessible.

Where the protocols are independent and a common device model is not suitable, then separate and independent descriptions must be used.

3.7 Four Locations for Property Value

There are four “kinds” of property that are distinguished by where and how their value, if any, is located.

NULL

We have seen (see: Hierarchy of Properties) that a property need not have a value at all – it may simply have child properties. This is a NULL value property.

Network

This is the common case where the value of a property is accessible (readable and/or writable) via the network (see: The Access Protocol).

Implied

An implied property is one whose presence is implied by the functional features of the device but whose value is not directly accessible and must be inferred. (see: Implied Properties)

Immediate

The fourth kind of property value is an immediate value. This is a value that is constant both over time and across all instances of the device. In this case the value may be provided directly in the DDL and is immediately available when reading the description.

3.7.1 Immediate Value Considerations

Before putting an immediate value into a description, a device designer must ask not only whether the value is the same for all instances of the device, but also whether the description may, in the future, be applied to new devices that may have a different value. For example, putting a product's model name into a description as an immediate value prevents the description being re-used if the model name changes (e.g. for marketing reasons). The availability of template devices and parameters can be used to mitigate this issue, but by using a Protocol reference for the model name marketing or geographical variations will not affect the description – on the other hand, the model name would not then be available to an offline program that had only the description available and no actual hardware.

Either course of action may have the advantage depending on the circumstances.

3.8 Grouping Properties

The relationships of properties representing states of a device cannot be expressed purely in the behavioral descriptions of those properties but are also expressed by the structural relationships of those properties in the description. Properties representing actual states of a device shall be grouped together in a way that directly models the logical way in which the control entities they describe relate within the device.

In the previous dimmer example each property is independent of the others and so each appears in its own group. If each channel of the device included a dimmer and a color changer, then these would appear together in the same group:

Example 4: Grouping Sample DDL

```
<DDL>
  <device>
    <property ...>
      <label>colored lights</label>...
      <property ...>
        <label>Luminaire 1</label>...
        <property ...>
          <label>Intensity</label>...
          <protocol name="example"><location>101</location></protocol>
        </property>
        <property ...>
          <label>Color</label>...
          <protocol name="example"><location>102</location></protocol>
        </property>
      </property>
      <property ...>
        <label>Luminaire 2</label>...
        <property ...>
```

```
          <label>Intensity</label>...
          <protocol name="example"><location>201</location></protocol>
        </property>
        <property ...>
          <label>Color</label>...
          <protocol name="example"><location>202</location></protocol>
        </property>
      </property>
      <property ...>
        <label>Luminaire 3</label>...
        <property ...>
          <label>Intensity</label>...
          <protocol name="example"><location>301</location></protocol>
        </property>
        <property ...>
          <label>Color</label>...
          <protocol name="example"><location>302</location></protocol>
        </property>
      </property>
    </property>
  </device>
</DDL>
```

3.9 Property Behaviors

Simply listing the properties of a device – even in a structured way – does not provide the algorithmic information on behavior that a controller might need.

For example, consider a property representing the “theta” axis in a polar coordinate positioning system. Two (sub) properties of the theta axis are its scale and units (e.g. “0.1” and “degree”) others are its minimum and maximum values (e.g. “-200” and “+200”). The meaning of each property is expressed as one or more behaviors. All properties in DDL must have one or more associated behaviors to indicate their use. Each behavior expresses some aspect of what that property means and how it should be used.

In practical terms, each behavior attached to a property is simply a reference (by name) to a separate behavior definition that provides a full description and specification for any property with the named behavior.

3.9.1 Multiple Behaviors of a Property

A property may have multiple behaviors. This is required because the semantics of many behaviors are independent of each other. For example, a property may declare both “minimum limit” and a “floating point” behavior.

In some cases, multiple behaviors may be applied because they relate to slightly differing device models or to different access protocols. Implementers of DDL consuming applications (controllers in the widest sense) should be aware of this since it means that a property may carry behaviors that are not relevant to the scope of a particular application.

3.9.2 Behavior Definitions

A behavior definition describes in precise terms what any property that references it represents, provides a specification of how it must behave and lists rules or constraints on its value and usage. It may also specify contextual constraints such as which children it must contain or which behavior(s) its parent property must

have (for example the maximum limit behavior is a constraint on its parent property that must have some sort of numerical value).

A DDL behavior definition shall associate three pieces of information.

- A name by which a controller can recognize that behavior and that is used to reference this behavior wherever it is applied to a property.
- A derivation that declares specific, more fundamental, or abstract behaviors from which this is refined by a process of specialization.
- A specification that defines the semantics of the behavior – what a property with this behavior means or does – and specifies any applicable constraints.

3.9.2.1 Reading Behavior Specifications

The specification part of a Behavior definition is written in structured text and is intended primarily to be read and understood by designers creating controller applications to enable them to write the necessary code to handle any special functionality they wish to implement to support the behavior.

A secondary target of the text is a user. While users cannot normally reprogram controllers to support specialized behavior, the description may assist them in controlling the equipment using whatever interface or facilities are available to them or guide them in assigning properties to particular control surfaces.

In view of the dual readership of the description part of a behavior declaration, it should be written carefully with an initial overview section followed by unambiguous technical detail.

3.9.2.2 Reading and Displaying Behaviors

The syntax of the DDL behavior element provides for structuring of the description text using paragraph elements arranged in nestable sections with headings. However, reading raw XML is not for the faint hearted and is certainly not expected to be imposed on the ordinary user.

There are a wealth of tools available for managing, manipulating and displaying XML and Behavior definitions – as with all of DDL – are best viewed using suitable tools rather than attempting to read the XML directly. A simple method is to develop a stylesheet that a browser can use to present an XML document in a readable form. However, off-the-shelf XML tools can be configured to go much further, for instance by extracting individual behavior descriptions, formatting them according to language and other preferences and displaying them within browsers or other applications.

A version of the behavior module transformed into HTML using XSL [XSL] is provided for easier reading. This version is not normative.

3.9.2.3 Availability and use of Behavior Definitions

In order to correctly interpret DDL, the behavior definitions must be available for not only the behaviors used within the device, but recursively for all the behaviors from which those are refined down to the most basic level. The mechanism for making DDL modules available depends on the rules governing generation and publication for particular protocols or environments. These rules should ensure that all necessary parts are made available together.

It is the choice of the controller designer to decide the extent to which individual behaviors are recognized and acted on when parsing a description.

For example in the polar axis scenario above:

- A very simple controller for diagnostic purposes could recognize only the base behaviors scalar and

string. It could also present the additional behavioral information to the user if asked. The user, sees simply a set of scalars and strings with descriptive information available, and is able to control them.

- A more sophisticated controller could recognize the minimum and maximum as constraints and use them for example to scale a slider between the two and to label the display.
- A mid range lighting controller recognizes theta as well and knows to assign it to a trackball's X-axis.
- Finally, the most sophisticated controller confronted with the same description, not only recognizes theta but understands its meaning well enough to combine it with scale and units properties to do polar to rectangular transformations and present to the user in x,y,z form.

3.9.3 Behavior Derivation and Refinement

3.9.3.1 General

Any Behavior definition should specify one or more other behaviors that it refines and is derived from. This is analogous to object inheritance in programming languages that allows the creation of classes of behavior based on generic or abstract concept behaviors that are progressively refined to more specialist ones. Thus, for example, the maximum behavior mentioned above would be derived from a more generic limit behavior whilst other more specific behaviors could then be refined from maximum if required (e.g. to distinguish between inclusive and exclusive limits).

A controller encountering but not recognising a very specific behavior may yet have some provision for the more generic concepts from which the generic behavior derives. For example, a property with behavior representing rotation about a specific axis, refines the more generic behavior of rotation about an arbitrary axis. By tracing the refinement backwards from the specific, a less dedicated but nonetheless useful control approach may be available.

It is not required that a behavior definition be refined from pre-existing behaviors, but there are already a rich set of behaviors available [acnbaseDDL] and it is strongly recommended that wherever possible, new behavior definitions should be refined from these, as this improves the coherence and consistency of approach to control across devices and greatly improves the chances that a device containing new behaviors will be controllable by controllers from other manufacturers. This applies both to existing controllers, which may be able to provide partial control based on the behaviors from which the new ones are derived, and to future controllers whose designers are more likely to add provision for behaviors that fit within existing paradigms.

3.9.3.2 Multiple Derivations

A behavior may be a refinement of more than one other behavior, particularly where those others represent independent concepts. The derived behavior is then a combination of the two or more behaviors it refines (with additional meaning as declared in its specification.

For example, a behavior that represents a label string refines not only the more generic behavior of labeling (associating a human readable annotation with another property), but also the datatype behavior associated with strings. Alternative refinements of the labeling concept might combine it with a referencing behavior that retrieves the label text via an index, or with a pictorial behavior.

3.9.4 Predefined Behavior

The DDL specification itself (this document) does not define any behaviors. Each protocol using DDL may define behaviors in its specification that are appropriate to its access methods and the kind of equipment it may be required to control.

All protocol specifications must define at least one device reference behavior if they are to allow modular construction of devices (see: Declaration of of Sub Devices) other than by static inclusion.

There is a rich set of behaviors defined as part of the ESTA ACN Standardization. While some are DMP specific, most of these behaviors may be used for almost any protocol ([acnbaseDDL]).

3.9.5 User Defined Behaviors

Additional behaviors may be declared as required by DDL authors. The corresponding definitions shall be made available according to the rules of the protocol or environment in which they are used.

It is a goal that the set of defined behaviors that the designer of a controller may need to support is kept as small as possible. Authors of DDL should make every effort to use existing behaviors where possible. If defining new behaviors they should seek consensus among users with similar requirements and should derive them as refinements of existing behaviors. When creating new behaviors that are very different from existing ones, authors should also create intermediate behaviors in steps of refinement that seek to preserve as much generality and commonality with other applications as possible at each stage of refinement.

Devices using little-known behaviors that do not extend existing better known behaviors are likely to have greatly reduced interoperability.

3.10 Labels

A label may be attached to many elements and is intended as an indication of the function of that element to the end user or operator. It is entirely up to a controller to choose whether, when and how to present labels to the operator. Since labels are optional and their text is completely free, they are not suitable for use by a controller or other automated system in making algorithmic decisions.

Labels may either contain directly supplied text or may reference string resources (see: String Resources), which provides a mechanism for multilingual labeling.

3.11 Property Values Types and Sizes

3.11.1 Value Types

The interpretation of a property value depends on the behavior associated with it, while the encoding and size are largely a matter of the protocol used to access it (see: Property Values). When an immediate constant value is supplied, it is a raw text string (in the XML serialization) and the processing application must know how to parse it. This information must be implicitly or explicitly provided.

3.11.2 Size is not Part of Value Type

Data size is not necessarily given with a value type. This is because it is up to the controller to choose it's own data representations, because sizing may vary from protocol to protocol and because size information may be misleading. Size may be specified as part of the protocol reference that may also specify a low level protocol data type if necessary to distinguish the encoding “on the wire”.

4 DDL Syntax

4.1 Introduction

The DDL syntax uses XML [XML] as a structure for rendering the device model in a textual format. This format goes beyond simply rendering the device model as it also provides the means to structure the documentation of devices in a modular fashion.

Because the device model is abstract, it would be possible to represent it using different formats and it is possible that alternatives may be developed in the future.

Note

Unrecognized elements in DDL shall be ignored by DDL interpreters. Future revisions of the DDL specification may add additional elements that must be ignored by implementations of earlier versions.

4.2 XML

Extensible Mark-up Language provides a framework for defining specific mark-up languages – DDL is such a language. The reader should be familiar not only with the XML standard [XML] but also with associated standards that are called out by this specification that makes no attempt to define or introduce XML beyond referring to these standards.

Note

Rules presented here that define restrictions on how XML may be used, apply to DDL documents as presented in the context of this standard for interoperable exchange between device and controller – this defines the flavor of XML that a controller or other consumer of DDL should expect to encounter whether those descriptions come from the device themselves, from manufacturer's sources or from elsewhere. Syntax restrictions (if any) for internal use within an application, for private use within an organization or for purposes standardized elsewhere are beyond the scope of this Standard.

4.2.1 Producers, Distribution and Consumers of DDL

DDL is *produced* by manufacturers of equipment and any other person or organization wishing to describe devices using the language.

It is *distributed* by many means including, but not limited to: supply over a network by the device itself, supply over the internet by the manufacturer or other organization, supply on transportable media (CD-ROM, USB-stick, Floppy disk etc.).

DDL is *consumed* by any application that reads and interprets one or more DDL descriptions. This includes controllers and monitors of equipment, applications presenting descriptions or parts thereof for human readers and applications transforming DDL into other forms.

4.2.2 Use of an XML Subset

While XML has the benefit of being a widely used off-the-shelf specification for which many software tools are readily available, it is still a fairly complex language with many subtleties and arcane features. On top of basic XML, a lot of other specifications have been built, not all of which are freely inter-operable that adds to the complexity. This means that the more all embracing XML processors and technologies are too complex for the straightforward needs of DDL particularly in lightweight systems. For these reasons a restricted subset of XML is defined for use in DDL.

4.2.2.1 Encoding and Detection of Encoding

DDL documents shall be encoded in Unicode encoding forms UTF-8 or UTF-16 [Unicode]. or in 7-bit US-ASCII [US-ASCII].

UTF-16 is indicated by the presence of a byte order mark that shall be present if this encoding is used. Since US-ASCII is a subset of UTF-8, in the absence of a byte order mark or explicit encoding declaration, a consumer of DDL that treats the document as UTF-8 will not encounter or generate any errors.

4.2.2.2 Restrictions

DDL shall not contain the following XML productions:

- PI
- doctypedecl
- CDSect
- EntityRef – except the predefined entities
- Mixed content

4.2.2.3 XMLDecl

An XML declaration (XMLDecl) is not required. If an XML declaration is supplied it shall specify `version="1.1"`.

Encoding declarations if present must indicate the encoding used. Note though that in order to read the encoding declaration, encoding will have already been established so this is not relevant to parsers.

SDDecl if present must indicate `standalone="yes"`

4.2.3 DTD, Schemas and Validation

In XML the syntax and grammar rules for a particular language or document format created from XML are called a schema. A document type declaration (DTD) as described in the XML1.0 recommendation is a form of schema and a number of more advanced schema languages and forms have been published.

4.2.3.1 DDL Document Type Definition (DTD)

A DTD for DDL is given in Generic DTD DDL and schemas specific to DDL with DMP are also available. These are made available as an aid to creation and parsing DDL using standard tools but are non-normative. Because further rules are also given in the text and because the capabilities of schema languages vary, validation against a DTD or particular schema does not guarantee full conformance to the DDL syntax.

4.2.3.2 Use of Validation

An author creating DDL documents by whatever method should validate DDL documents when they are created. The author must be aware that validation against a formal schema does not guarantee that the document is correct or even that it makes sense.

Interpreters of DDL documents (controllers etc.) are not normally expected to validate DDL documents. Usually they will be parsing XML for specialist purposes that go much deeper than formal validation anyway.

4.2.3.3 Embedding Schemas in Documents

A DDL document shall not contain a Document Type Declaration (XML production: doctypedecl) either internal or external. Nor shall it contain any other embedded schema. Links with DTDs or schemas for validation purposes must be made externally.

4.2.4 Namespace

This DDL version is identified with the namespace URL: “http://www.esta.org/acn/namespace/ddl/2008/”. See [XMLnames].

In common usage of DDL within control systems documents are homogeneous DDL and have no requirement for namespaces. However, the namespace above is provided for use when required.

Compliant controllers must be capable of recognizing namespace declarations and identifying the namespace of elements and attributes, but are not required to process nodes from any other namespace – such nodes may simply be discarded.

If namespace declarations are present, the DDL namespace shall be the default namespace within any <DDL> element.

4.3 Attribute and Element Value Types – XSD

XML attributes and elements with text content can represent names, numbers, dates and a host of other things. To standardize the text representation of such things, DDL uses the data type representations of XML Schema Part 2: Datatypes [XSD]. The prefix "xsd:" is used throughout this specification to denote such a type e.g. "xsd:nonPositiveInteger", "xsd:NCName". Please refer to XML Schema Part 2: Datatypes. for further information and full specification.

4.4 DDL Modules – Devices, Behaviorsets and Languagesets

Three elements in DDL descriptions are defined as "modules". These are device, behaviorset and languageset.

Each module shall be assigned an identifier that is unique to that definition. This identifier may be used as a key to recognize already known modules or to retrieve a document containing the definition for a specific module.

A module identifier shall be a 128-bit universally unique identifier (UUID). The algorithm used to generate these is specified in [UUID]. This choice of algorithm means that an author of device descriptions can generate module identifiers without explicit recourse to an authorizing body. It also means that where required, a computer can generate its own identifiers (e.g. if there is a requirement to generate DDL on the fly).

Each DDL document (as defined by the XML document definition) shall contain exactly one module.

4.4.1 Module Content May not Change

All copies of a DDL module definition as identified by a given UUID, shall have exactly the same information content no matter where or how they are stored, or retrieved. This means that copies may vary in such things as encoding (subject to encoding rules see: Encoding and Detection of Encoding), insignificant white space, order of attributes etc. but the result of parsing any copy of the module, however transferred must always be the same.

For purposes of this rule XML comments shall not be regarded as part of the information content.

If any change is made to the information content of a module, then a new UUID shall be assigned and a new module is created.

4.4.2 Module Versions and Variations – alternatefor

If any change is made to the information content of a module, then a new UUID must be assigned. There is no explicit version mechanism provided in DDL and changes or improvements to modules may be made by different bodies from the original provider.

However, a DDL module that is intended to replace or update an existing module (e.g. for corrections, addition of extra languages, minor upgrades, etc.) may indicate so using alternatefor.

Authors using alternatefor must ensure that the new module they are adding it to genuinely covers all instances of devices conforming to the module it is replacing. They must consider that the old module may

have been re-used by other users (as happens with low level modules) or may be used in the future without consideration of the newer version.

The interpretation of alternatefor by processors is advisory only. alternatefor suggests replacement and in many cases the replacement will be an improvement, however, since there is no process for registering, certifying or policing descriptions this cannot be guaranteed in all cases.

4.4.3 Module Extension – extends

Another case of variation on a module occurs when the original is completely unchanged but new features are added – the new module is a superset of the original. A module can indicate this using extends that identifies the module upon which the current one is a superset. A processor can then recognize the underlying module even though the extended one has a new UUID.

The rules and semantics of extends depend on the type of module, as explained in clauses 4.4.3.1 through 4.4.3.3.

4.4.3.1 languageset

A languageset may be extended by adding versions of existing strings in languages in which they were previously undeclared, by adding completely new languages, or by adding new strings. Existing alternatefor declarations shall remain unchanged but new ones may be added where new languages are added. All the strings of the original languageset shall be present with unchanged keys and text in all the same language versions.

4.4.3.2 behaviorset

A behaviorset may be extended by adding new behavior definitions. Provided all the behavior definitions of the original behaviorset are present unchanged then the new behaviorset is an extension of the original.

4.4.3.3 device

For a new device to extend an original, the entire property tree of the original shall be present with identical functionality in the new device. New properties may be added provided they do not in their default states cause different behavior from the original device in any way. A controller must be able to access and control the new device exactly as though it were the original type and achieve the same functionality as the original would have provided.

4.5 Root Element

The root element of any DDL document shall be DDL. This element shall contain exactly one module (see: DDL Modules – Devices, Behaviorsets and Languagesets).

4.6 Warning: Documents, Files and Resources

The XML syntax defines a document that is a logical structure. A document as defined by XML does not necessarily correspond with a file as defined by a computer filesystem or with a single resource as referenced by a URI. There may be applications or environments where it is necessary to combine multiple resources or files to create a document, or to separate a single file or resource into multiple documents. These cases are beyond the scope of this specification but must be clearly defined where they occur.

4.7 String Resources

As part of documenting devices, DDL provides a mechanism for collections of string resources to be stored in multiple alternative languages within DDL documents. These resources are available for labeling parts of the description, but are also available to devices themselves to call by reference as property values, either

directly or indirectly via mechanisms such as selectors (referencing of strings from property values depends on appropriately defined behaviors). This means, for example, that a property that returns a numeric error code can (and should) be described in such a way that a controller can report verbose error messages in multiple languages in response to those error codes.

Each string associates a text – the value of the string, which is intended for presentation to the user or other human readers – with a key that is a short name that identifies the string by lexical matching.

Strings are contained in language sets that define a set of keys and their replacement texts in various languages. There shall be no more than one text value per key, per language in a languageset.

4.7.1 Languages and Search Order

A languageset element contains one or more language elements. Each language element contains a set of strings in the (human) language specified by the lang attribute of the language element.

Within a language element each string key shall be unique. However, the same key may appear in multiple language elements, and the replacement text will therefore be in the language defined by the language element in which the string occurs.

Example 5: example of languageset definition

```
<languageset ...>
  <language lang="en-us">
    <string key="colr">color</string>
  </language>
  <language lang="en-gb">
    <string key="colr">colour</string>
  </language>
  <language lang="fr">
    <string key="colr">couleur</string>
  </language>
</languageset>
```

In this example three alternative language translations corresponding to the key “colr” are provided: American English, British English, and French.

language elements are not required to include text for all keys present in the languageset, but a language element that does not include strings for every key shall specify an alternative language to substitute if a value for a key is not present. Thus given a key and a preferred language, a processor first searches the language element corresponding to the preferred language for a string, and if one is not found repeats the process for the specified alternate language, and so on recursively until a value is found.

Example 6: languageset definition with incomplete languages

```
<languageset UUID="9fcb433e-8dd4-4418-9559-cb1a8e51648a">
  <language lang="en-us">
    <string key="colr">color</string>
    <string key="tim">time</string>
  </language>
  <language lang="en-gb" altlang="en-us">
    <string key="colr">colour</string>
  </language>
  <language lang="fr">
    <string key="colr">couleur</string>
    <string key="tim">temps</string>
```

```
  </language>
</languageset>
```

Here the British English language element does not include all strings but specifies that for strings not present, the American English language element should be used.

The chain of alternate languages shall produce a text value for any key that is present in the languageset irrespective of which language is first chosen (out of those represented anywhere in the languageset). Any failure of this rule will either result in a dead-end at a language that does not specify any further alternative, or will result in a loop of alternative languages that repeats itself. Both conditions are errors.

The specification of an alternate language is optional, but if a language element does not specify an alternate then it must contain values for every key in the set.

One easy way to ensure that any key generates a value without recursive loops in the search is to ensure that there are one or more languages within the set that contain values for all keys in the set, and that all chains of alternate languages terminate at one such language.

4.7.2 Identification of Strings

Following the rules above, to unambiguously refer to a string text, three items are required:

- The languageset containing the string
- The string's key
- The preferred language (in the form of a language tag [LANG-TAG]).

The first two items must be provided by the description or by the device itself anywhere a string reference is used. The preferred language is normally a matter for the processor that is presenting the values and is typically a user configuration choice.

Example 7: Labels Using String References

```
<device>
  <UUIDname UUID="9fcb433e-8dd4-4418-9559-cb1a8e51648a"
            name="exampleStrings"/>
  ...
  <property ...>
    <label set="exampleStrings" key="colr">
    ...
  </property>
  <property ...>
    <label set="exampleStrings" key="tim">
    ...
  </property>
</device>
```

The property labels above use the strings in Example 6: . With preferred language set to “fr” the two property labels are “couleur” and “temps”, with language “en-us” they are “color” and “time” and with language “en-gb” they are “colour” and “time”.

4.8 Arrays of Properties

Arrays of properties of the same type or function are common in devices and can range from simple array values (e.g. the levels of a graphic equalizer or an arbitrary dimmer curve), to arrays of complete sub-trees or sub-devices (e.g. an automation rack with an array of motor controllers).

In order to avoid repetitive text and to highlight to controllers where arrays of identical properties occur, the

syntax allows any property or included sub device to be declared as an array. This means that not only that property or sub device but the entire sub-tree of which it is the root is actually an array of identical sub-trees.

The mechanism for defining the different addresses of network values within an array is protocol dependent and must be specified as a part of the access protocol interface specification.

Where properties within an array are declared with immediate values, then an array of values must be provided.

4.8.1 Multidimensional Arrays

A property with sub-properties forms a branch of the tree and if declared as an array, it forms an array of branches. If one among its sub-properties is also an array, then an array of arrays or multidimensional array is declared. Thus while the specification of individual properties only allows array declarations in one dimension, it is quite possible to generate multidimensional arrays by nesting arrays within arrays.

In order to avoid ambiguities, applications must keep track of the count of each dimension of an array individually – that is, an array of m properties with a child array of n properties must be treated as a two dimensional array [0 .. m-1][0 .. n-1] and not simply as an array of m×n properties [0 .. m×n-1].

4.8.2 Properties That Do Not Iterate Across an Array

In some cases of arrays, it is required for some properties in the sub-tree not to iterate with the array. For example, an array of scalar variables may share a common maximum limit. This is an application for the generic shared property mechanism for representing properties with multiple parents and in the simplest case, there is a special simplified mechanism (see: Shared Properties).

4.8.3 Immediate Values for Array Properties

If an array property has an immediate value, then in general a value must be provided for each property of the array. However, the case where all elements of the array have the same immediate value is so common that a special case is made and it is not necessary to use the shared property mechanism.

In general for any array property with an immediate value, the description shall provide either one single value that applies to all elements of the array, or a number of values equal to the maximum number of properties in the array. For example, for a property with array size [a][b][c] then there shall either be *1* or $a \times b \times c$ values. That is, if there are multiple values, then these must be given separately for each element within each dimension of the array.

For multidimensional arrays property values shall be provided in inner array first order. For example for a property array size 3 containing a child property array size 2 the values shall be in order $V_{0,0}$ $V_{0,1}$ $V_{1,0}$ $V_{1,1}$ $V_{2,0}$ $V_{2,1}$

4.8.4 Parametric Array Sizes

When the size of an array is determined by a parameter value (see: Parametric Devices), then it is not always possible to specify exactly the right number of values for each property of the array. However, rules governing parametrization of array sizes require that a maximum size is always known and this is the number of values that shall be supplied. It is therefore not an error for there to be more values provided than the actual size of the array determined by the parameter value.

In the case of multi-dimensional arrays, where the size in one or more dimensions is supplied by a parameter, a DDL processor must use the maximum value for the array size rather than the supplied parameter, in calculating how to iterate between higher dimensions in the array.

4.9 Element Identifiers

The XML specification defines the use of two types of attributes, ID and IDREF, as a mechanism of cross-linking between XML elements. This mechanism is extensively used and implemented by XML processors for a wide variety of purposes. A very brief specification [xml-id] outlines the use of the attribute name "xml:id" for ID type attributes so that they can be easily recognized in the absence of specific schema information.

Nearly all elements within DDL may carry an xml:id attribute that shall conform to the [xml-id] spec. As well as use within generic XML tools (it is useful in Xinclude [XInclude], Xpointer [XPointer], Xpath [XPath] and many higher applications), DDL uses and extends xml:id for structural references to the items represented by the device model. However, this use is constrained where references are limited within a single document so DDL extends this to allow references across devices.

4.10 Use of XML Identifiers to Reference Device Model Features

The use of xml:id attributes on properties, or other parts of the device model provides a mechanism to build cross references between parts of devices. In order to do this, any reference must identify an instance of the device. The match then identifies by way of the DDL, the property or other item represented by the element within the device that is referenced. Finally, since in the device model a match might identify a whole array of items (represented by just one element in the actual XML), rules for referencing across arrays may need to be applied.

References of this sort are used directly by the shared property mechanism. They are also used extensively by a number of behaviors that rely on pointers or references between properties.

4.10.1 Reference to Elements in Device Trees – Extended Reference Syntax

The extended reference syntax is specific to references to instances of devices. It is used by the shared property mechanism (shared attribute) and may be used by behaviors or future specifications.

Bearing in mind that each DDL device is a separate module and therefore a discrete and complete XML document.

- The syntax defines an extended reference using a series of steps.
- Successive steps shall be separated by the character "/".
- For each step there is a context device. Each step except the last shall identify a device relative to the context device that shall then be used as the context device for the subsequent step.
- If the reference begins with /, the context device shall be a root device in an appliance. If there is no explicitly indicated appliance (e.g. by a separate reference to a network component) this shall be the appliance that contains the reference.
- If the reference does not begin "/", the context device for the first step shall be the device containing the reference.
- A step whose text is "." identifies the context device itself.
- A step whose text is ".." identifies the parent device of the context device.
- A step whose text is anything else shall be an XML Name, which may optionally be followed by array subscripts and/or an array matching specifier – see below. The Name is an xml:id reference within the context device and matches the xml:id attribute of an element within that device. e.g. step "foo" identifies the element with `xml:id="foo"`.
- A step that identifies an includedev element and that is not the last step, establishes the device instance specified by the includedev as the context device for subsequent steps (see: Statically Included Device Descriptions).
- A step that identifies an property element that has device reference behavior and that is not the last

step, establishes the device instance specified by the device reference as the context device for subsequent steps (see: Sub Devices Attached by Reference).
- Any step that is not the last and that does not identify either an includedev, or device reference property is an error.
- The last step may identify any element within the context device established by all preceding steps.
- *Array Subscripts:* A reference that identifies an element that is a part of a property or includedev array, represents multiple items in the description. The specific item may be indicated by subscripting the xml:id reference step using square brackets. If subscripting is used, there shall be exactly one subscript for dimension of the array within the context device. Multiple subscripts shall occur in the same order that the array elements are nested – outermost first. e.g. `ref="foo[3][201]"`.
- A step that identifies an element that is a part of a property array and for which no subscript is specified is subject to the rules in section References in Arrays.
- A reference that is itself in an array (by being declared as a descendent of an array property) is an array of references and shall resolve according to the rules in section References in Arrays.
- *Array Matching Specifier:* When a reference is made from within an array the last step of the reference may be an Array Matching String enclosed in either single or double quotes. The Array Matching Specifier shall follow any subscripts provided in the same step.

4.10.1.1 Examples

`ref="localprop"` references an element with `xml:id="localprop"` in the current device.

`ref="mysubdev/eleminsubdev"` references element with `xml:id="eleminsubdev"` in the device whose `xml:id="mysubdev"` in the current device.

`ref="/foo[8][12]/bar[5]/boff"` references element with `xml:id="boff"` in subdevice 5 in the array of subdevices with `xml:id="bar"` in subdevice number 12 in sub-array number 8 in the array of subdevices with `xml:id="foo"` in the root device of the appliance.

4.10.2 References in Arrays

In DDL both a reference and/or its target may occur within an array. This creates four types of reference: one-to-one; one-to-many; many-to-one and many-to-many. propertypointer references cannot be one-to-many as this creates ambiguity. The rules provided below apply to propertypointer and any property-reference behaviors in which one-to-many references are forbidden and ensure that references can be unambiguously resolved.

A reference to an array that specifies a subscript is pointing to a singular item. In the general case of multidimensional arrays, a subscript in the reference may still identify a sub-array of targets so for each subscript provided the number of dimensions of the target array is reduced by one.

In the multidimensional case, taking a single dimension at a time, there are two ways to match an array of references to the target: correspondence match and many-to-one match.

4.10.2.1 Single Dimension Correspondence Match

The array size of the target item must be identical to the array size of the reference. Each reference shall be interpreted as pointing to the item at the same position in the target array as the position of the reference in its own array (1:1 correspondence).

4.10.2.2 Single Dimension Many-to-one Match

The target is a single item. Each reference points to the same item.

4.10.2.3 Multidimensional Arrays

In the case of multi-dimensional arrays, the same reference types are generalized, with each array being an array of items, and each item being a sub-array, except the innermost case where items are singular.

When a reference is made within an *R* dimensional array, and the target element is within an *T* dimensional array. The following rules are applied recursively until all dimensions are matched:

- If $R < T$ it is an error and no match can be made.
- If $R = T$ then matching shall be by multidimensional correspondence match as described below.
- If $R > T$ then matching shall be either by multidimensional correspondence match, or by multidimensional many-to-one match as described below.

4.10.2.3.1. multidimensional correspondence match

The size (the array item count) of the outermost dimension of the reference array shall be exactly equal to the size of the outermost dimension of the target array. Each item in the reference array points to the corresponding item in the target array. Each reference item is now a *R-1* dimensional array of references pointing to a *T-1* dimensional array of targets, for which these rules are applied again.

4.10.2.3.2. multidimensional many-to-one match

All members of the outermost dimension of the reference array, point to the same target array. There remain *R-1* dimensions of reference, pointing to a target array that still has *T* dimensions for which these rules are applied again.

4.10.2.3.3. Notes

It follows from the rules above, that in order to make matches between multidimensional arrays of references and targets:

- the number of dimensions of reference must be greater than or equal to the number of dimensions of target ($R \geq T$);
- for each dimension of target, there must be a correspondence match with a single dimension of reference in which the array sizes must be the same;
- any excess dimensions of reference must be many-to-one matches;
- the ordering of dimensions that have correspondence matches must be the same in both arrays.

4.10.2.4 Array Matching String syntax

An array matching string specifies how the dimensions of an array of references are to be matched to an array of targets:

- Each character in the string shall correspond to a single dimension of the reference array with the first (leftmost) character representing the outermost array dimension, and the last representing the innermost.
- Each character shall be either “=” or “*”.
- A dimension whose character is “=” shall be correspondence matched with the target array according to the rules above.
- A dimension whose character is “*” shall be many-to-one matched with the target array according to the rules above.

It follows from this, that where there are *R* dimensions of reference to *T* dimensions of target the string must have *R* characters of which *T* must be “=” and the rest “*”.

4.10.2.5 Default Matching for Multi-dimensional Arrays

In the absence of explicit specification (e.g. by supplying an array matching string), a multidimensional match from an *R* dimensional array of references to a *T* dimensional array of targets shall be performed by correspondence matching the first (outermost) *T* dimensions and many-to-one matching any remaining dimensions.

4.10.2.6 Examples

These examples all use propertypointer references to illustrate matching – see below.

The following XML creates a 3×4 array of target properties named $Tgt_{i,j}$ that can be visualized like this:

$Tgt_{0,0}$ $Tgt_{0,1}$ $Tgt_{0,2}$ $Tgt_{0,3}$
$Tgt_{1,0}$ $Tgt_{1,1}$ $Tgt_{1,2}$ $Tgt_{1,3}$
$Tgt_{2,0}$ $Tgt_{2,1}$ $Tgt_{2,2}$ $Tgt_{2,3}$

```
<property array="3">
   <property array="4">
      <property xml:id="Tgt" sharedefine="true">
         ...
      </property>
      ...
   </property>
   ...
</property>
```

Example 8: One-to-one Match

This is a one-to-one matching array of references named Refi,j:

$Ref_{0,0} \rightarrow Tgt_{0,0}$, $Ref_{0,1} \rightarrow Tgt_{0,1}$ ... $Ref_{0,3} \rightarrow Tgt_{0,3}$
...
$Ref_{2,0} \rightarrow Tgt_{2,0}$ ... $Ref_{2,2} \rightarrow Tgt_{2,2}$, $Ref_{2,3} \rightarrow Tgt_{2,3}$

```
<property array="3">
   <property array="4">
      <propertypointer ref="Tgt"/>
   </property>
</property>
```

Example 9: Mixed one-to-one and many-to-one matches

This is a 3 dimensional array of references in which the second dimension is many-to-one matched:

$[Ref_{0,0,0}...Ref_{0,99,0}] \rightarrow Tgt_{0,0}$, $[Ref_{0,0,1}...Ref_{0,99,1}] \rightarrow Tgt_{0,1}$ ... $[Ref_{2,0,3}...Ref_{2,99,3}] \rightarrow Tgt_{2,3}$

```
<property array="3">
   <property array="100">
      <property array="4">
         <propertypointer ref="Tgt'=*='"/>
      </property>
   </property>
</property>
```

4.11 Shared Properties

Section 3.4.1.2 introduced the idea of shared properties that represent places where branches of the property tree recombine so a property has multiple parents. A shared property in DDL is declared in multiple places. Exactly one declaration shall be a defining-declaration. All other declarations of the shared property shall be reference declarations that are called propertypointers.

4.11.1 Defining-declaration

The defining-declaration of a shared property is a normal property declaration in which the full structure of the property including all its behaviors, its value, its sub-properties etc. appear. It is identified as a shared property by setting its sharedefine attribute to “true” or to “arraycommon”. Since it will be referenced using the reference syntax above, it must also carry an xml:id attribute.

It is not an error if a property with `sharedefine="true"` is not referenced. It may be referenced from outside the device and the presence of such references may depend on how this device relates to other devices in a particular appliance.

4.11.2 propertypointer

A *propertypointer* contains a ref attribute whose value is a reference that obeys the syntax rules for extended references and references in arrays (see: Reference to Elements in Device Trees – Extended Reference Syntax, References in Arrays). It has no behaviors, child elements or other structure – these must be declared at the defining-declaration – but in interpreting the DDL, the propertypointer represents the same property as the defining property to which it points in each place it is declared, so by definition all its behaviors, values and sub-properties are the same.

The rule in Section Branch Re-combination and Shared Properties forbidding any property to be simultaneously both parent and child of any other, or of itself means that a propertypointer cannot resolve to itself or to any of its ancestor properties whether directly, or indirectly via any number of levels of references or parent or child devices.

It is permitted for a propertypointer to reference another propertypointer (they both represent the same property) provided that such a chain of references resolves to a full property declaration eventually. The target of the ref attribute of a propertypointer shall be a property whose sharedefine attribute is set to either “true” or “arraycommon”.

It is not an error if a propertypointer does not resolve when considering a device description in isolation, but It is an error if any instance of a propertypointer, does not resolve unambiguously to a defining-declaration of the shared property when the device is built, together with other devices, into an appliance (and after any parameter substitutions have been made. See: Parametric Devices). Note that while the xml:id of a defining declaration cannot be parametrized, it is allowable to parametrize ref attributes to ensure that propertypointers within template devices resolve correctly.

4.11.3 Special Case – Property Shared Across an Array

The case that a common sub-property exists within an array that is the same property for all branches of the array occurs sufficiently often that a special-case syntax is available to improve readability.

Any property within an array (of any number of dimensions) whose sharedefine attribute is set to *arraycommon* shall be interpreted as a single property that occurs in all branches of the array.

The case where a property with attribute `sharedefine="arraycommon"` is not in an array is the special case where the array size is 1 and is not an error.

The case of a property that has both attribute `sharedefine="arraycommon"` and attribute array="*N*" where $N > 1$, is forbidden.

A property with `sharedefine="arraycommon"` may be referenced with a propertypointer in the same way as for `sharedefine="true"`. As a single property, any references to it from within an array must follow many-to-one matching rules.

Note

The very common case where a sub-property within an array has an immediate value and no children does not require `sharedefine="arraycommon"` because a single value is automatically applied to all items in the array.

4.12 Declaration of of Sub Devices

As mentioned in Hierarchical Device Structure, sub devices in DDL may occur anywhere that one or more properties are legal and become a logical part of the entire device property tree. DDL provides two mechanisms for declaring sub devices that have different applications.

4.12.1 Statically Included Device Descriptions

Any device description (identified by its UUID) may be included within another description at any point that a property is allowed using an includedev element. When included in this way, all the properties of the included device become part of the tree of the description that includes them, at the point they are included.

Devices included by static inclusion are entirely specified within the text of the description and so because of the rules preventing changes (see: Module Content May not Change), cannot be used for dynamically variable sub devices (e.g. interchangable modules). For example, if description of A includes device B, then B becomes a fixed part of A.

Static inclusion does however, provide the opportunity to specify parameters to the included device and this mechanism allows a template description to be included with different parameters in different places (see: Parametric Devices).

Access protocol specific information may also be specified at the point of inclusion depending on the rules of the specific access protocol(s).

Example 10: Static Inclusion Sample DDL

```
<DDL version="1.1">
  <!-- First we define a color wheel device -->
  <device UUID="c158b08d-e03e-43b3-88b2-d56dc1447155" ...>
    <label>Color Wheel</label>
    <property valuetype="network" ...>
      <label>color selector</label>
      ...
    </property>
  </device>
  <!-- Now we can use the definition -->
  <device UUID="77ee2876-ed3a-4728-b789-2a330d55c051" ...>
    <property ...>
      <label>color changing luminaire</label>
      ...
      <includedev UUID="c158b08d-e03e-43b3-88b2-d56dc1447155"/>
      ...
    </property>
  </device>
</DDL>
```

4.12.2 Sub Devices Attached by Reference

The second method uses a property to mark an attachment point where a sub device is attached to the

description tree. This property is called a device reference property and shall be identified by a suitably defined behavior. The device reference behavior must be defined in the basic set of behaviors for any protocol adaptation in which this method of attachment is to be used. The value of a device reference property is the UUID of the device attached. So far as the control model is concerned, the property tree(s) of the sub device can be considered to replace the property that marks its attachment.

Because the device attached in this way is specified by a property value, it follows all the usual rules for property values and may be specified as an immediate value embedded within the description or a network value that is accessed over the network. If a network value, it can be constant or may vary – for instance as hot-plug modules are added or removed or as a software defined device such as an Audio DSP is reconfigured. The UUID may also be a driven value – for example driven by a selector property to select a value from a set of choices, thus allowing network control over the choice of sub-device.

Attachment of sub devices by reference does not however, allow parameters to be directly specified for the attached device. (see: Parametric Devices).

Example 11: Device Reference Sample DDL

```
<DDL version="1.1">
  <!--
    Define a color wheel device
    (same description as previous example)
  -->
  <device UUID="c158b08d-e03e-43b3-88b2-d56dc1447155" ...>
    <label>Color Wheel</label>
    <property valuetype="network" ...>
      <label>color selector</label>
      ...
    </property>
  </device>
  <!-- This luminaire has an optional color wheel -->
  <device UUID="77ee2876-ed3a-4728-b789-2a330d55c051" ...>
    <property ...>
      <label>color changing luminaire</label>
      ...
      <property valuetype="network">
        <behavior set="example" name="device_reference"/>
        <!--
          if the color wheel is fitted
          this property will hold the value:
            c158b08d-e03e-43b3-88b2-d56dc1447155
          otherwise it will hold NULL indicating no sub-device
        -->
        ...
      </property>
    </property>
  </device>
</DDL>
```

As shown in the example, a device reference property whose value is the null UUID (all zeros) shall indicate that no subdevice is present.

4.12.3 Circular References

Whenever a the UUID of a subdevice is declared within a description, whether by static inclusion or using a

device reference property with an immediate value, there is no way that the UUID can change from instance to instance – it is invariant. It is an error for any device description to reference its own UUID as an invariant sub device or, recursively as any invariant descendant (a class cannot be its own parent or ancestor).

It is also an error for any instance of a device to reference itself as a sub device or other descendant.

4.12.4 Which Method to Use?

If a child device needs to be dynamically attached or to vary from instance to instance of the parent then attachment by reference is the only method that can work.

If any parameters of the included device are to be varied then static inclusion can specify these directly and should be used.

If a child device is statically defined and constant for all instances of the parent, and included purely to gain benefits of device re-use, then either static inclusion or attachment using a immediate valued reference property may be used.

4.12.5 Discovery of Device Structure

The mechanism for discovery of a root level device for a piece of equipment (in ACN protocols this is a component) is protocol dependent. However, the DCID is a key that may be used to retrieve the XML text of the definition itself. Within a definition, references to subsidiary devices are made by DCID that then identifies the XML text of their definitions. Thus once a root level device has been identified the entire tree can be constructed, provided that all necessary device descriptions are available.

4.13 Parametric Devices

The rule that module content may not change (see: Parameter Substitution) means that a processor that encounters and recognizes a device class it already knows, can be confident that the description is no different from one it has already parsed. However, there are many examples of devices that have the same control structure but differ in a few parameters. For example many cars share the same control structure of steering wheel, gearstick, clutch accelerator and brake, but differ in top speed, turning circle, gear ratios and so on. It would be possible to describe all such cars using a fixed device class while using network property values for values that differ, but this means that a controller needs to have the actual instance available on line before it can configure itself for specific values. In entertainment technology and any other applications where off-line configuration is commonplace, this is a serious disadvantage.

If, on the other hand, a separate device is declared for each variant, this allows the top speed and other items to be specified within the description as immediate values but because each device has a separately identified description the commonality of structure is lost.

DDL's solution is to allow individual items in the description to be marked as parameters. These are called *parameter fields*. Any device may declare zero or more named parameters, each of which applies to one or more parameter fields.

When a device that contains parameters is statically included in a description (using includedev), values for any of the parameters of that device may be defined. The included device is called a template device.

4.13.1 Parameter Fields

A parameter field shall be either an attribute value or the text content of an element (#PCDATA in DTD terms). In either case the parameter value is a text string and cannot contain XML elements or other markup. Each parameter field is bound to a single parameter name (see below). A parameter name may have multiple fields bound to it.

4.13.2 Defining Parameter Names

In order to define a parameter in a device, one or more parameter elements shall be provided near the top of the device definition. Each parameter element provides a name for the parameter, and may optionally specify various restrictions on the value that can be supplied for that parameter (see: Restricting Parameter Values).

4.13.3 Parameter Substitution

When a template device is included within a parent device using includedev, each of its parameters may have a string value provided (using a setparam element), and this value shall then be used in place of whatever value that was present in the original field(s) when interpreting the DDL of the template device.

Any value supplied in a setparam element shall be valid and legal in all fields that are bound to that parameter and shall meet all restrictions imposed in any parameter declarations declaring that parameter.

4.13.4 Restricting Parameter Values

When a parameter is declared, the range of values it is permitted to take may be left unbounded or may be restricted in several ways. If no restrictions are specified, then any value that is legal for all fields bound to that parameter may be specified for the parameter when the device is included. If a restriction is imposed then it is illegal to specify a parameter value that contravenes the restriction when including the device. Restrictions are defined by choice, mininclusive, maxinclusive or refinement elements.

4.13.5 Binding Fields to Parameters

For any field within a device that can be parametrized, there is a corresponding parameter-binding attribute, whose value is the name of the parameter to which that field is bound. Since this attribute contains a single name, it is impossible to bind a field to more than one parameter. It is an error if the name does not match one specified in a parameter element within the same device.

Each parameter-binding attribute consists of the name of the field (element or attribute name) with the suffix ".paramname" appended. For example, a property array size is specified with the array attribute. The corresponding parameter binding attribute is *array.paramname*, so to bind the array size to parameter "num-props", you might specify `<property array="10" array.paramname="num-props" ...>`. To bind a label with literal content (the parametrized field is the element content, not an attribute) you might write `<label label.paramname="labeltext">...</label>`.

Only those fields for which a corresponding .paramname attribute is explicitly declared in the rules of Section 5 can be parametrized.

One field that shall never be parametrized is the xml:id attribute because parametrizing this would break interoperability with many standard XML processing applications.

Future Extensions to DDL

This mechanism means that if any attribute of DDL has the same name as its parent element then neither the attribute, nor the parent element can be made a parameter field without ambiguity. This does not create any problems with current DDL that has no instances of attributes having the same name as their parent, but must be considered when extending DDL (e.g. for new protocols).

When creating new elements and attributes for DDL (e.g. for new protocols), it should be the rule to make all fields parametrizable, by creating corresponding .paramname attributes, unless doing so would create a specific problem or conflict. It is unacceptable not to create a .paramname attribute simply because there is no obvious reason why a user would want to parametrize the field.

Because parameters are only in scope in the body of a device and after the point of declaration, there are some elements whose attributes can never be parametrized. These include DDL, device, UUIDname, behaviorset, languageset and all child elements of behaviorset and languageset.

4.13.6 Default Values

All of the requirements applying to fields within devices (as specified elsewhere in this Standard) apply whether or not that field is parametrized. Thus any parametrized field must have a valid value where it occurs in the description. If no value for the parameter is supplied when the template device is instantiated, or if the template device is attached by reference, or if it forms the root device of an appliance, then the value used for each parametrized field shall default to that given in place.

It is legal for differing default values to be given in-place for multiple fields of the same parameter, but there is no way to override these with differing values when that parameter's value is explicitly specified on inclusion using setparam. For this reason, supplying differing in-place values for different fields of the same parameter is confusing and rarely a good idea.

4.13.7 Attributes with Default Values

Many DDL attributes have default values specified (e.g. array defaults to 1) in the case they are not explicitly provided. To aid readability and avoid confusion it is strongly recommended that when binding such attributes to a parameter, the attribute itself as well as its .paramname attribute should be explicitly written in the DDL, even if its in-place value is the default.

In the case that such an attribute is not explicitly written, the DDL specified default value for the attribute shall be treated as the parameter default.

4.13.8 Template Example

The DDL shows fragments of DDL from a color-wheel template device whose color count can be modified when included.

Example 12: Static Inclusion With Parameters Sample DDL

```
<DDL version="1.1">
  <!-- First we define a parametric color wheel device -->
  <device UUID="c158b08d-e03e-43b3-88b2-d56dc1447155" ...>
    <parameter name="colorCount">
      <mininclusive>2</mininclusive>
      <maxinclusive>100</maxinclusive>
    </parameter>
    <label>Color Wheel</label>
    <property valuetype="network" ...>
      <label>color selector</label>
      ...
      <property valuetype="immediate" ...>
        <label>number of colors</label>
        ...
        <value value.paramname="colorCount">6</value>
        ...
      </property>
    </property>
  </device>
</DDL>
<DDL version="1.1">
```

```
  <!--
    Now we can use the definition with 10 colors instead of 6
  -->
  <device UUID="77ee2876-ed3a-4728-b789-2a330d55c051" ...>
    <property ...>
      <label>color changing luminaire</label>
      ...
      <includedev UUID="c158b08d-e03e-43b3-88b2-d56dc1447155">
        <setparam name="colorCount">10</setparam>
      </includedev>
      ...
    </property>
  </device>
</DDL>
```

4.13.9 Adoption of Parameters from Included Devices by Parent Devices

As seen above, it is not required to specify values for every (or even any) parameter of a template when instantiating it. Any parameter of a template device whose value is not specified using setparam at inclusion, automatically becomes a parameter of the parent device. Its name gains the same scope as other parameters within the parent and may have values specified in the normal way if the parent is itself included in a grandparent device.

Because of this inheritance, it is possible that multiple parameter definitions using the same name are in scope at the same time. This can occur when the same sub-device is included multiple times, when different included sub-devices have parameters of the same name, or when a parent device defines a parameter of the same name as one used within an included sub-device.

Since parameter matching is performed solely by name, any supplied value for a parameter shall apply to every parameter field whose name matches and is in scope, wherever they are specified. This is not an error and can be used to advantage, but great care must be taken when template devices are included, to ensure that any names that are adopted, either do not conflict, or make sense when a common value is applied in all places.

Note

If a parameter value is fixed at inclusion using setparam, the name of that parameter has no further visibility elsewhere in the parent (or in any device that includes the parent) and any use of the same parameter name elsewhere in the parent has no effect on the included device.

4.13.10 Re-parametrizing Supplied Values

It is legal for the value provided by a setparam element to itself be parametrized (using setparam.paramname). This allows for parameters within included devices to be renamed within the parent, and for new default values to be supplied. This in turn allows naming conflicts caused by adoption of parameters (see above) to be resolved. It also allows parameters to be defined within the parent that apply to fields within both the parent and the included device.

In the following example, the same color-wheel as the preceding example, is included twice in a double color-wheel device in which the color counts for wheel 1 and wheel 2 can be separately specified but default to 7 and 9.

Example 13: Nested Static Inclusion Sample DDL

```
<DDL version="1.1">
  <device UUID="77ee2876-ed3a-4728-b789-2a330d55c051" ...>
    <parameter name="colorCount_1"/>
    <parameter name="colorCount_2"/>
    <label>double color wheel template</label>
    ...
    <includedev UUID="c158b08d-e03e-43b3-88b2-d56dc1447155">
    <setparam
      setparam.paramname="colorCount_1"
      name="colorCount">7</setparam>
    </includedev>
    ...
    <includedev UUID="c158b08d-e03e-43b3-88b2-d56dc1447155">
    <setparam
      setparam.paramname="colorCount_2"
      name="colorCount">9</setparam>
    </includedev>
    ...
  </device>
</DDL>
```

4.13.11 Notes and Additional Rules

The format and content of any parametrized value as supplied in a setparam element shall be governed by the rules provided for the individual field or fields to which that parameter applies and shall be a valid value within all such fields. It is an error if a single parameter applies to multiple fields for which there is no possible common value that is permitted and valid in all those fields.

A parameter shall have a name (specified by the parameter element) that is unique among all the explicitly identified parameters of the same device.

The scope of a parameter name shall be the device in which it is declared (extending from the point of its declaration to the closing </device> tag, excluding any subdevices (except in the case of adoption of parameters from children as defined above). The scope also extends to any device that statically includes this device according to the rules in Section 4.13.9.

Careless parametrization of arbitrary fields or sharing of parameter names between parent and child devices may make a template device difficult or even impossible to use – a template that when instantiated cannot generate legal DDL or can only do so for a single specific parameter value is pointless.

4.13.12 Dynamic use of Parameteric Devices

There is no direct mechanism to specify parameters when a device is attached by reference or when a root device is declared. However, it is quite easy to create a wrapper device that gives the required parameter values around a template and attach that.

In the previous example of a car with steering, gearstick, accelerator clutch and brake, once we have a full description of one such car, we can define another car device that does not introduce any new properties but includes the fully defined car at its top level and at the same time specifies new values for the various parameters. We now have two car devices, each with its own DCID, but a very brief inspection of the description of the second will reveal that it is merely a variant of the first.

4.14 Logical and Syntactic Property Structures

The DDL syntax, often gives rise to properties that are declared for syntactic or structural reasons but that are not logically a part of the device model structure at a given time. This generates the concept of logical and syntactic trees and of structural properties. The syntactic tree is the tree of properties as it occurs in the DDL document and is defined by the XML. The logical tree is the tree of properties that represent device control model at any given time, properties that govern the structure of the logical tree but that are not a part of it are called structural properties. The logical tree is not necessarily fixed but can change when the value of certain properties change (often those in the syntactic tree that are not present in the logical tree).

Example 1: Device Reference

> A device reference property has a value that identifies a sub-device. So far as the logical device model is concerned, the root property or properties of that sub-device are inserted in place of the device reference property to form a tree incorporating both the parent and child devices. However, if the value of the device reference property is changed (representing attachment of a different sub-device) then the entire logical tree from the device reference point down may be different. The device reference property is a structural property.

Example 2: Alternative Property Sets

> A rotating part may be controlled by position (sometimes called indexing) or may rotate continuously and be controlled by speed. In some applications both means are useful and a property is used to switch between two alternative sets of control properties (e.g. see DMP's propertySetSelector behavior). In this case, the primary parent property is the rotation angle and the switch property is syntactically its child while the target position (for positional control) and speed (for speed control) are both children of the switch that chooses which of the two is active. The target position and speed are therefore syntactically the grandchildren of the rotation angle but whichever is active is logically the child of rotation angle since that is how target position or speed work. The control method selector switch is a structural property.

5 DDL Element and Attribute Reference

5.1 alternatefor

empty element

A new version of a DDL module may indicate that it replaces a previous version using the “alternatefor” element. This provides a mechanism for corrections, addition of extra languages, improvements, etc.

Authors using this element must ensure that the new module they are adding it to, genuinely covers all instances of devices conforming to the module it is replacing. They must consider that the old module may have been re-used by other manufacturers (as happens with low level modules) or may be used in the future without consideration of the newer version.

The interpretation of alternatefor by processors is advisory only since there is no formal version mechanism for modules and an alternate version may be provided by any organization.

Attributes

- UUID (required) the UUID the current module is intended to replace.
- UUID.paramname (optional) binds the UUID attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- device

- behaviorset
- languageset

5.2 altlang

attribute

Identifies the language to search if a string matching a particular key is not found in the current language element.

The values of the attribute shall be language identifiers as defined by *Tags for the Identification of Languages* [LANG-TAG] or its successor.

Value

Shall conform to xsd:language

Parents

- language

5.3 array

attribute

Declares that its parent property or included sub device is an array of identical properties, or sub devices. The value of array shall be a non-zero positive integer.

If array is not present the parent property or includedev is a single instance. All child properties of an array property or within an array sub device are part of the array (Section 4.8, “Arrays of Properties”).

Value. Shall conform to xsd:unsignedInt with minInclusive set to 1. Default if not present is 1.

Parameter restrictions

When array is made a parameter field the range of allowable values shall always be restricted using either choice or maxinclusive. This allows a maximum array size to be determined.

Parents

- property
- includedev

5.3.1 Arrays of Values

When a property that is declared to be an array contains immediate values, either directly or within sub properties, the number of values given depends on the maximum size of the array.

- one if all properties in the array have the same value.
- the same as the value of array if any values differ and array is not a parameter.
- the same as the maximum allowable value of array if any values differ and array is a parameter.

See also Arrays of Properties.

5.4 behavior

empty element

Identifies a single behavior that applies to its parent property. This is a reference to a behavior definition that must be contained in a behaviorset. In order for the reference to be unambiguous, both the behaviorset and the name of the behavior within that set must be given.

Attributes

- name (required) the name of the behavior identified.
- name.paramname (optional) binds the name attribute to the named parameter.
- set (required) the UUID of the behaviorset within which the behavior definition occurs.
- set.paramname (optional) binds the set attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- property

5.5 behaviordef

element

Contains the definition of a property behavior that is specified in one or more sections. This specifies the behavior of any property that references it.

Each behaviordef has a name that must be unique among all behaviordef elements within its containing behaviorset. For readability, the name should provide a meaningful reminder of the behavior it identifies.

Each behavior should be a refinement of one or more, more generic or abstract behaviors (see: Behavior Derivation and Refinement, refines).

The content of the behavior element is a text description of the semantics of the behavior. To allow structuring of the prose contained within behaviordef, it is divided into sections – that may be nested. At least one section is required. Each section has an optional heading and may contain text paragraphs and subsections. Processors presenting behaviors to users should use these to assist in layout. (See: Behavior Definitions).

Attributes

- name (required) must be unique within the containing behaviorset module.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. label (zero or one)
2. refines (zero or more)
3. section (one or more)

Parents

- behaviorset

5.6 behaviorset

element

A behaviorset contains a set of named behaviors. Each behavior definition is referenced by the set in which it occurs (identified by its UUID) and by its name within that set.

Attributes

- UUID (required) the unique identifier for this behaviorset
- provider (required) the organization that published this behaviorset
- date (required) the publication date of this behaviorset
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. UUIDname (zero or more)
2. label (zero or one)
3. alternatefor (zero or more)
4. extends (zero or more)
5. behaviordef (one or more)

Parents

- DDL

5.7 choice

element

Specifies one of a set of permitted values for a parameter (see: Statically Included Device Descriptions, Parametric Devices). Each value within the set is specified by a separate choice element.

If a choice element is present within a parameter element then no mininclusive, maxinclusive or refinement element shall be present in the same parameter.

The value given by a choice element shall be a legal value within the constraints of the parameter field.

Attributes

- choice.paramname (optional) binds the element content to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content

Text content containing one value that may legally be used for the specified parameter field(s).

Parents

- parameter

5.8 date

attribute

The date attribute shall contain the date the module was created. The date must be the same in any instance of the document.

Value: Shall conform to xsd:date (YYYY-MM-DD format based on [ISO-DATE])

Parents

- device
- behaviorset
- languageset

5.9 DDL

element

DDL is the root element of all DDL documents.

A DDL document shall fit the XML “document” production. The single root element shall be DDL.

DDL defines three module types: device, behaviorset and languageset (see: DDL Modules – Devices, Behaviorsets and Languagesets). Each is defined in an element of the corresponding name. A DDL element shall contain exactly one module.

Attributes

- version (required)
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children

- Exactly *one* from device *or* behaviorset *or* languageset

Parents

None: DDL is always the root element.

5.10 device

element

The device element is a DDL module (see: DDL Modules – Devices, Behaviorsets and Languagesets) and follows all the requirements for modules.

Each device definition shall contain one or more root properties (with nested sub-properties as required) each of which forms the root of a property tree.

Attributes

- UUID (required) the unique identifier for this device type
- provider (required) the organization that published this description
- date (required) the publication date of this description
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. UUIDname (zero or more)
2. parameter (zero or more)
3. label (zero or one)
4. alternatefor (zero or more)
5. extends (zero or more)
6. useprotocol (one or more)
7. Either property *or* includedev *or* propertypointer (one or more in any combination)

Parents

- DDL

5.11 extends

empty element

Identifies a module that is extended by the current one.

One module extends another when the extended module includes all the information of the original in unchanged form, but also includes further information. Rules for extension are separately defined for each module type.

Attributes

- UUID (required) the UUID the current module is extending.
- UUID.paramname (optional) binds the UUID attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- device
- behaviorset
- languageset

5.12 hd

element

The heading of a section.

Attributes

- xml:id (optional) used to reference this element using ID/IDREF matching.

Content

Text content containing the heading.

Parents

- section

5.13 includedev

element

Includedev marks the point at which a sub device is included (see: Statically Included Device Descriptions).

Attributes

- UUID (required) the UUID of the device being included.
- UUID.paramname (optional) binds the UUID attribute to the named parameter.
- array (optional) if this is an array of sub devices.
- array.paramname (optional) binds the array attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. label (zero or one)
2. protocol (zero or more) rules depend on specific protocols
3. setparam (zero or more)

Parents

- device
- property

5.14 key

attribute

The key identifying a string. Shall conform to the NCName production of the XML Namespace recommendation [XMLnames] (this is the same as the XML "Name" production but with colons disallowed).

Keys are used for lexical matching. They are case sensitive and shall be unique in the context in which they are used. See String Resources for language and key matching rules.

Parents

- label – key identifies the string to use as the label value
- string – key identifies this string within the specified language

5.15 label

element

Labels may be assigned to many elements in DDL. A label is intended for human consumption and should indicate the function of its parent element. Labels may take one of two forms. An immediate label contains the label text as its content. A referenced label contains both a string key and a languageset attribute that reference a string (or set of strings in different languages) that contains the text of the label.

A label shall have either immediate text content or both key and set attributes. If shall not have both content and attributes.

Attributes

- key (required if no text content, forbidden otherwise) the key identifying a string or strings (see: String Resources).
- key.paramname (optional) binds the key attribute to the named parameter.
- label.paramname (optional) binds the element content to the named parameter.
- set (required if no text content, forbidden otherwise) the UUID of the languageset in which the string(s) may be found.
- set.paramname (optional) binds the set attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content

Shall fit xsd:string – (required if no attributes present, forbidden otherwise) the immediate text value of the label.

Parents

- behaviordef

- behaviorset
- device
- includedev
- language
- languageset
- parameter
- property

5.16 lang

attribute

Identifies the language used in all strings in a language element.

The values of the attribute are language identifiers as defined by *Tags for the Identification of Languages* [LANG-TAG].

No two languages within a languageset shall have the same value for their lang attribute. (the value of lang must be unique within the languageset).

Value: Shall conform to xsd:language

Parents

- language

5.17 language

element

A language contains a set of string definitions in a particular language that is identified by its lang attribute. It may optionally specify an alternative language within the same languageset that should be searched if no string with a specified key is found in this language.

The altlang attribute points to another language in the same languageset by matching this language's altlang attribute with the lang attribute on another language. It is an error if there is no matching language element in the same languageset.

If no altlang attribute is present then this language must contain a string element for every key present in the languageset.

The use of altlang as a pointer can create chains of languages. It is an error if a language points to itself, whether directly or indirectly.

Attributes

- lang (required)
- altlang (optional)
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. label (zero or one)
2. string (one or more)

Parents

- languageset

5.18 languageset

element

A languageset is a container for string resources with variants in multiple languages (see: String Resources). Multilingual string resources are available for labeling the description and also for reference from within devices.

String resources are grouped into language elements.

The languageset element groups one or more language elements each of which defines replacement text for the same set of keys but in a different language. Thus given a single key, the application may choose the replacement text according to the language preferences of the user. Each key potentially has replacement text in each language declared within the languageset.

Each languageset is a DDL module and bears a UUID that is used by elements that use its string resources to identify it. The set of keys defined by a languageset is the union of the keys defined by all the strings within it.

Attributes

- UUID (required) the unique identifier for this languageset
- provider (required) the organization that published this languageset
- date (required) the publication date of this languageset
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. UUIDname (zero or more)
2. label (zero or one)
3. alternatefor (zero or more)
4. extends (zero or more)
5. language (one or more)

Parents

- DDL

5.19 maxinclusive

element

Specifies a maximum value restriction for a numeric parameter value. (see: Statically Included Device Descriptions and Parametric Devices).

If a maxinclusive element is present within a parameter element then no choice or refinement element shall be present in the same parameter.

maxinclusive shall only be used for parameter fields that have a numeric value.

The value given by a maxinclusive element shall be a legal value within the constraints of the parameter fields to which the parameter applies and also shall conform to any restrictions imposed on the value of the fields by other elements either in the included device or in the same parameter element.

A maxinclusive value shall not be less than the value of any mininclusive present.

Attributes

- maxinclusive.paramname (optional) binds the element content to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content: Text content containing a numeric value in the same format as the field being restricted.

Parents

- parameter

5.20 mininclusive

element

Specifies a minimum value restriction for a numeric parameter field. (see: Statically Included Device Descriptions and Parametric Devices).

If a mininclusive element is present within a parameter element then no choice or refinement element shall be present in the same parameter.

mininclusive shall only be used for parameter fields that have a numeric value.

The value given by a mininclusive element shall be a legal value within the constraints of the parameter fields to which the parameter applies and also shall conform to any restrictions imposed on the value of the fields by other elements either in the included device or in the same parameter element.

A mininclusive value shall not be greater than the value of any maxinclusive present.

Attributes

- mininclusive.paramname (optional) binds the element content to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content: Text content containing a numeric value in the same format as the field being restricted.

Parents

- parameter

5.21 name

attribute

A name that matches the NCName production of the XML Namespace recommendation [XMLnames] (this is the same as the XML “Name” production but with colons disallowed).

Names are used in various elements and special restrictions apply to some.

Names are used for lexical matching. They are case sensitive and shall be unique in the context in which they are used. e.g. a behavior name must be unique within a behaviorset, but may match the name of a behavior from another set or the name of a string that exists in a different context.

Value: Shall conform to xsd:NCName

Parents

- behavior
- behaviordef

- parameter
- protocol
- refines
- setparam
- useprotocol
- UUIDname – length of name in a UUIDname shall be 32 characters or less.

5.22 p

element

A paragraph of text (in a behavior definition). In presenting behaviors, processors may (and should) freely "fold" whitespace and word-wrap text in paragraphs to suit the presentation medium, unless the paragraph carries an xml:space attribute with the value "preserve", in which case the paragraph should be presented as is with no formatting.

Attributes

- xml:space (optional)
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content: Text content is the text of the paragraph.

Parents

- section

5.23 parameter

element

This element declares a parameter of a device for which a substitution value may be provided when the device is included as a template (see: Statically Included Device Descriptions and Parametric Devices). When the value for the parameter is provided on inclusion, it is applied to all the included fields that are bound to the parameter name using .paramname attributes.

The parameter element declares the parameter near the start of a device and provides a name for it. It may optionally contain choice, refinement, mininclusive or maxinclusive elements restricting the range of values the parameter may take.

Attributes

- name (required) the name of the parameter
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order):

1. label (zero or one)
2. Mutually exclusive options:
   either choice (zero or more) *or* refinement (zero or more) *or* (in order):
   1. mininclusive (zero or one)
   2. maxinclusive (zero or one)

Parents

- device

5.24 .paramname

attribute

This is not a single attribute but is a suffix for an attribute used to bind a field to a parameter – see: Parametric Devices. The first part of the attribute name is the name of the attribute or element to be bound to the parameter. For example, attribute array can be parametrized by declaring a array.paramname attribute on the same element. If the field to be bound is the content of an element then the first part of the .paramname attribute shall be the same as the element name.

Value: Shall conform to xsd:NCName

Parents

- alternatefor (UUID.paramname)
- behavior (name.paramname, set.paramname)
- choice (choice.paramname)
- extends (UUID.paramname)
- includedev (UUID.paramname, array.paramname)
- label (label.paramname, key.paramname, set.paramname)
- maxinclusive (maxinclusive.paramname)
- mininclusive (mininclusive.paramname)
- property (array.paramname, valuetype.paramname, sharedefine.paramname)
- propertypointer (ref.paramname)
- protocol (name.paramname)
- setparam (setparam.paramname)
- refinement (refinement.paramname)
- useprotocol (name.paramname)
- value (value.paramname, type.paramname)

Note

Additional parametrizable fields may be specified within protocol elements whose content is separately declared.

5.25 property

element

The property is the basic building block of device structure. Each property may itself have multiple properties nested within it. Every property has zero or one value.

Parents

- device
- property

Attributes

- array (optional) if not present defaults to 1 (a single value).
- array.paramname (optional) binds the array attribute to the named parameter.
- valuetype (required). Permitted values are:

    - NULL
    - network
    - implied
    - immediate
- valuetype.paramname (optional) binds the valuetype attribute to the named parameter. Note parametrizing valuetype is of limited use since changing it in most cases requires changes to the descendent XML structure that cannot be done using parameters.
- sharedefine (optional) if set “true” or “arraycommon” declares that this property is a defining declaration that may be referenced from elsewhere as a shared property. When set to “true” or “arraycommon” the property must also have a valid xml:id attribute.
- sharedefine.paramname binds the sharedefine attribute to the named parameter.
- xml:id (optional, required for defining declarations of shared properties) used to reference this element using ID/IDREF matching.

Content: depends on the valuetype attribute:

5.25.1 valuetype="NULL"

Children (in order):

1. label (optional)
2. behavior (one or more)
3. protocol (optional).
4. *Either* property *or* includedev *or* propertypointer (zero or more in any combination)

5.25.2 valuetype="network"

Children (in order):

1. label (optional)
2. behavior (one or more)
3. protocol (required).
4. *Either* property *or* includedev *or* propertypointer (zero or more in any combination)

5.25.3 valuetype="implied"

Children (in order):

1. label (optional)
2. behavior (one or more)
3. protocol (optional).
4. *Either* property *or* includedev *or* propertypointer (zero or more in any combination)

5.25.4 valuetype="immediate"

Children (in order):

1. label (optional)
2. behavior (one or more)
3. protocol (optional).
4. value (one or more)
5. *Either* property *or* includedev *or* propertypointer (zero or more in any combination)

5.26 propertypointer

empty element

A propertypointer is a shared property whose defining declaration is elsewhere. It is permitted in any place that a property is valid and represents the same property as the one to which it points. This allows the same property to be child of multiple parents that are in different places in the device. It is permitted for a propertypointer to reference another propertypointer provided that such a chain of references resolves to a full property declaration eventually.

The target of the ref attribute of a propertypointer shall be a property that has a sharedefine attribute whose value is either “true” or “arraycommon”.

Parents

- device
- property

Attributes

- ref (required). Identifies the defining declaration of this property.
- ref.paramname (optional) binds the ref attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

5.27 protocol

element

Protocol elements contain whatever information is required to enable processors to access networked property values. Their content is of necessity dependent on the access protocol(s) used.

Protocol elements may optionally be used within properties that do not have network values and within includedev to provide other protocol information as necessary – this is usually to specify rules applying to child properties. It is illegal for a protocol element to directly identify a networked value for a property unless that property has been declared with a value type of “network”.

Definition of the content of protocol is a large part of interfacing DDL to a particular access protocol. Any protocol using DDL must specify how this is done.

For validation, a more complete schema definition for protocol tailored to an individual access protocol is often desirable. However, should definitions be written using multiple protocols, those definitions might prevent validation.

Attributes

- name (required) shall match an ESTA defined protocol name [ESTA-IDs]. Shall match a protocol identified by a useprotocol element in the same device.
- name.paramname (optional) binds the name attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children: any – depends on definitions supplied with specific protocols.

Parents

- property
- includedev

5.28 provider

attribute

The provider attribute shall unambiguously indicate the organization that created and maintains the definition. This is not necessarily the same as the organization that produces the equipment (for example a definition may be reused). The preferred form for the provider attribute is a URL that clearly identifies the organization and the appropriate department or section.

Value: Should conform to xsd:anyURI

Parents

- device
- behaviorset
- languageset

5.29 ref

attribute

Contains a reference that points to a property.

Value: Shall conform to the syntax and rules for extended references and references between arrays (see: Use of XML Identifiers to Reference Device Model Features).

Parents

- propertypointer

5.30 refinement

element

This establishes a restriction on the permissible values for a parameter (see: Statically Included Device Descriptions, Parametric Devices). It shall only be used when the parametrized field identified is a behavior name. refinement specifies a base behavior. Any value provided for the parameter field (including ang default behavior specified in-place) shall be a refinement of this base behavior.

If a refinement element is present within a parameter element then no choice, shall be present in the same parameter. (behavior name attributes are not numeric so mininclusive and maxinclusive cannot apply).

The value given by a refinement element shall be a legal value within the constraints of the parameter field and also shall conform to any restrictions imposed on the value of the field by other elements either in the included device or in the same parameter element.

Attributes

- refinement.paramname (optional) binds the element content to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content. Text content shall conform to xsd:NCName and shall match a behavior name within the behaviorset identified by the target behavior element.

Parents

- parameter

5.31 refines

empty element

Every behavior may be a refinement of one or more existing definitions (see: Behavior Derivation and Refinement). refines identifies a single behavior definition that the current one derives from. The refines element is a reference to a behaviordef using exactly the same syntax as the behavior element. In order for the reference to be unambiguous, both the behaviorset and the name of the behavior within that set must be given.

Attributes

- name (required) the name of the behavior identified
- set (required) the UUID of the behaviorset within which the behavior definition occurs.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- behaviordef

5.32 section

element

Behavior definitions are textual descriptions that can be verbose. To allow some structuring of the description it is contained in sections that may be nested recursively, Each section has an optional heading and contains any number of freely intermixed sections and paragraphs.

Attributes

- xml:id (optional) used to reference this element using ID/IDREF matching.

Children (in order)

1. hd (optional)
2. one or more of: section and/or p

Parents

- behaviordef
- section

5.33 set

attribute

Identifies (by UUID) a behaviorset or languageset. The format and rules for the value are identical to UUID.

Parents

- refines – set must identify a behaviorset
- behavior – set must identify a behaviorset
- label – set must identify a languageset

5.34 setparam

element

Specifies the value a parameter is to take, when a sub device is included in a description (see: Parametric Devices).

Attributes

- name (required) the name of the parameter – shall match the name of a parameter element within the device identified by the includedev containing this setparam element.
- setparam.paramname (optional) binds the element content to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content. Text content is the value to be substituted for the named parameter in this instance of the template device.

Parents

- includedev

5.35 sharedefine

attribute

Used to identify the defining declaration of a property that is or may be shared(see: Shared Properties).

Value

Shall conform to xsd:boolean. If *true*, then the property must also carry an xml:id attribute.

Legal values are:

false
: The property cannot be referenced by a propertypointer as a defining declaration

true
: This is a defining declaration of a shared property and may be the target of propertypointer declarations elsewhere in the device. The property must also carry an xml:id attribute.

arraycommon
: This is a single property despite any array declarations on its parents or ancestors(see: Special Case – Property Shared Across an Array). Anywhere that this property is contained within an array, it is is common to all items of that array. This may also be a defining declaration of a shared property and may be the target of propertypointer declarations elsewhere in the device. The property must also carry an xml:id attribute.

Default: If no sharedefine attribute is present then the default value false shall be used.

Parents

- property

5.36 string

element

A single text string resource. It is located by its key attribute.

The key attribute shall be unique within the enclosing language.

Attributes

- key (required) the key identifying this string.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children: Text content is the text of string in the language of the parent language element.

Parents

- language

5.37 type

attribute

Specifies the type of a property value – signals to a processor how to parse the text representation.

Value

Legal values are:

uint
: Value shall conform to xsd:unsignedInt. The value represents an integer within the range 0..4294967295.

sint
: Value shall conform to xsd:int. The value represents an integer within the range -2147483648..+2147483647.

float
: Value shall conform to xsd:double. The value represents an IEEE double precision floating point number.

string
: The property value should be parsed or processed as a (Unicode) text string.

object
: The property value is an arbitrary sized binary object that is not a regular integer. Immediate values shall be represented as a sequence of octets with each octet being represented as a pair of hexadecimal nibbles with the most significant nibble first. For readability, spaces, periods, commas or hyphens may be included between octets. Where a property is of this type, the behavior description must specify any byte ordering and formatting conventions necessary to interpret the value. It is recommended that network byte order be used unless there are overriding reasons to use other conventions.

Example 14: Immediate values of type "object"

```
<property
  type="object"
  value="1d2d5369 3f1809a6 07dcbb18 510fe564 8b65104d"
  ...
/>
<property
  type="object"
  value="b6.6b.b9.93.2f.44"
  ...
```

```
/>
```

Parents

- value

5.38 useprotocol

empty element

Identifies a protocol that may be used to access a device. It is illegal for a protocol element within a description to name a protocol that has not been declared in a useprotocol element within the same device.

Attributes

- name (required) shall match an ESTA defined protocol name [ESTA-IDs].
- name.paramname (optional) binds the name attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- device

5.39 UUID

attribute

A UUID attribute shall either contain a literal UUID value or a UUID name that has been assigned using the UUIDname element and that is in scope.

Literal values shall be written in the format given in [UUID]. All hexadecimal letters shall be in lower case but parsers should accept either lower case or upper case.

Note

Module identifiers are also referred to as DCIDs in this specification and elsewhere. The attribute name is always UUID.

Note

Applications may distinguish the format used by the length of the string. A literal value always has 36 characters while a name is required to be less than this.

Example 15: Examples of UUID values

```
UUID="3f399f5e-ad01-11d9-8525-000d613667e2"
UUID="xyz" <!-- named UUID UUIDname -->
```

Value: Shall fit xsd:NMTOKEN with maxLength = 36.

Parents

- device (must be literal UUID since UUIDname cannot be in scope).
- behaviorset (must be literal UUID since UUIDname cannot be in scope).
- includedev (may be explicit UUID or UUID name).
- languageset (must be literal UUID since UUIDname cannot be in scope).
- UUIDname (must be literal UUID – this element defines a corresponding UUID name).
- alternatefor (may be explicit UUID or UUID name).
- extends (may be explicit UUID or UUID name).

5.40 UUIDname

empty element

UUIDs are unwieldy for reasons of size and readability. A set of UUIDname elements at the start of a module, allow short readable names to be assigned to UUIDs. Each UUIDname associates a single name with a UUID. The scope of a UUIDname is the entire module in which it occurs, with the exception of the attribute values of the root element of the module.

It is illegal for two UUIDname elements within the same module to have the same value for their name attribute.

Attributes

- UUID (required) is the UUID to be assigned the name. Shall be specified explicitly Named UUIDs are not permitted.
- name (required) shall confirm to xsd:NCname with maxlength = 32.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- device
- behaviorset
- languageset

5.41 value

element

Contains the value of an immediate property (one whose value is declared within the description). If the property is within an array, it may contain multiple value elements – see Immediate Values for Array Properties.

Attributes

- type (required) the type of value contained
- type.paramname (optional) binds the type attribute to the named parameter.
- value.paramname (optional) binds the element content to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Content. Text content is the textual representation of the value of the parent property. See type for format.

Parents

- property

5.42 valuetype

attribute

Specifies the value type of a property (see: Four Locations for Property Value).

Value

Legal values are:

NULL
: The property has no value – only sub properties

immediate
: The value of the property is given within the description in a value

network
: The property value is accessible over the network. One or more protocol elements contain the details of how to access it.

implied
: The property has a hidden value that is not directly accessible. In most cases this value is driven by other properties (see: Meaning of Property Values – Driven Properties) and in some cases the value may be inferred. For bidirectional protocols, use of implied properties is discouraged – it is generally better to make it readable. For unidirectional protocols such as DMX512 [DMX512] implied values are often inevitable.

Parents

- property

5.43 version

attribute

This is the DDL version. The version number "1.1" shall be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.1" if it does not conform to this version of this specification. This construct is provided, as a means to allow the possibility of automatic version recognition, should it become necessary. Processors should report an error if they receive documents labeled with versions they do not support.

Value shall be "1.1"

Parents

- DDL

5.44 xml:space

attribute

Follows the semantics and rules defined in the XML specification [XML]. See p element for details.

Value

Legal values are:

default
: default behavior is to "fold" whitespace and wrap words

preserve
: text is preformatted and should be presented "as is"

Parents

- p

5.45 xml:id

attribute

Follows the semantics and rules defined in [xml-id] specification. See Use of XML Identifiers to Reference Device Model Features.

Value. The value shall match the NCName definition in the XML namespace specification [XMLnames].

Parents. xml:id may be used on any element within DDL.

6 Interface to the Access Protocol

Protocols vary wildly in their notations, and capabilities. In order for DDL to describe real devices using any given protocol, an interface needs to be defined between DDL and that protocol. This section gives guidelines for what needs to be defined in order to apply DDL to a particular protocol. A concrete example is also given in [DMP] that defines the interface to a real protocol.

All protocol specific information must be specified within protocol elements that identify the protocol they refer to. The protocol element must be customized for each protocol to be interfaced.

6.1 Access to Network Properties

6.1.1 Network Property Values

Property values may either be constant immediate values embedded in the description or may be accessed via the network or data link. In the latter case a protocol element shall be provided that must provide sufficient information for a controller using that protocol to access the value. This includes not just network location of the property or the message type(s) needed to access it, but frequently such things as “on the wire” data size and representation and read/write characteristics.

6.1.2 Network Data Type and Size

In many protocols data representation and encoding is frequently best expressed as a behavior, however, a minimal processor should be able to read or write and pass on property values without knowledge of specific behaviors. Therefore, the networked value reference elements must specify any items such as the size, read/write accessibility and so on that are required to access and handle the data without necessarily interpreting it.

6.2 Necessary Definitions and Restrictions

All items in this section must be customized or defined for a protocol interface.

6.2.1 Protocol Declaration

A protocol to be used with DDL needs to have a key-name registered with ESTA.

Format of protocol identifiers and processes for registration are given in [ESTA-IDs].

6.2.2 Network Property Access Mappings

The protocol interface must define how the protocol is used with the information given in a declaration to examine and/or modify the values of a network property.

6.3 Behaviors

While the behavior mechanism is part of DDL, specific behaviors are not. A base set of behaviors are provided for [DMP] that may be reused or adapted as necessary with other protocols.

Appendix A. Generic DTD DDL

This DTD (refer to [XML]) is provided for reference purposes. See DTD, Schemas and Validation for discussion of schemas and validation. Note that the protocol element is declared with content ANY, but that any elements for specific protocols contained within it will nevertheless need to be declared before this DTD can be used.

The public identifier for this DTD shall be:

"-//ESTA//DTD Device Description Language 1.1//EN"

```
<!--
  DTD for Device Descritpion Language
  no specific protocol declared
-->
<!ELEMENT DDL ((behaviorset | device | languageset))>
<!ATTLIST DDL
  version ( 1.1 ) #REQUIRED
  xml:id ID #IMPLIED
>
<!-- common module content -->
<!ELEMENT label (#PCDATA)>
<!ATTLIST label
  label.paramname NMTOKEN #IMPLIED
  set NMTOKEN #IMPLIED
  set.paramname NMTOKEN #IMPLIED
  key NMTOKEN #IMPLIED
  key.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT alternatefor EMPTY>
<!ATTLIST alternatefor
  UUID NMTOKEN #REQUIRED
  UUID.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>

<!ELEMENT extends EMPTY>
<!ATTLIST extends
  UUID NMTOKEN #REQUIRED
  UUID.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT UUIDname EMPTY>
<!ATTLIST UUIDname
  name NMTOKEN #REQUIRED
  UUID NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- languageset module -->
<!ELEMENT languageset (
  UUIDname*, label?,
  alternatefor*,
```

```
  extends*,
  language+
)>
<!ATTLIST languageset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- languageset content -->
<!ELEMENT language (label?, string*)>
<!ATTLIST language
  lang CDATA #REQUIRED
  altlang CDATA #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT string (#PCDATA)>
<!ATTLIST string
  key NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- behaviorset module -->
<!ELEMENT behaviorset (
  UUIDname*, label?,
  alternatefor*,
  extends*,
  behaviordef+
)>
<!ATTLIST behaviorset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- behaviorset content -->
<!ELEMENT behaviordef (label?, refines*, section+)>
<!ATTLIST behaviordef
  name NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!ELEMENT refines EMPTY>
<!ATTLIST refines
  set NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!ELEMENT section (hd?, (section | p)+)>
<!ATTLIST section
  xml:id ID #IMPLIED
>
<!ELEMENT hd (#PCDATA)>
<!ATTLIST hd
```

```
  xml:id ID #IMPLIED
>
<!ELEMENT p (#PCDATA)>
<!ATTLIST p
  xml:space (default | preserve) 'default'
  xml:id ID #IMPLIED
>
<!-- device module -->
<!ELEMENT device (
  UUIDname*, parameter*, label?,
  alternatefor*,
  extends*,
  useprotocol+,
  (property | propertypointer | includedev)+
)>
<!ATTLIST device
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- parameter declarations -->
<!ELEMENT parameter (
  label?,
  ( choice*
    | refinement*
    | ( mininclusive?, maxinclusive?)
  )
)>
<!ATTLIST parameter
  name NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!--parameter restrictions -->
<!ELEMENT choice (#PCDATA)>
<!ATTLIST choice
  choice.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT mininclusive (#PCDATA)>
<!ATTLIST mininclusive
  mininclusive.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT maxinclusive (#PCDATA)>
<!ATTLIST maxinclusive
  maxinclusive.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT refinement (#PCDATA)>
<!ATTLIST refinement
  refinement.paramname NMTOKEN #IMPLIED
```

```
  xml:id ID #IMPLIED
>
<!-- see below for useprotocol and other protocol dependent content -->
<!-- properties -->
<!ELEMENT property (
  label?,
  behavior+,
  value*,
  protocol*,
  (property | propertypointer | includedev)*
)>
<!ATTLIST property
  array CDATA #IMPLIED
  array.paramname NMTOKEN #IMPLIED
  valuetype (NULL | immediate | implied | network | common) #IMPLIED
  valuetype.paramname NMTOKEN #IMPLIED
  sharedefine (false | true | arraycommon) "false"
  sharedefine.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT behavior EMPTY>
<!ATTLIST behavior
  set NMTOKEN #REQUIRED
  set.paramname NMTOKEN #IMPLIED
  name NMTOKEN #REQUIRED
  name.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value
  value.paramname NMTOKEN #IMPLIED
  type (uint | sint | float | string | object) #REQUIRED
  type.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT propertypointer EMPTY>
<!ATTLIST propertypointer
  ref CDATA #REQUIRED
  ref.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!-- included devices and parameter instantiation -->
<!ELEMENT includedev (label?, protocol*, setparam*)>
<!ATTLIST includedev
  UUID NMTOKEN #REQUIRED
  UUID.paramname NMTOKEN #IMPLIED
  array CDATA #IMPLIED
  array.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT setparam (#PCDATA)>
<!ATTLIST setparam
```

```
  name NMTOKEN #REQUIRED
  setparam.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!-- Protocol dependent section -->
<!--
  The following elements have generic content models here
  They may be specialized as described for specific protocols
-->
<!ELEMENT useprotocol EMPTY>
<!ATTLIST useprotocol
  name CDATA #REQUIRED
  name.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!--
  Note:
  The name attribute shall be an ESTA registered name.
-->
<!ELEMENT protocol ANY >
<!ATTLIST protocol
  name CDATA #REQUIRED
  name.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!--
  Notes:
  The name attribute shall match the ESTA registered name
  of a   protocol declared in a preceding useprotocol element.
  Content must conform to rules specified for the protocol
  identified by the name attribute.
  No further attributes permitted.
-->
<!-- End of protocol dependent section -->
```

Appendix B. DDL Interface to DMP

Device Management Protocol (DMP) and Device Description Language (DDL) are part of the ESTA ACN suite of protocols for control of entertainment technology equipment. They are fully described in the DMP specification [DMP] and the DDL Specification [DDL].

B.1 Relationship Between DDL Devices and DMP Components

Each DMP component (see [Arch]) that exposes properties is defined in DMP to be a device – in DMP there can only be one set of properties within a component and the factoring of these properties into multiple DDL devices is a function of DDL that is not visible in DMP. In DDL terms this is a single appliance. The UUID [UUID] of its single root device is expected to be made available through whatever ACN Discovery method is applicable. The appliance may be described by any number of additional child devices and descendant devices. Thus given the root device class, the structure of the entire component can be examined by finding the child devices of the root device and then their children and so on.

B.2 Message Mappings

The values of network property elements in DDL correspond directly to DMP values that are accessed using the various forms of Get_Property, Set_Property and Event messages of DMP.

B.3 DMP Property Access

The characteristics necessary to access DMP properties are all specified within protocol elements using propref_DMP and childrule_DMP. propref_DMP specifies the location and access characteristics of a DMP property while childrule_DMP does not specify any DMP property itself but sets the location origin which is used by the relative address syntax for sub properties and included devices.

B.3.1 Addressing

DMP properties are single values that have a one-to-one correspondence with DDL Network Accessed properties (see: Four Locations for Property Value). A DMP property is specified using a single scalar address range (0..4294967295).

A DMP address may be specified in DDL in a relative or absolute form.

B.3.1.4 Relative Addressing and Location Origin

To provide the address of a DMP property, a relative addressing scheme is available in DDL and is the preferred address syntax. DMP addresses or network properties are expressed as signed offsets relative to a location origin which is defined for each DDL property. For details see loc and childrule_DMP.

The use of relative addressing means that any property tree or subtree in a device defines a pattern of locations for its properties that are relative to the location of the root of the subtree. This allows a subtree to be repeated elsewhere in the address space with a different location origin but no change at all to its description.

Note

The expression of DMP addresses in relative form in DDL does not affect the way the addresses must be specified in DMP messages and is not connected with any specific addressing modes provided by DMP. Any address specified in DDL must be resolved to an absolute address before it can be used in generating a DMP message.

B.3.1.5 Absolute Addressing

It is possible to specify DMP addresses (and location origins) in absolute form using abs but this is not recommended in normal circumstances because it restricts re-use of devices by inclusion or reference.

B.3.1.6 Non-network Properties

Properties that do not have a network value shall not contain a DMP address specification. However they may contain childrule_DMP to specify address characteristics for their children.

B.3.2 Property Arrays

In accordance with Arrays of Properties any property may be declared as an array. The DMP addresses of array properties shall be specified as follows:

For any property that has an array attribute with a value greater than one (any array property):

- If the array property has a DMP network value. The propref_DMP element specifying the DMP address shall have an inc attribute specifying how the DMP property address increments. This increment applies to iteration over the array specified on this property only and not to iteration over any array inherited from ancestor properties.
- If the array property has one or more descendant properties that have DMP network values, or if the array property includes devices (either statically included or attached by reference) that have or could have DMP network accessible properties; it shall contain a childrule_DMP element (within a protocol element) that includes an inc attribute specifying how the addresses of all child DMP properties increment. The increment specified by this childrule_DMP applies to iteration over the array count specified by the immediate property only (by the property element that is the grandparent of the childrule_DMP elelement). It does not apply to any other index of a multidimensional array.

Any property within an array for which `sharedefine="arraycommon"` is declared, is a single property (see: Special Case – Property Shared Across an Array). Its address is determined solely by the relevant loc and abs attriubtes of its propref_DMP element and the childrule_DMP elements of its ancestors without reference to their inc attributes. This is identical to the address its first element would have if it were left as an array.

B.3.3 Accessibility

DMP properties may be accessed using Get_Property and Set_Property and Event. However, it is not required that a particular property be accessible using all three messages. Attributes read and write and event indicate whether a property may be accessed using Get_Property and/or Set_Property and/or Event respectively. Note that event publishing may require auxiliary properties to be manipulated (e.g. to set publishing frequency) and these are described in behaviors.

B.3.4 Property size and variable size properties

DMP provides two transfer methods for property values: fixed length and variable length.

Variable length transfers encode the value length within the message and can vary in length from one message to another depending on the value being transferred. This carries an overhead in order to encode the length. Variable length transfers are suitable for text strings and other items whose intrinsic size varies.

Fixed length transfers do not encode the length of the value so this must be known. Fixed length transfers are suitable for items such as integers, floating point quantities, or objects that have a fixed format.

The value of any particular DMP property must always be transferred using the same method and if fixed length, using the same number of octets irrespective of the value, message or any other factor.

The DDL description of a property shall specify which transfer method must be used for that property and for fixed length transfers, the size (the number of octets used). These are specified using the size and varsize attributes.

B.3.5 Additional Characteristics of DMP Properties

With just the information in DMP protocol elements, an application can store and retransmit properties but cannot generate or modify them in any way since they may use different internal representations (e.g. floating point vs integer vs unordered bitmap).

Any higher interpretation placed on a property value must be discovered from its behavior – behaviors for common data types are defined in the core ACN behavior set.

B.3.6 Byte Ordering

While specific encodings are specified and described in DDL behaviors, it is forbidden to create behaviors that specify transmission of property values in non network byte order except in the exceptional case where an existing encoding is used for a specific function and where that encoding is standardized by a recognized standards body for use across platforms of both byte orders.

B.4 DMP Element and Attribute Reference

B.4.1 useprotocol (DMP devices)

empty element

Identifies a protocol that may be used to access a device. For any description of DMP access to a device there shall be a useprotocol element with `name="ESTA.DMP"`. There may be additional useprotocol elements if the description also applies for other access protocols.

Attributes

- name (required) shall have the value “ESTA.DMP”.
- name.paramname (optional) binds the name attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- device

B.4.2 protocol (DMP devices)

element

In DMP a protocol element shall have the following content.

Attributes

- name (required) shall have the value “ESTA.DMP”
- name.paramname (optional) binds the name attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Children

- propref_DMP (required if parent is property with `valuetype="network"`, forbidden otherwise) specifies a network property.
- childrule_DMP (optional) specifies location origin of children of this property or of the root

properties of an included device.

Parents

- property
- includedev

B.4.3 propref_DMP

empty element

Specifies the location and characteristics of a DMP property or property array.

Attributes

- loc (required)
- loc.paramname (optional) binds the loc attribute to the named parameter.
- size (required)
- size.paramname (optional) binds the size attribute to the named parameter.
- abs (optional)
- abs.paramname (optional) binds the abs attribute to the named parameter.
- read (optional)
- read.paramname (optional) binds the read attribute to the named parameter.
- write (optional)
- write.paramname (optional) binds the write attribute to the named parameter.
- event (optional)
- event.paramname (optional) binds the event attribute to the named parameter.
- varsize (optional)
- varsize.paramname (optional) binds the varsize attribute to the named parameter.
- inc (optional)
- inc.paramname (optional) binds the inc attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- protocol

B.4.4 childrule_DMP

empty element

Specifies the location origin for all child properties of the current property or included device (includedev). The location origin is used to specify DMP addresses using a relative form – see loc.

If a property does not contain a childrule_DMP specification (inside a protocol element), then its children (both property and includedev) shall inherit its own location origin unchanged.

The location origin for the root properties in a device that is attached by reference (see: Sub Devices Attached by Reference) shall be received from the device-reference property that anchors the sub-device within the parent device as though they were direct children of that property.

The location origin for the root properties in a device that is statically included (see: Statically Included Device Descriptions) may be specified explicitly using childrule_DMP (inside protocol) in the includedev element. If not specified explicitly the location origin shall be the same as would apply for a property at the point of inclusion.

The location origin for the root properties of the root device in an appliance shall be 0.

Attributes

- loc (optional)
- loc.paramname (optional) binds the loc attribute to the named parameter.
- abs (optional if loc is present, forbidden otherwise)
- abs.paramname (optional) binds the abs attribute to the named parameter.
- inc (optional)
- inc.paramname (optional) binds the inc attribute to the named parameter.
- xml:id (optional) used to reference this element using ID/IDREF matching.

Parents

- protocol

B.4.5 loc

attribute

Specifies a DMP location (address). DMP addresses are in the range 0..4294967295. This address may be either a property address (propref_DMP) or the location origin for child properties (childrule_DMP).

If abs is present on the same element and is true then this is an absolute DMP address and must fall within the legal range.

If abs is not present or is false then the DMP address is given in a relative form and is a signed offset of the DMP address relative to the location origin of the property or device (see: Relative Addressing and Location Origin).

Value

***if* abs is true**

loc shall conform to xsd:unsignedInt

else

loc shall conform to xsd:int

Parents

- propref_DMP
- childrule_DMP

B.4.6 abs

attribute

If true abs specifies that the DMP location (address) given by loc is in an absolute form. If false or absent, then loc is in its relative form.

Value. Shall conform to xsd:boolean.

Default. If no abs attribute is present then the default value false shall be used.

Parents

- propref_DMP
- childrule_DMP

B.4.7 inc

attribute

When the property is a part of an array, inc specifies the increment in DMP address between each element.

Value. Shall conform to xsd:unsignedShort

Default. No default is provided. inc shall be explicitly included wherever an increment must be specified according to the rules in Property Arrays.

Parents

- propref_DMP
- childrule_DMP

B.4.8 read

attribute

If true the DMP property is readable (using a Get_Property message). If false or not present, the property may not be read.

Value. Shall conform to xsd:boolean.

Default. If no read attribute is present then the default value false shall be used.

Parents

- propref_DMP

B.4.9 write

attribute

If true the DMP property is writable (using a Set_Property message). If false or not present, the property may not be written.

Value. Shall conform to xsd:boolean.

Default. If no write attribute is present then the default value false shall be used.

Parents

- propref_DMP

B.4.10 event

attribute

If true the DMP property is capable of generating events (set up using a Subscribe_Event message). If false or not present, the property cannot generate events.

Note

A number of behaviors are defined in the ACN base behaviorset for properties that modify or control the generation of events and further behaviors may be defined. Simply sending a subscribe message may not be sufficient to actually receive events from a property.

Value. Shall conform to xsd:boolean.

Default. If no event attribute is present then the default value false shall be used.

Parents

- propref_DMP

B.4.11 varsize

attribute

If true the DMP property value is transferred using DMP's variable length encoding. If false or not present, the value is always transferred in the fixed number of octets given by size (see: Property size and variable size properties).

Value. Shall conform to xsd:boolean.

Default. If no varsize attribute is present then the default value false shall be used.

Parents

- propref_DMP

B.4.12 size

attribute

Specifies the number of octets DMP uses to transfer the property value. If varsize is true then this is the maximum size of the data itself excluding the size indicator (see: Property size and variable size properties).

Value. Shall conform to xsd:unsignedShort

Parents

- propref_DMP

B.5 Examples

These assume that an (non-existent) example behaviorset “f3347bac-3e20-437d-a69e-019f37a71288” has been assigned the name “foo.bset” in the parent device, and that the ACN base behaviorset “71576eac-e94a-11dc-b664-0017316c497d” has been assigned the name “acnbase.bset” (using UUIDname):

```
<device ...>
  <UUIDname UUID="f3347bac-3e20-437d-a69e-019f37a71288" name="foo.bset"/>
  <UUIDname UUID="71576eac-e94a-11dc-b664-0017316c497d"
name="acncore.bset"/>
  ...
</device>
```

B.5.1 Simple Property

Define a property that contains a 16 bit signed integer, that can be read and written at DMP address “10”.

Example B.5.1: DDL for a Simple Property

```
<property valuetype="network">
  <behavior set="foo" name="bar"/>
  <protocol name="ESTA.DMP">
```

```
    <propref_DMP
      loc="10"
      abs="true"
      size="2"
      read="true"
      write="true"
    />
  </protocol>
</property>
```

B.5.2 Array of Simple Properties

Define an array of 512 properties that are 8 bit unsigned integers, that can be written but not read, and live at 0—511 within the DMP address space:

Example B.5.2: DDL for Array of Simple Properties

```
<property valuetype="network" array="512">
  <behavior set="foo" name="bar"/>
  <protocol name="ESTA.DMP">
    <propref_DMP
      loc="0"
      abs="true"
      size="1"
      write="true"
      inc="1"
    />
  </protocol>
</property>
```

B.5.3 Included Device

Define an included sub device that has DCID "38763d0c-b0f9-11d9-8615-000d613667e2" and whose root properties have a location origin offset by 100 relative to the current location origin. (Current location origin + 100 is the location origin for the root properties within the sub-device.)

Example B.5.3: DDL for a Direct Device Reference

```
<includedev UUID="38763d0c-b0f9-11d9-8615-000d613667e2">
  <protocol name="ESTA.DMP">
    <childrule_DMP loc="100"/>
  </protocol>
</includedev>
```

B.5.4 Subdevice Attached by Reference

Define an attached sub device that has DCID "38763d0c-b0f9-11d9-8615-000d613667e2" and whose root properties have a location origin offset by 100 relative to the current location origin. (Current location origin + 100 is the location origin for the root properties within the sub-device.)

This example achieves the same logical structure as the previous example and is included for informative purposes and as an introduction to attachment. In general where the attached device is statically defined (using an immediate value) attachment by reference is less flexible than static inclusion.

Example B.5.4: DDL for a Direct Device Reference

```
<property valuetype="immediate">
```

```
  <behavior set="acnbase.bset" name="deviceRef"/>
  <value type="object">38763d0c-b0f9-11d9-8615-000d613667e2</value>
  <protocol name="ESTA.DMP">
    <childrule_DMP loc="100" />
  </protocol>
</property>
```

B.5.5 External Device References

Define a reference to a sub device that is not predefined. The DCID of the sub device is at address 20 while the location origin of the sub devices properties is DMP address 1000. Because address 20 allows events, the controller can be notified should the type of the sub-device be changed dynamically.

Example B.5.5: DDL for an External Device Reference

```
<property valuetype="network">
  <behavior name="deviceRef" set="acnbase.bset"/>
  <protocol name="ESTA.DMP">
    <propref_DMP
      loc="20"
      size="16"
      read="true"
      event="true"
    />
    <childrule_DMP
      loc="1000"
      abs="true"
    />
  </protocol>
</property>
```

B.5.6 Array of Sub Devices

Define a reference to an array of 96 identical sub devices that have the DCID “38763d0c-b0f9-11d9-8615-000d613667e2” with the first sub device's properties addresses starting at 500 and subsequent sub devices having properties starting at 1500, 2500, 3500 etc. through the DMP address space.

The “foo” parameter for all subdevices is set to “99”.

Example B.5.6: DDL for an Array of Direct Device References

```
<property valuetype="NULL" array="96">
  <behavior name="group" set="acnbase.bset"/>
  <includedev
    UUID="38763d0c-b0f9-11d9-8615-000d613667e2"
    array="96"
  >
    <protocol name="ESTA.DMP">
      <childrule_DMP
        loc="500"
        abs="true"
        inc="1000"
      />
    </protocol>
    <setparam name="foo">99</setparam>
  </includedev>
```

```
</property>
```

B.5.7 Array of External Device References

Define a reference to an array of 16 sub devices that are not predefined. The DCIDs of the sub devices live in an array from addresses 11-26 while the sub devices themselves have properties starting at addresses 1024, 2048, 3072... etc.

Example B.5.7: DDL for an Array of External Device References

```
<property valuetype="network" array="16">
  <behavior name="deviceRef" set="acnbase.bset"/>
  <protocol name="ESTA.DMP">
    <propref_DMP
      loc="11"
      abs="true"
      inc="1"
      size="16"
      read="true"
      event="true"
    />
    <childrule_DMP
      loc="1024"
      inc="1024"
      abs="true"
    />
  </protocol>
</property>
```

Table B.1: Address space diagram

Address	Function
0 – 10	Available for other properties
11 – 26	Array of 16 DCIDs of sub devices
27 – 1023	Available for other properties
1024 – 2047	Address space for first sub device
2048 – 3071	Address space for second sub device
…	…
16384 – 17407	Address space for 16th sub device
17408 – 4294967295	Available for other properties or sub devices

B.5.8 Array Example

This very simplified example using fictitious behaviors (similar to some DMP behaviors) declares an array of 96 dimmers (intensity behavior). Each intensity is a driven value that is produced by an array of 10 Highest Takes Precedence inputs (HTPdriver behavior). Each HTP input also has a “preheat” level in a (minLimit behavior), however the preheat level is shared across all the HTP inputs for each dimmer and therefore there are only 96 rather than 960 preheat properties.

Relative to the location origin specified for the top level property, the 96 intensity properties are at addresses 0..95. The 960 HTP inputs are at 200..209, 220..229, 240..249, etc. The 96 preheat levels are at locations 219, 239, 259 etc.

Example B.5.8: DDL for an Array of Direct Device References

```
<property array="96" valuetype="network">
  <label>Dimmer Array</label>
  <behavior name="intensity" set="foo.bset" />
  <behavior name="HTPdriven" set="foo.bset" />
  <protocol name="ESTA.DMP">
    <propref_DMP event="true" inc="1" loc="0"
                 read="true" size="2" />
    <childrule_DMP inc="20" loc="200" />
  </protocol>
  <property array="10" valuetype="network">
    <label>HTP Array</label>
    <behavior name="HTPdriver" set="foo.bset" />
    <protocol name="ESTA.DMP">
      <propref_DMP inc="1" loc="0"
                   read="true" size="2" write="true" />
    </protocol>
    <property valuetype="network" xml:id="preheat"
sharedefine="arraycommon">
      <label>Preheat</label>
      <behavior name="minLimit" set="foo.bset" />
      <protocol name="ESTA.DMP">
        <propref_DMP loc="19"
                     read="true" size="2" write="true" />
      </protocol>
    </property>
  </property>
</property>
```

Appendix C. DTD for DMP Only

A much more complete DTD for use with the DMP protocol only is produced from the generic DDL version by adding declarations for the propref_DMP and childrule_DMP elements and tightening a few other declarations to restrict to one specific protocol.

The public identifier for this DTD shall be:

"-//ESTA//DTD Device Description Language 1.1//EN//DMP"

```
<!--
  DTD for Device Descritpion Language using ESTA DMP protocol

  Registered protocol name is "ESTA.DMP"
-->
<!ELEMENT DDL ((behaviorset | device | languageset))>
<!ATTLIST DDL
  version ( 1.1 ) #REQUIRED
  xml:id ID #IMPLIED
>
<!-- common module content -->
<!ELEMENT label (#PCDATA)>
<!ATTLIST label
  label.paramname NMTOKEN #IMPLIED
  set NMTOKEN #IMPLIED
```

```
  set.paramname NMTOKEN #IMPLIED
  key NMTOKEN #IMPLIED
  key.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT alternatefor EMPTY>
<!ATTLIST alternatefor
  UUID NMTOKEN #REQUIRED
  UUID.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>

<!ELEMENT extends EMPTY>
<!ATTLIST extends
  UUID NMTOKEN #REQUIRED
  UUID.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT UUIDname EMPTY>
<!ATTLIST UUIDname
  name NMTOKEN #REQUIRED
  UUID NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- languageset module -->
<!ELEMENT languageset (
  UUIDname*, label?,
  alternatefor*,
  extends*,
  language+
)>
<!ATTLIST languageset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- languageset content -->
<!ELEMENT language (label?, string*)>
<!ATTLIST language
  lang CDATA #REQUIRED
  altlang CDATA #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT string (#PCDATA)>
<!ATTLIST string
  key NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- behaviorset module -->
<!ELEMENT behaviorset (
  UUIDname*, label?,
```

```
  alternatefor*,
  extends*,
  behaviordef+
)>
<!ATTLIST behaviorset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- behaviorset content -->
<!ELEMENT behaviordef (label?, refines*, section+)>
<!ATTLIST behaviordef
  name NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!ELEMENT refines EMPTY>
<!ATTLIST refines
  set NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!ELEMENT section (hd?, (section | p)+)>
<!ATTLIST section
  xml:id ID #IMPLIED
>
<!ELEMENT hd (#PCDATA)>
<!ATTLIST hd
  xml:id ID #IMPLIED
>
<!ELEMENT p (#PCDATA)>
<!ATTLIST p
  xml:space (default | preserve) 'default'
  xml:id ID #IMPLIED
>
<!-- device module -->
<!ELEMENT device (
  UUIDname*, parameter*, label?,
  alternatefor*,
  extends*,
  useprotocol+,
  (property | propertypointer | includedev)+
)>
<!ATTLIST device
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!-- parameter declarations -->
<!ELEMENT parameter (
  label?,
```

```
  ( choice*
    | refinement*
    | ( mininclusive?, maxinclusive?)
  )
)>
<!ATTLIST parameter
  name NMTOKEN #REQUIRED
  xml:id ID #IMPLIED
>
<!--parameter restrictions -->
<!ELEMENT choice (#PCDATA)>
<!ATTLIST choice
  choice.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT mininclusive (#PCDATA)>
<!ATTLIST mininclusive
  mininclusive.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT maxinclusive (#PCDATA)>
<!ATTLIST maxinclusive
  maxinclusive.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT refinement (#PCDATA)>
<!ATTLIST refinement
  refinement.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!-- see below for useprotocol and other protocol dependent content -->
<!-- properties -->
<!ELEMENT property (
  label?,
  behavior+,
  value*,
  protocol*,
  (property | propertypointer | includedev)*
)>
<!ATTLIST property
  array CDATA #IMPLIED
  array.paramname NMTOKEN #IMPLIED
  valuetype (NULL | immediate | implied | network | common) #IMPLIED
  valuetype.paramname NMTOKEN #IMPLIED
  sharedefine (false | true | arraycommon) "false"
  sharedefine.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT behavior EMPTY>
<!ATTLIST behavior
  set NMTOKEN #REQUIRED
  set.paramname NMTOKEN #IMPLIED
```

```
  name NMTOKEN #REQUIRED
  name.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value
  value.paramname NMTOKEN #IMPLIED
  type (uint | sint | float | string | object) #REQUIRED
  type.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT propertypointer EMPTY>
<!ATTLIST propertypointer
  ref CDATA #REQUIRED
  ref.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!-- included devices and parameter instantiation -->
<!ELEMENT includedev (label?, protocol*, setparam*)>
<!ATTLIST includedev
  UUID NMTOKEN #REQUIRED
  UUID.paramname NMTOKEN #IMPLIED
  array CDATA #IMPLIED
  array.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT setparam (#PCDATA)>
<!ATTLIST setparam
  name NMTOKEN #REQUIRED
  setparam.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!-- Protocol dependent section -->
<!--
  useprotocol fixed for DMP
-->
<!ELEMENT useprotocol EMPTY>
<!ATTLIST useprotocol
  name ( ESTA.DMP ) #REQUIRED
  xml:id ID #IMPLIED
>
<!ELEMENT protocol ((propref_DMP?, childrule_DMP?) | propmap_DMX?) >
<!ATTLIST protocol
  name ( ESTA.DMP ) #REQUIRED
  xml:id ID #IMPLIED
>
<!ELEMENT propref_DMP EMPTY>
<!ATTLIST propref_DMP
  loc CDATA #REQUIRED
  loc.paramname NMTOKEN #IMPLIED
  abs (true | false) "false"
  abs.paramname NMTOKEN #IMPLIED
```

```
  inc CDATA #IMPLIED
  inc.paramname NMTOKEN #IMPLIED
  size CDATA #REQUIRED
  size.paramname NMTOKEN #IMPLIED
  read (true | false) "false"
  read.paramname NMTOKEN #IMPLIED
  write (true | false) "false"
  write.paramname NMTOKEN #IMPLIED
  event (true | false) "false"
  event.paramname NMTOKEN #IMPLIED
  varsize (true | false) "false"
  varsize.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!ELEMENT childrule_DMP EMPTY>
<!ATTLIST childrule_DMP
  loc CDATA #REQUIRED
  loc.paramname NMTOKEN #IMPLIED
  abs (true | false) "false"
  abs.paramname NMTOKEN #IMPLIED
  inc CDATA #IMPLIED
  inc.paramname NMTOKEN #IMPLIED
  xml:id ID #IMPLIED
>
<!-- End of protocol dependent section -->
```

Appendix D. Languagesets and Behaviors Defined for DDL

A base set of property behaviors is defined. These and their associated strings are specified in [acnbaseDDL]. These are a normative part of the specification of DDL for DMP. However, they are best read using XML presentation tools (for example a browser in conjunction with a stylesheet, by transformation into HTML using XSLT or using a suitable XML editor).

Although these behaviors are defined as part of DDL-DMP, they may be freely referenced by DDL using other protocols without change of UUID provided that they do not conflict with rules specific to those other protocols.

Appendix E. 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 [DMP]). DDL may be adapted to many different access protocols both within and outside of the ACN suite, and a single device may support multiple access protocols (see: The Access Protocol).

appliance
: In DDL an appliance is a piece of equipment described by a root device and all its children and descendents. In DMP systems [DMP] 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).

CID
: Short for Compinent Identifier. The ACN Architecture [Arch] requires that each component has a unique identifying CID which is a 128-bit Universally Unique Identifier [UUID].

component
: The process, program or application corresponding to a single ACN endpoint. All messages in ACN are sent and received by a component that is identified by a CID. See [Arch] for a more complete definition.

controller
: The term controller is often used loosely to refer to any piece of equipment that controls or monitors other equipment via the network. However, in the context of DMP a controller is defined precisely in terms of the messages implemented, while in DDL context a controller is defined by its use of device descriptions. Other protocols or contexts may have their own definitions. See controller (DDL).

controller (DDL)
: Within DDL a controller is a network entity that interprets the DDL descriptions of devices to know how to access or control them using the access protocol(s) of the Device Description to access each device.

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.

device (DDL)
: Within DDL, a device is a DDL module describing an entity that may be monitored and controlled by means of a network or datalink. In DDL there is no distinction

between a device and a sub-device except for the context in which they are encountered (device is a recursive term).

device class

The set of devices that are all described by the same device description.

module

In DDL elements behaviorset, device and languageset are defined to be modules. Each module must be wrapped by itself in a DDL element which constitutes a XML document. See sections 3.3, 4.4, 4.5, 4.6.

root device

An instance of a device described in DDL that has no parent device. See also appliance.

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).

Appendix F. References

Normative

[ACN] Entertainment Services and Technology Association [https://tsp.esta.org]. E1.17. Entertainment Technology – Architecture for Control Networks. The edition current when this Standard is approved.

[Arch] Entertainment Services and Technology Association [https://tsp.esta.org]. E1.17. Entertainment Technology – Architecture for Control Networks. “ACN” Architecture. The edition current when this Standard is approved..

[DDL] Entertainment Services and Technology Association [https://tsp.esta.org]. E1.17. Entertainment Technology – Architecture for Control Networks. Device Description Language. The edition current when this Standard is approved..

[DMP] Entertainment Services and Technology Association [https://tsp.esta.org]. E1.17. Entertainment Technology – Architecture for Control Networks. Device Management Protocol. The edition current when this Standard is approved..

[acnbaseDDL] Entertainment Services and Technology Association [https://tsp.esta.org]. urn:uuid:71576eac-e94a-11dc-b664-0017316c497d. ACN Base Behavior Set. As extended by the addition of behaviorsets that are identified as “Core Modiules” according to [CoreDDL] when this Standard is approved..

[ESTA-IDs] Entertainment Services and Technology Association [https://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 Standard is approved..

[CoreDDL] Entertainment Services and Technology Association [https://tsp.esta.org]. E1.17. Entertainment Technology – Architecture for Control Networks. EPI 22 Revised. DDL Core Modules for ACN Devices (Draft Standard). The edition current when this Standard is approved..

[ISO-DATE] International Standards Organisation [http://www.iso.org/]. ISO 8601. Data elements and interchange formats – Information interchange. Representation of dates and times. 2000.

[RELAXNG] Organization for the Advancement of Structured Information Standards (OASIS) [http://www.oasis-open.org/]. RELAX NG Specification. Committee Specification. 3 December 2001. http://relaxng.org/spec-20011203.html.

[LANG-TAG] Internet Engineering Task Force (IETF) [http://ietf.org/]. RFC 3066 [http://ietf.org/rfc/rfc3066.txt]. Alvestrand. Tags for the Identification of Languages. 2001.

[Unicode] The Unicode Consortium [http://www.unicode.org/consortium/consort.html]. The Unicode Standard [http://www.unicode.org/versions/Unicode5.0.0/], Version 5.0.0, defined by: The Unicode Standard, Version 5.0 (Boston, MA, Addison-Wesley, 2007. ISBN 0-321-48091-0)

[US-ASCII] American National Standards Institute [http://www.ansi.org/]. ANSI INCITS 4-1986 (R2007). Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII). 2007.

[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.

[XML] World Wide Web Consortium (W3C) [http://www.w3c.org/]. Extensible Markup Language (XML) 1.0 (The edition current when this Standard is approved.). W3C Recommendation. [http://www.w3.org/TR/xml/].

[XMLnames] World Wide Web Consortium (W3C) [http://www.w3c.org/]. Namespaces in XML. W3C Recommendation. 14 January 1999. http://www.w3.org/TR/REC-xml-names/.

[XSL] World Wide Web Consortium (W3C) [http://www.w3c.org/]. Extensible Stylesheet Language (XSL). W3C Recommendation. 15 October 2001. http://www.w3.org/TR/xsl.

[XSD] World Wide Web Consortium (W3C) [http://www.w3c.org/]. XML Schema Part 2: Datatypes. W3C Recommendation. 28 October 2004. http://www.w3.org/TR/xmlschema-2/.

[xml-id] World Wide Web Consortium (W3C) [http://www.w3c.org/]. xml:id Version 1.0. W3C Recommendation. 9 September 2005. http://www.w3.org/TR/xml-id/.

Informative

[DMX512] Entertainment Services and Technology Association [https://tsp.esta.org]. ANSI E1.11. USITT DMX512-A – Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories. The edition current when this Standard is approved..

[RDM] Entertainment Services and Technology Association [https://tsp.esta.org]. ANSI E1.20. Entertainment Technology – Remote Device Management over DMX512 Networks. The edition current when this Standard is approved..

[XInclude] World Wide Web Consortium (W3C) [http://www.w3c.org/]. XML Inclusions (XInclude) Version 1.0. W3C Recommendation. 15 November 2006. http://www.w3.org/TR/xinclude/.

[XPointer] World Wide Web Consortium (W3C) [http://www.w3c.org/]. XML Pointer Framework. W3C Recommendation. 25 March 2003. http://www.w3.org/TR/xptr-framework/.

[XPath] World Wide Web Consortium (W3C) [http://www.w3c.org/]. XML Path Language (XPath). W3C Recommendation. 16 November 1999. http://www.w3.org/TR/xpath.